Dünaamiliselt uuendatav 3D kaardi lahendus Iseautole

Magistritöö

Mirjam Feodorov
153673IAPM

Juhendaja:
Juhan-Peep Ernits, PhD

Tallinn 2019

Autorideklaratsioon

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

Mirjam Feodorov
06.01.2019

Annotatsioon

Tallinna Tehnikaülikooli100. sünnipäevaks valmis Eesti esimene isesõitev auto nimega Iseauto, mis vajab autonoomseks sõiduks 3D kaarte. Iseauto eripäraks on, et kaardistamiseks ja lokaliseerimiseks kasutatakse kahte kallutatud lidarit.

Käesoleva magistritöö eesmärgiks on leida dünaamiline lahendus 3D kaartide loomiseks kahe lidariga ja 3D kaartide töötlemiseks. Dünaamiline lahendus antud töö kontekstis tähendab, et kaarte saab vastavalt vajadusele jooksvalt uuendada ja neid saab kokku panna erinevatest osadest. Töös töödeldakse 3D kaarte järeltöötlusena, mitte raalajas.

Kahe kallutatud lidariga 3D kaardi loomise eelduseks on auto andurite asukohti iseloomustavate korrektsete transformatsioonide olemasolu, mida kasutatavas tarkvaras esitatakse puuna. Oluline on ka 3D kaardi loomisel tarkvaraga Autoware luua konfiguratsioonid, kus erinevaid parameetreid seadistatakse õiges järjekorras. Kaardistamise lihtsustamiseks töötasime välja lahenduse 3D kaartide automaatseks loomiseks.

Kahe erineva kaldega lidarite andmetest loodud 3D kaardid ei ole automaatselt kattuvad ja seetõttu on vaja 3D kaarte töödelda. Antud töös pakutakse välja lahendus, kuidas 3D kaarte registreerida, liita, osadeks lõigata, müra filtreerida ning kuidas kaartidelt eemaldada staatiline vananenud info.

Töödeldud 3D kaartide kvaliteedi hindamiseks kasutatakse Autoware lokaliseerimise algoritmi NDT_matching veahinnangut. Töö tulemusena valmis lahendus, mis võimaldab luua kaarte, kus lokaliseerimine kahe kallutatud lidariga töötab stabiilselt. Lahendust välja pakkudes püstitasime hüpoteesi, et töödeldud kaardid võimaldavad täpsemat lokaliseerimist, kui töötlemata kaardid. Lahenduse kontrollimisel reaalses keskkonnas tehtud mõõtmistele tuginedes selgus, et töödeldud 3D kaardi veahinnang oli väiksem kui töötlemata 3D kaardi veahinnang, s.t. loodud lahendus võimaldab täpsemat lokaliseerimist..

Magistritöö on kirjutatud Eesti keeles ning sisaldab teksti 54 leheküljel, 4 peatükki, 18 joonist ja 3 tabelit.

Abstract

For the 100th anniversary of the Tallinn University of Technology, a first self-driving car built in Estonia called Iseauto was developed. Iseauto has two lidars for mapping and localization that have been tilted forward and sideways to improve coverage at close range.

The goal of the current work is to find a dynamical solution for 3D mapping with two lidars and for the maintenance 3D maps.

The prerequisite for creating a 3D map with two tilted lidars is the presence of the correct transformations that are represented in the form of tree in the software used. To simplify mapping, the 3D map making process is automated.

As two lidars are tilted to opposite sides, the 3D maps created from the data of the different lidars are not automatically overlapping and therefore it is necessary to process the 3D maps. In this work, a solution is proposed for registrating, merging, splitting, noise filtering and removing outliers from 3D maps.

To assess the quality of the processed 3D maps, we used a score attribute from the Autoware localization algoritm NDT_matching. The expected result was that the processed 3D maps have smaller score than unprocessed 3D maps. The hypothesis was confirmed by experiments using real world data.

The thesis is written in Estonian and contains 54 pages, 4 chapters, 18 figures and 3 tables.

Sisukord

Jooniste loetelu

Tabelite loetelu

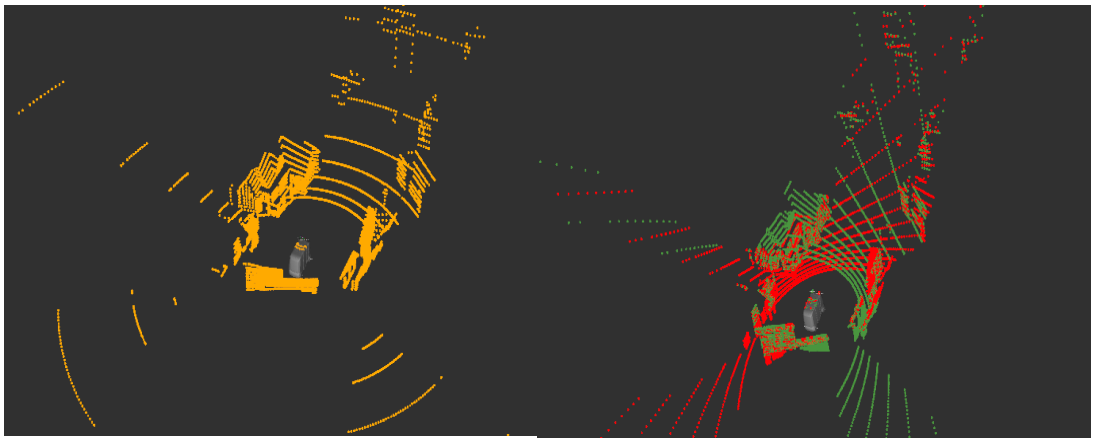
Sissejuhatus

Tallinna Tehnikaülikooli 100. sünnipäevaks valmis 2018. aastal Eesti esimene isesõitev auto nimega Iseauto. Iseauto loomisele aitasid kaasa nii TTÜ erinevate teaduskondade tudengid kui õppejõud, koostööd tehti AS-iga Silberauto.

Isesõitvate autode arendus on muutunud maailmas üha laialdasemaks ja 3D kaardid moodustavad sellest arendusest tähtsa osa. Isesõitvad autod vajavad 3D kaarte navigeerimiseks ja lokaliseerimiseks.

Kuna väline keskkond on pidevas muutumises, tuleb uuendada ka 3D kaarte – nendelt tuleb eemaldada dünaamilised objektid, nagu näiteks inimesed, autod, bussid, veokid.
Aja jooksul võivad muutuda ka staatilised objektid, näiteks ehitatakse uusi maju, püstitatakse teeserva aed või mõni hoone hoopis lammutatakse. Olemasolevatel 3D kaartidel tuleb sel juhul infot uuendada. 3D kaartide kaasajastamine on eelduseks, et isesõitev auto saaks planeerida teekonda ja autonoomselt navigeerida. Kui dünaamilised objektid jääksid 3D kaardilt eemaldamata, siis ei suudaks isesõitev sõiduk mõne aja möödumisel enam navigeerida. Lisaks kannatab lokaliseerimise täpsus [1].

Käesolevas töös otsime lahendusi, kuidas Iseauto lidareid kasutada nii kaardi loomiseks kui ka navigeerimiseks. Oluliseks küsimuseks on kaartide värskendamine ning suure kaardi väiksematest osadest kokkupanek. Iseautol on lidarid paigutatud kallutatuna nii ette kui ka küljele, et paremini katta ära auto lähiümbrus. Selline lidarite paigutus on valitud turvalisuse kaalutlusel. Lidareid on pööratud nii, et nad näeksid võimalikult hästi auto ette ja seega on võimelised paremini tuvastama takistusi. Joonis 1 illustreerib kahe kallutatud lidari eelist ühe lidari ees auto lähiümbruse aspektist lähtudes. Sellise kallutatuse miinuseks on aga see, et lidarid ei näe auto ümbrust sümmeetriliselt. Lisaks oleme avastanud loodud 3D kaartidel moonutusi: vertikaalsed seinad on kaartidel viltused ning loodud 3D kaardid on kõverad.



(a) (b)

Joonis 1. (a) Ühe 16-kiirega lidari punktipilv. (b) Kahe ette ja kõrvale kallutatud 16-kiirega lidari punktipilv. [2]

Uurimuses [2] toodi tulemusena välja, et kahe kallutatud lidariga saab lokaliseerida küll, kuid olulist rolli määrab 3D kaardi kvaliteet.

Käesolev töö otsib vastust järgmistele küsimustele:

- Kuidas luua 3D kaarte kahe lidariga nii, et kahe lidariga lokaliseerimine toimiks hästi?
- Kuidas 3D kaartide loomist automatiseerida?
- Kuidas teisendada 3D kaarte Eesti ristkoordinaatidele?

Käesoleva töö põhieesmärgid on:

- Leida meetod, kuidas töötlemata 3D kaartidelt eemaldada moonutused ja viia kaardid vastavusse pärismaailmaga.
- Leida lahendus, kuidas lidari tekitatud 3D kaardi punktid transformeerida Eesti ristkoordinaatideks.
- Realiseerida lahendus ja automatiseerida see.
- Kontrollida, kas loodud lahendus muudab 3D kaardid kvaliteetsemaks.

3D kaardi kvaliteedi hindamine on keerukas. 3D kaardi kvaliteedi hindamise probleeme on käsitletud ka kirjanduses [3]. Käesolevas töös defineerime loodavate kaartide rakendusest lähtuva kvaliteedi kriteeriumi, milleks on kaardil lokaliseerimise täpsus.

Seega kasutame 3D kaardi kvaliteedi kriteeriumiks Autoware [4] lokaliseerimise algoritmi NDT matching veahinnangut. Mida väiksem on NDT matching veahinnang, seda parem. Oodatav tulemus on see, et uuendatud 3D kaardi lokaliseerimise veahinnangud on väiksemad, kui töötlemata 3D kaardil.

3D kaardi kvaliteeti saab hinnata ka selle suutlikkusega ennast taasesitada [5]. Kaardi visuaalsel vaatlusel on võimalik kvalitatiivselt hinnata, kas näiteks maja seinad on sirged. Samuti võrreldakse geograafiliste koordinaatidega kaarti Maa-ameti ortofotode suhtes. Oodatavaks tulemuseks on, et kaardid kattuvad.

Tulenevalt Iseauto projekti nõuetest, peab lahendustes kasutatav tarkvara olema vabavara.

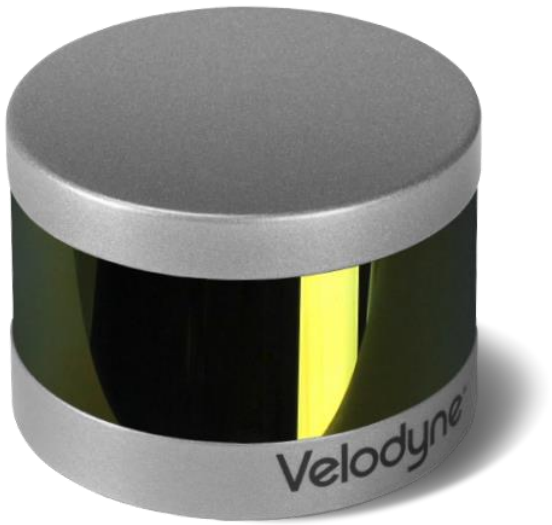
Magistritöö koosneb neljast peatükist, millest esimeses peatükis antakse ülevaade kaardistamise meetoditest ning kasutatavast riist- ja tarkvarast. Teises peatükis on välja toodud seotud tööd. Kolmandas peatükis kirjeldatakse 3D kaardi loomise protsessi ja kuidas toimub 3D kaartide järeltöötlus. Neljandas peatükis analüüsin ja hindan tulemusi läbi lokaliseerimise täpsuse.

1 Taust

Isesõitvad autod saavad informatsiooni ümbritseva keskkonna kohta andurite kaudu. Anduriteks võivad olla lidarid, globaalne navigatsiooni satelliitide süsteem (GNSS), inertsiaalne mõõteseade (IMU), radarid, kaamerad vms. Andurid ise aga otsuseid ei tee, nemad ainult annavad informatsiooni. Otsuseid teeb spetsiaalne tarkvara, mis analüüsib anduritelt saadud infot. Iseauto tarkvaralahendus põhineb vabavaralisel tarkvaral Autoware, mis on mõeldud autonoomsete robotite juhtimiseks.

1.1 Lidar

3D kaartide loomiseks ja lokaliseerimiseks kasutatakse seadet nimetusega lidar.
Lidar on valgusimpulsse välja saatev tüüpiliselt pöörlev laser. Valgusimpulsid peegelduvad tagasi, kui jõuavad objektini ning punkti kaugust on võimalik tagasipeegeldumiseks kulunud aega mõõtes täpselt hinnata. Igale punktile omistatakse x-, y-, z-koordinaat, ja peegeldumisintensiivsus. Antud töös kasutatakse lidarit Velodyne VLP-16 (Joonis 2). Antud lidari töötamise ulatus on 360°, tal on 16 vertikaalset saatja-vastuvõtja paari ja ta suudab mõõta üle 300000 punkti sekundis.



Joonis 2.Velodyne VLP-16 [6]

1.2 3D kaardi loomise meetodid

Lidariga salvestatud algandmetest saab luua 3D kaardi kasutades kaardistamise algoritme. Allikas [7] tuuakse välja kaks juhtivat 3D kaardistamise algoritmi – ICP ja NDT. Iteratiivne lähima punkti (ICP) algoritm minimeerib iteratiivselt punktide vahelist vahemaad kahe skaneerimise vahel [7]. Normaaljaotuse teisenduse (NDT) algoritm esitab vaadeldavate punktide kogumikke kui mingi lidarikiirt tagasi peegeldava objekti vastavas punktis esinemise tõenäosuse normaaljaotust. NDT algoritm taasesitab keskkonda sõrestikuna. Idee on kõigepealt ruum jagada alamruumideks ja seejärel arvutada mõõtmise keskväärtus ning korrelatsioonmoment iga alamruumi jaoks. Seega on NDT kaart normaaljaotuste kogumik, mis annavad tõenäosuse, et punkt on mõõdetud konkreetsel füüsilisel asukohal [5]. [8]-s tuuakse välja, et NDT annab madala resolutsiooni korral parima tulemuse.

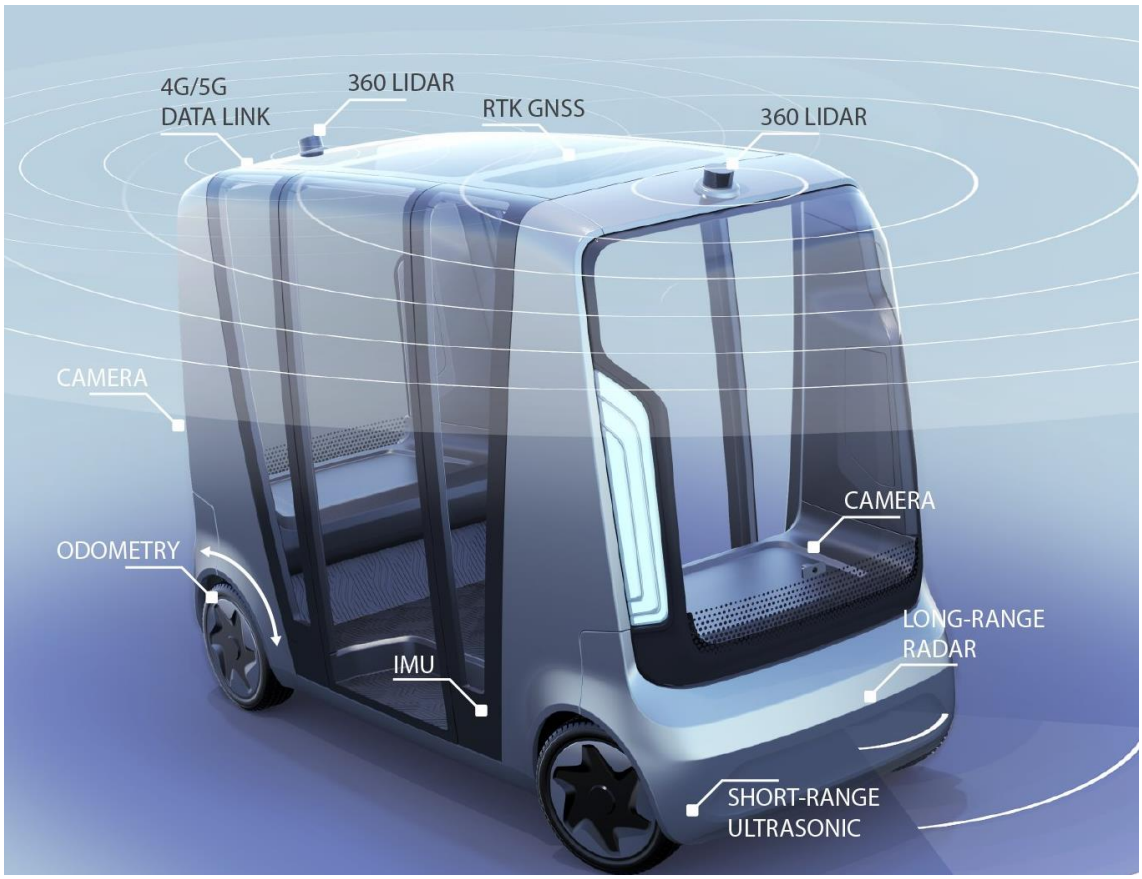
Vastavalt Autoware publikatsioonile [9] eelistatakse Autoware kasutamisel peamiselt NDT algoritmi. Seda andmete töötlemise kiiruse tõttu ja kuna NDT algoritmi arvutuslik hind ei domineeri kaardi suuruse üle, siis saab luua suurema resolutsiooniga 3D kaarte. Lisaks on NDT enamikel juhtudel täpsem ja töökindlam kehva algasendi hinnangu korral [10].

1.3 TTÜ iseauto

Iseauto [11] on esimene isesõitev auto Eestis, mis on arendatud TTÜ ja Silberauto AS koostöös. Iseauto on nn viimase miili buss, mis on mõeldud opereerima TTÜ linnakus. Iseautol on elektrimootor võimsusega 47 kW ja maksimaalne liikumiskiirus on 20 km/h. Iseauto töötab tarkvaraga Autoware.

[11] loetleb Iseauto sensorid:

- Kaks Velodyne VLP-16 lidarit bussi katusel;
- Kaheksa ultraheli andurit;
- Üks lidar bussi ees allosas;
- Lühimaa radar ees;
- Kaamerad;
- RTK-GNSS;
- IMU.



Joonis 3: Iseauto sensoorika [11]

1.4 Autoware

Iseauto tarkvaralahendus põhineb vabavaralisel avatud lähtekoodiga platvormil Autoware, mis omakorda põhineb ROS-il. Autoware võimaldab arendada isesõitvaid masinaid linnaoludes ja pakub selle jaoks täielikku komplekti tarkvarakomponente, sh ka kasutajaliidest. Autoware põhifunktsioonid on järgmised:

- 3D lokaliseerimine, mis saavutatakse 3D kaartide, SLAM algoritmi, GNSSi ja IMU andurite abil;
- 3D kaardistamine;
- Takistuse tuvastamine, milleks kasutatakse kaamerate ja lidarite andmeid;
- Planeerimine ja prognoosimine, mille käigus tegeletakse teekonna ja liikumise planeerimisega;
- Sõiduki juhtimine, mille käigus Autoware edastab sõidukitele infot näiteks kiiruse ja nurkkiiruse kohta;

1.5 ROS

ROS (Robot Operating System) on vahetarkvara, mis töötab Linux operatsioonisüsteemis. Antud töös kasutatakse ROS Kinetic versiooni.
ROS pakub robotite tarkvara arendamiseks sobivaid teeke ja tööriistu, millest põhilisemad on :

- Esialgne ehitussüsteem (Catkin);
- Pilditöötluse teek (OpenCV);
- Visualiseerimise tööriist (RViz);
- Koordinaatide teisendamise teek (TF);
- Andmete logimise tööriist (ROSBAG);
- Graafilise kasutajaliidese arendamise tööriist (RQT).

ROS kasutab sõnumite edastamiseks kanaleid (ingl.k topic). Saatja võib infot saata ühte või mitmesse vastuvõtjasse. Sõlm (ingl.k node) kirjutab sõnumeid kanalitesse ja teine sõlm saab neid sõnumeid sealt lugeda. Kõiki kanalites liikuvaid sõnumeid saab salvestada faili (rosbagi), kus sõnumitele lisatakse ajatempel.

1.6 Point Cloud Library

3D kaarte kutsutakse ka punktipilvedeks (ingl.k point cloud). Point Cloud Library (PCL) on vabavaraline raamistik 2D/3D piltide ja punktipilvede töötluseks. PCL sisaldab hulgaliselt algoritme, mis võimaldavad nii filtreerimist, segmenteerimist, pinna rekonstrueerimist, funktsioonide hindamist kui sisendite omavahelist liitmist. PCL algoritme saab näiteks kasutada müra filtreerimiseks, 3D kaartide liitmiseks, oluliste objektide segmenteerimiseks, punktipilvede visualiseerimiseks jpm [12].

1.7 PCD failiformaat

Point Cloud Data (PCD) on üks punktipilve failiformaatidest ja antud töös on kõik 3D kaardid vastava formaadiga. PCD võimaldab salvestada ja töödelda organiseeritud punktipilve andmekogusid. Toetatud on kõik põhilised andmetüübid – int, float, double, char, short [13]. Üks PCD eeliseid on see, et saab ise määrata, mis andmetüübiga fail salvestatakse – kas ascii- või binaarkujul. Töödeldud 3D kaardid salvestame antud töö

raames binaarkujul, kuna see on kiireim võimalik viis andmete laadimiseks ja salvestamiseks.

1.8 Cloud Compare

Cloud Compare on avatud koodiga vabavaraline tarkvara, mis võimaldab 3D kaarte töödelda. Erinevalt PCL-st on Cloud Compare-il olemas kasutajaliides, mis võimaldab käsitsi 3D kaarte töödelda juhul, kui automaatsed protsessid ei anna soovitud tulemusi. Funktsionaalsus on sarnane PCL-iga, olemas on algoritmid nii 3D kaartide liitmiseks, müra filtreerimiseks, statistika arvutamiseks, interaktiivseks või automaatseks segmenteerimiseks jpm [14]. Cloud Compare tarkvara jaoks on olemas ka korralik dokumentatsioon.

2 Seotud tööd

Kuna autonoomsete sõidukite temaatika on aktuaalne ja populaarne ning 3D kaardi kaasajastamine on vajalik protsess, et sõiduk turvaliselt ning täpselt saaks sõita, siis on rohkelt erinevaid lahendusi 3D kaartide uuendamiseks välja pakutud. Peamiselt pakutakse välja lahendusi, kuidas eemaldada 3D kaartidelt dünaamilisi objekte - [15], [1]. [15] esitleb algoritmi, mis teostab reaalajas SLAM-i, kaardi uuendamist, 3D kaardi punktide klassifitseerimist kas staatiliste või dünaamilistena ning dünaamiliste punktide kiiruse hindamist.

Iseauto projektiga sarnaste sensoritega (odomeeter, IMU, GPS ja Velodyne lidar) uurimus [16] ühendab andurite andmed ja pakub välja kaardistamise meetodi erinevate keskkondade jaoks.

Siiski on vähe leida töid, mis pakuksid lahendusi kaartide korrashoiuks, järeltöötluseks ja erinevatel ajahetkedel salvestatud kaartide omavaheliseks liitmiseks nii, et tekiks üks kvaliteetsem kaart. Ilmselt on tegemist suuresti ärisaladusega ja seda infot ei taheta avalikustada. Ka Iseauto projektis kasutusel oleval muidu vabavaralisel Autoware-l on 3D kaardist vektorkaardi loomise moodul ning mitme 3D kaardi sidumise moodul antud tasulisele rakendusele MapTools [17]. 3D kaardi parandamise ja uuendamise moodul puudub Autowares täielikult.

Iseauto eripäraks võrreldes teiste isesõitvate autodega on kahe lidari olemasolu, mis on lisaks veel kallutatud ettepoole ja küljele. Kahe kallutatud lidariga on ka Ford Fusion Hybrid autonoomne uurimisauto [18], aga need lidarid on suunatud külgedele, et saada parem ülevaade auto kõrval olevast ruumist. Lisaks on Fordi autonoomsel uurimisautol veel kaks horisontaalselt sirget lidarit. Kahjuks uurimistöid ja tehnilist dokumentatsiooni Fordi autonoomse uurimisauto kohta saadaval ei ole.

Kuna Iseautol on kaks lidarit, tuleb mõlema lidari poolt loodud 3D kaardid kohakuti viia, et saada üks detailsem 3D kaart. Kaartide kohakuti viimise protsessi nimetatakse registreerimiseks. Registreerimise algoritm seob 3D kaardid ühtsesse koordinaatsüsteemi minimeerides joondusviga. [19] toob välja, millised olukorrad parandavad ja millised halvendavad autonoomsete sõidukite 3D kaartide registreerimist. Olukorrad, mis on

värvitud punasega, on kriitilised. Kollased olukorrad on olulised ja rohelised vähem tähtsad.

Mõjur	Registreerimist soodustav	Registreerimist kahjustav
Keskkond	Hooned, teed	Puud
Tegevuspaik	Liiklust pole	Mitu teed
Vähene piirkondade kattuvus		Kiiresti liikuv sõiduk koos madala skaneerimiskiirusega
Muutuv kattuvus		Kiiruse suur kõikumine
Reaalajas registreerimine	Madal skaneerimisekiirus, hea hinnang liikumisele, Täpsed lidarite asendid auto suhtes määratud	Suur kaart
Algse transformat-siooni müra	Täpne odomeetria	

Tabel 1. Autonoomsete sõidukite 3D kaartide registreerimisele soodustavalt ja kahjustavalt mõjuvad tegurid.

3 3D kaardi loomine

3.1 Kaardistamise eeltööd

Lidariga salvestatud info kogutakse rosbagi, mis on ROS-i sõnumite logimise formaat. Rosbage saab hiljem kasutada andmete taasesitamiseks. Samasse rosbagi saab salvestada kõikide publitseeritud kanalite andmeid [20].

Iseauto kõigi kolme lidari andmed edastatakse erinevatesse ROS kanalitesse. Käesolevas töös kasutame ainult kahe katusel oleva lidari andmeid. Kolmas lidar, mis asub auto esiotsa alumises osas, on mõeldud takistuste tuvastamiseks, mitte kaardistamiseks, ja tema salvestusi me ei kasuta. Vasaku lidari andmed publitseeritakse kanalisse `/lidar_left` ja parema lidari andmed kanalisse `/lidar_right`. Mõlema lidari andmed on kokku kogutud kanalisse `/lidar_top`.

Lidari paiknemine sõiduki ja teiste lidarite suhtes määratakse projektsioonidega, mida ROSi kontekstis nimetatakse transformatsioondeks ehk TF-deks. TF on ROS-is koordinaadistike teisendamise ehk transformatsiooni teek, mis aitab vahendada koordinaatide raamistike omavaheliste suhete infot. Raamistik on koordinaatide süsteem, mis on ROS-is enamasti kolmedimensionaalne. TF-e kirjeldatakse puustruktuurina, seega saab igal raamistikul olla ainult üks vanem. Baaslüli on TF puu lähtekoht.

3D kaardi loomiseks sobiva TF puu välja töötamine ei olnud lihtne ülesanne. Autoware dokumentatsioon on selles osas puudulik. Töö käigus selgus, et õige TF puu loomine on võtmetähtsusega nii 3D kaartide loomise kui lokaliseerimise juures. Sellest, mis probleemid tekivad, kui TF puu ei ole korrektne, kirjutan lähemalt alapunktis 4.3. Kuna dokumentatsioon puudus, siis õnnestus alles peale logide ning ndt_mappingu [21] ja ndt_matchingu [22] koodi lugemist aru saada, milline TF puu luua. Selgus, et tarkvara eeldab, et lidarid on paigutatud horisontaalselt. Kuna Iseautol on lidarid kaldus, siis saab tarkvara aru sellest nii, et auto on viltu. Lisaks eeldab tarkvara, et põhilidari asukoht ja kaldenurk tuleb määrata baaslüli ja `velodyne` raamistiku vahelise TF-ga, sõltumata lidari andmeid sisaldava kanali nimest. Baaslüli raamistik on `velodyne` raamistiku

vanem ja lidarite kanalite raamistikud on velodyne raamistiku lapsed. Baaslüli raamistiku nimi on base_link .Vastav TF puu on kujutatud Joonis 4.

Kaardi loomisel anname TF-d kaasa käivitusfailides. Käivitusfail võimaldab käivitada üheaegselt mitu ROS sõlme või seadistada sõlmede parameetreid. Käivitusfailid kirjutatakse keeles XML.

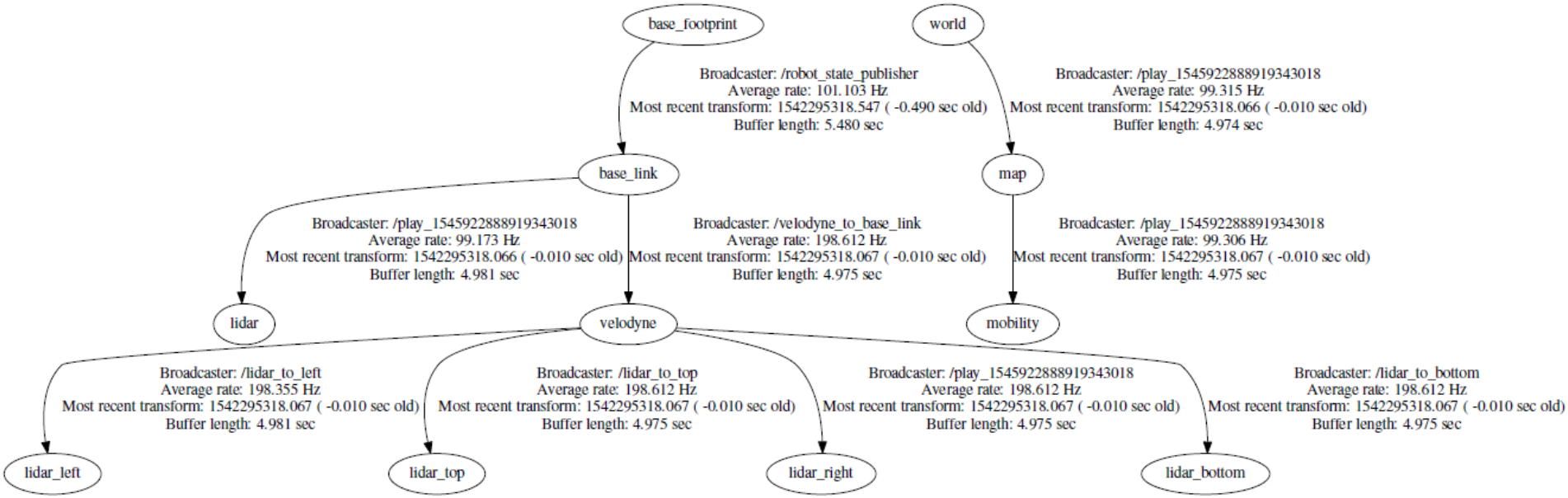
Koodinäites 1 on määratud lidarite asukohad teineteise ja velodyne raamistiku suhtes ning velodyne ning base_link suhe. Vasaku lidari asukoha määrasime auto asendi järgi garaažis. Parema lidari koordinaatide leidmiseks kasutasime Autoware teeki Multi Lidar Calibrator, kus määrasime põhilidariks vasaku lidari ja parem lidar jäi laps-lidariks. Seejärel sai parema lidari koordinaadid määrata vasaku lidari suhtes. Raamistikus on xyz koordinaadid ja koordinaatide nurgad radiaanides.

Velodyne suhtes on kirjeldatud järgmised koordinaatide raamid: lidar_top, lidar_left, lidar_right, lidar_bottom. Kuna vasak lidar on põhilidariks, siis kasutatakse tema koordinaate velodyne ja base_link vahelises TF-s. Seega on lidar_left ja velodyne vaheline TF nullid st need on samaväärsed. Sama kehtib ka raamistike lidari_left ja lidar_top TF-i kohta.

```
<launch>
    <arg name="use_tf_static" default="false"/>
    <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" >
          <param name="use_tf_static" value="$(arg use_tf_static)"/>
    </node>
    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_base_link" args="2.6 0.54 2.2 0.115 0.12 -0.03
/base_link /velodyne 10" />

    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_top" args="0 0 0 0 0 0 velodyne lidar_top 10" />

    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_left" args="0 0 0 0 0 0 velodyne lidar_left 10" />

    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_right"  args="-0.121917   -1.08513 -0.0630073
2.92476  3.10318 -3.00629 velodyne lidar_right 10" />
```

```
    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_bottom" args="0.614426 -0.499121  -1.15677
0.0174392 -0.0110698  0.0387979 velodyne lidar_bottom 10" />


    <group ns="lidar_top">
        <include file="$(find
points_preprocessor)/launch/points_concat_filter.launch">
            <arg name="output_frame" value="lidar_top" />
            <arg name="input0" value="/lidar_left/points_raw" />
            <arg name="input1" value="/lidar_right/points_raw" />
            <arg name="output" value="points_raw" />
        </include>
    </group>

</launch>
```

Koodinäide 1. Käivitusfail sensing4mapping.launch



Joonis 4. TF puu kaardistamiseks

3.2 Kaardistamise protseduur

3D kaartide loomiseks vajalikud toimingud Iseauto tarkvara keskkonnas on järgmised:

1. Seadistada keskkond vastava haru `setup.sh` kestaskriptiga.

2. Avada Autoware käsuga `./Autoware/ros/run`.

3. Varasemalt salvestatud sensorite andmete taasesitamiseks on vaja sünkroniseerida rosbagi ajatemplid ROS süsteemiga käsuga
   `rosparam set use_sim_time true`

4. Kuna tarkvara vaikimisi eeldab, et andmed tulevad kanalist `/points_raw` ning lisaks me soovime teha eraldi 3D kaardi faili mõlema lidari jaoks, tuleb kanali nimi ümber nimetada. Rosbagi maha mängimine vasaku lidari andmetega:

```
rosbag play file.bag /lidar_left/points_raw:=/points_raw
```

   a. Kui rosbagi on salvestatud ka andmed TF-de kohta, siis võib olla vajalik rosbagis sisalduva TF infot sisaldava kanali ümbernimetamine täiendava parameetriga: `/tf:=/tf_notused`.

5. Rosbagi taasesitus tuleb peale paari sekundit mängimist panna pausile, et kuulavate kanaliteni jõuaks info, et sõnumid võivad järgneda. Vastasel juhul ei pruugi kuulavad kanalid saada esimesi avaldatud sõnumeid.

6. Autowares määrame Vehicle Model lahtris automudeli URDF failiga.

7. Sensorite käivitusfail Koodinäide 2 käivitada käsuga
   `roslaunch sensing4mapping.launch`

8. Ndt_mapping käivitusfail aktiveerida. Seda saab ka teha Autoware kasutajaliideses.

9. Rosbag mängimisel paus lõpetada.

10. Võib avada visualiseerimise tööriista RVIZ, et näha kaardi tekkimist.

11. Kui ndt_mapping on rosbagi töötlemise lõpetanud, tuleb loodud PCD fail salvestada. Seda saab teha kasutades PCD_OUTPUT funktsiooni Autoware ndt_mappingu rakenduses.

Oluline on, et kõik eelnevalt kirjeldatud toimingud tehakse eelnevalt kirjeldatud järjekorras. Vastasel juhul 3D kaardi loomine ei õnnestu.

Kui kõik eelnev on tehtud, siis on meil olemas vasaku lidari 3D kaart failiformaadiga PCD. Sama tuleb korrata ka parema lidari jaoks, muutes punktis 4 kanali nime vastavalt `/lidar_right`. Kuigi meil on samas rosbagis olemas nii parema kui vasaku lidari salvestused, siis mõlema lidari andmeid üheaegselt kaardi loomiseks kasutada ei saa, tulemuseks on kasutamiseks kõlbmatu 3D kaart. Põhjustest kirjutan täpsemalt alapunktis 3.3.

3.2.1 Kaardistamise automatiseerimine

Käesoleva magistritöö üheks eesmärgiks oli 3D kaardi loomise protsessi automatiseerimine. See võimaldab vajadusel kohe peale rosbagide loomist kaardistamise protsess käivitada ilma, et kasutaja peaks täpselt teadma seadistuse spetsiifikat või vajalike toimingute järjekorda. Lisaks võimaldab kaardistamise automatiseerimine luua 3D kaarte uutest kohtadest juba autos.

3D kaardi automaatseks loomiseks on kaks faili:

- Kestaskript ndt_mapping.sh
- Käivitusfail ndt_mapping.launch

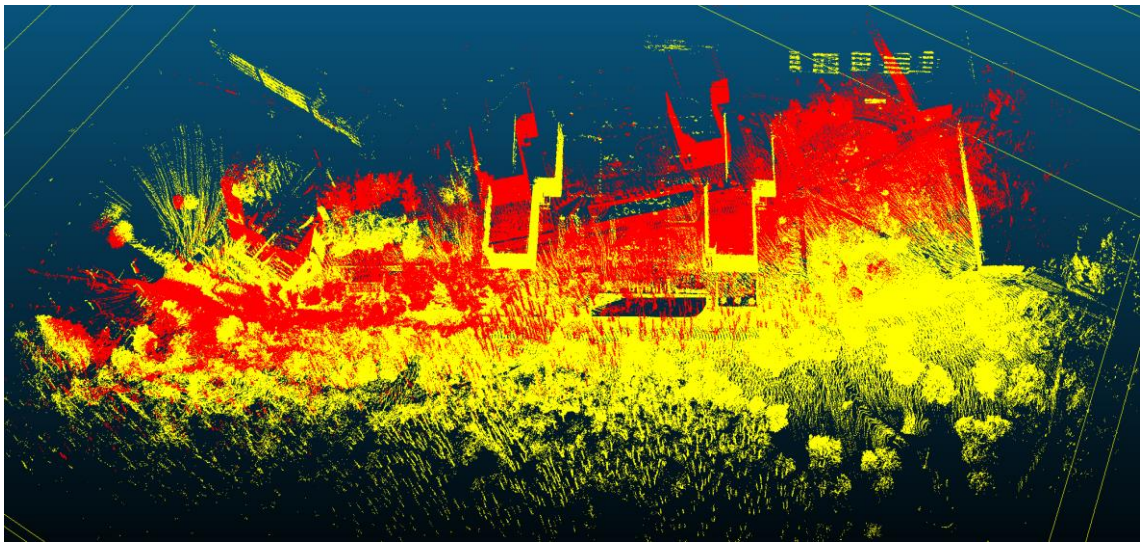
Kaardistamise protseduur kutsutakse välja kestaskriptiga, mis omakorda käivitab käivitusfaili. Käivitusfail aktiveerib kaardistamise protsessi. Muutujana saab ette anda rosbagi, mille andmetest 3D kaarti looma hakatakse. Kestaskript jälgib, et millal kaardistamise protseduur lõppeb ja see järel salvestab 3D kaardi PCD failina.
Käivitusfail ndt_mapping.launch on Lisas 1 ja kestaskript ndt_mapping.sh on Lisas 2.

3.3 3D kaartide töötlus

Kui 3D kaardid on olemas, siis esimene samm on kõrvalekallete eemaldamine kaardilt. Selleks kasutan PCL tööriista kõrvalekallete eemaldamise statistilist meetodit `pcl_outlier_removal`, mis põhimõtteliselt on iga punkti ümbruse statistiline analüüs. Igale punktile arvutatakse välja keskmine kaugus tema naabrite suhtes. Eeldades, et saadud jaotus on normaaljaotus keskmise- ja standardhälbega, siis eemaldatakse need punktid, mille keskmine kaugus on väljaspool keskmist- ja standardhälvet [23]. Kõrvalekallete eemaldamine on vajalik, et hiljem mitme 3D kaardi registreerimine ja seega ka lokaliseerimise oleks täpsem.

Järgmine samm on 3D kaartide töötlemine nii, et need oleksid võimalikult sarnased pärismaailmaga. Ettepoole viltu asetsevad lidarid on küll kasulikud eesolevate takistuste tuvastamisel, aga kaardistamisel tekitavad nad hulgaliselt probleeme. Keerukust lisab ka asjaolu, et Iseautol on kaardistamiseks ja ka lokaliseerimiseks kaks lidarit.

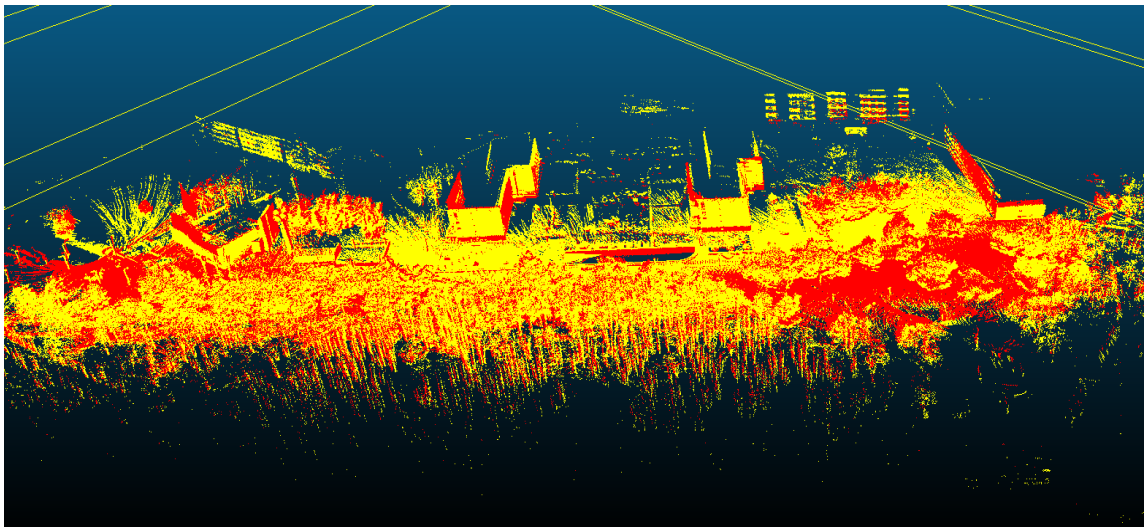
Joonis 5**Error! Reference source not found.** on kujutatud kahe kallutatult paigutatud lidari tõttu tekkinud probleem – parema ja vasaku lidari poolt tehtud 3D kaardid kaugenevad teineteisest mida kaugemale algpunktist liigutakse. Et saada keskkonnast detailsem 3D kaart, soovime kasutada mõlema lidari andmeid. Selleks, et 3D kaardid kohakuti viia, tuleb kasutada registreerimise algoritmi, mis seob 3D kaardid ühtsesse koordinaatsüsteemi minimeerides joondusviga.



Joonis 5. Kollane 3D kaart on tehtud vasaku lidari poolt, punane 3D kaart parema lidari poolt sama testsõidu käigus. Et demonstreerida objektide nihet, on vaade ülalt.

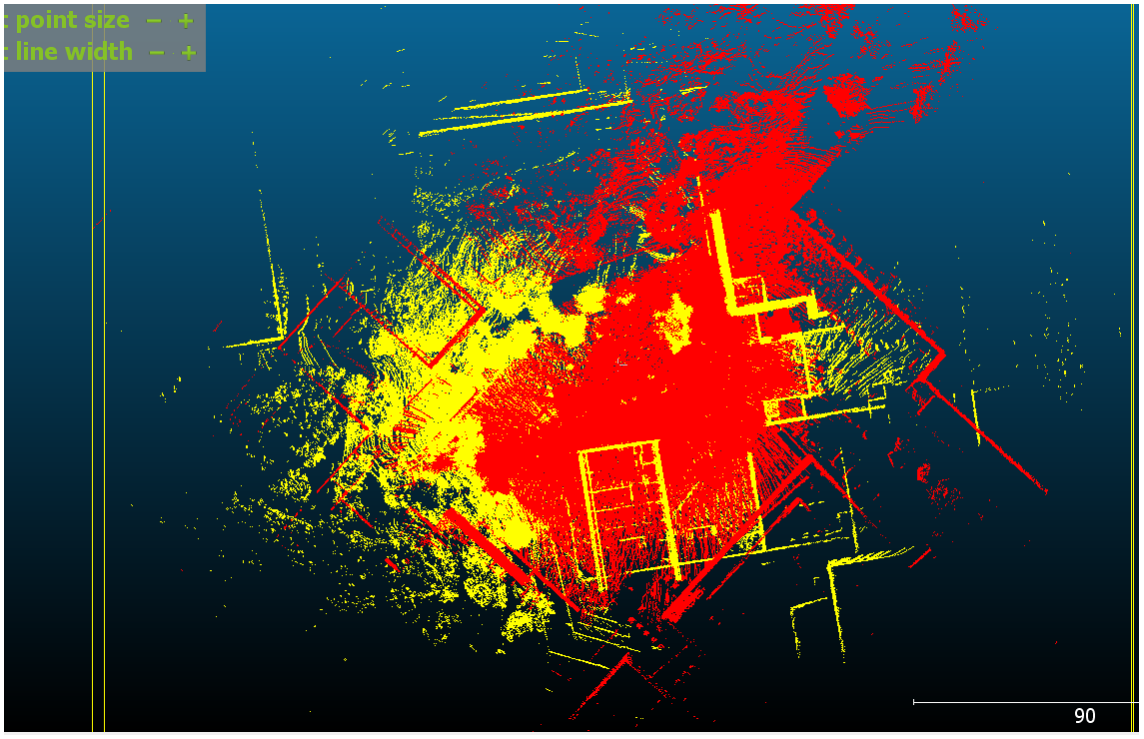
Juhul kui kaardid on suures osas kattuvad, siis on võimalik kaardid registreerida, kasutades programmi Cloud Compare ICP põhist algoritmi. ICP on iteratiivne protsess, mille käigus registreerimisviga järjest vähehaaval väheneb. Enne registreerimist tuleb ära määrata, millist 3D kaarti liigutatakse (ingl.k aligned) ja kumb jääb paigale (ingl.k reference). Cloud Compare võimaldab ICP registreerimisel ka anda ette maksimaalse iteratsioonide arvu või registreerimisvea alammäära, mille korral registreerimisprotsess peatatakse, kui erinevuse kahe iteratsiooni vahel on väiksem kui registreerimisvea alammäär.

Joonis 6 on näidatud ICP registreerimise tulemus, kus registreerimisviga oli 1,3268.



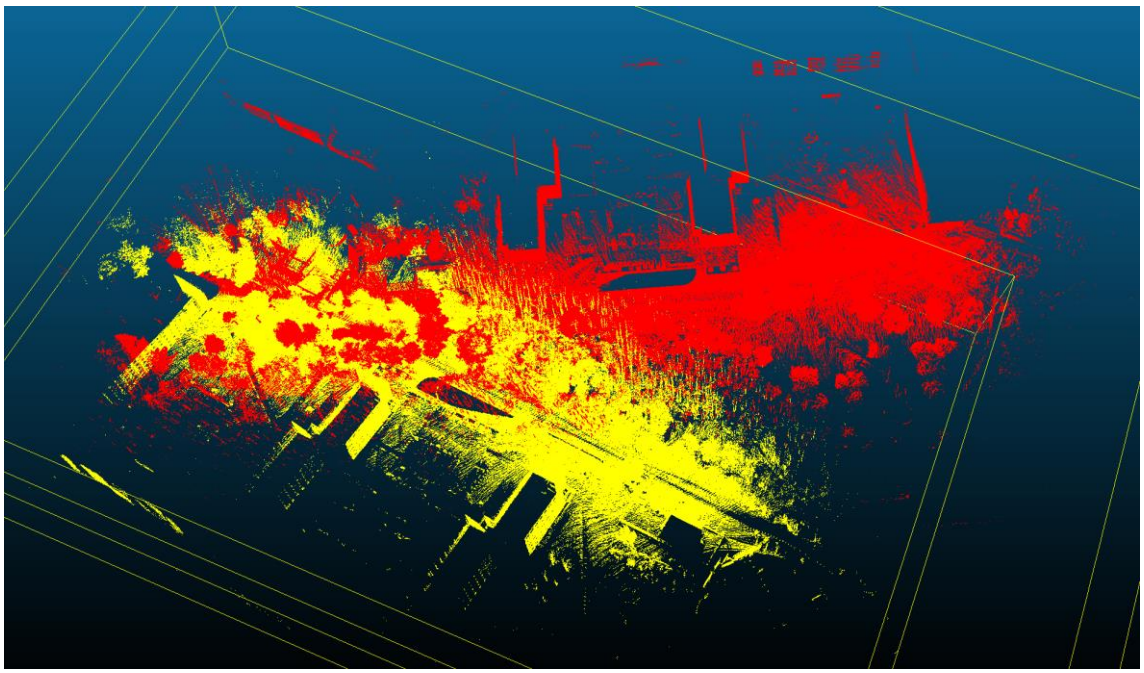
Joonis 6. ICP algoritmiga registreeritud Joonis 5 kujutatud 3D kaardid.

ICP automaatne registreerimise algoritm annab proovitud meetoditest täpseima tulemuse ainult juhul, kui kaardid on suures osas kattuvate objektidega ja nad on eelnevalt joondatud st samasse suunda pööratud. ICP algoritm ei toimi, kui kaardid on ainult osaliselt kattuvad või nad ei joondu. Näiteks Joonis 7 on kujutatud üks ebaõnnestunud ICP registreerimine, kus mõlemad kaardid on täpselt samast alast, aga nad on üksteise suhtes pööratud.



Joonis 7. Ebaõnnestunud ICP registreerimine

Joonis 8 on samuti kaks 3D kaarti täpselt samades objektidest, aga nad ei joondu. Kollase kaardi alguspunkt on Akadeemia tee 5 ja lõpp-punkt on IT Kolledži juures. Punase kaardi alguspunkt on IT Kolledži ja lõpp-punkt Akadeemia tee 5 juures. Antud juhul ICP registreerimise algoritm kaarte kokku sobitada ei suuda. Sel juhul ei saa meie praeguse parima teadmise juures kasutada automatiseerituid meetodeid ja 3D kaardid tuleb liita käsitsi.



Joonis 8. Samast alast tehtud 3D kaardid TTÜ kampuses Tipi teel.

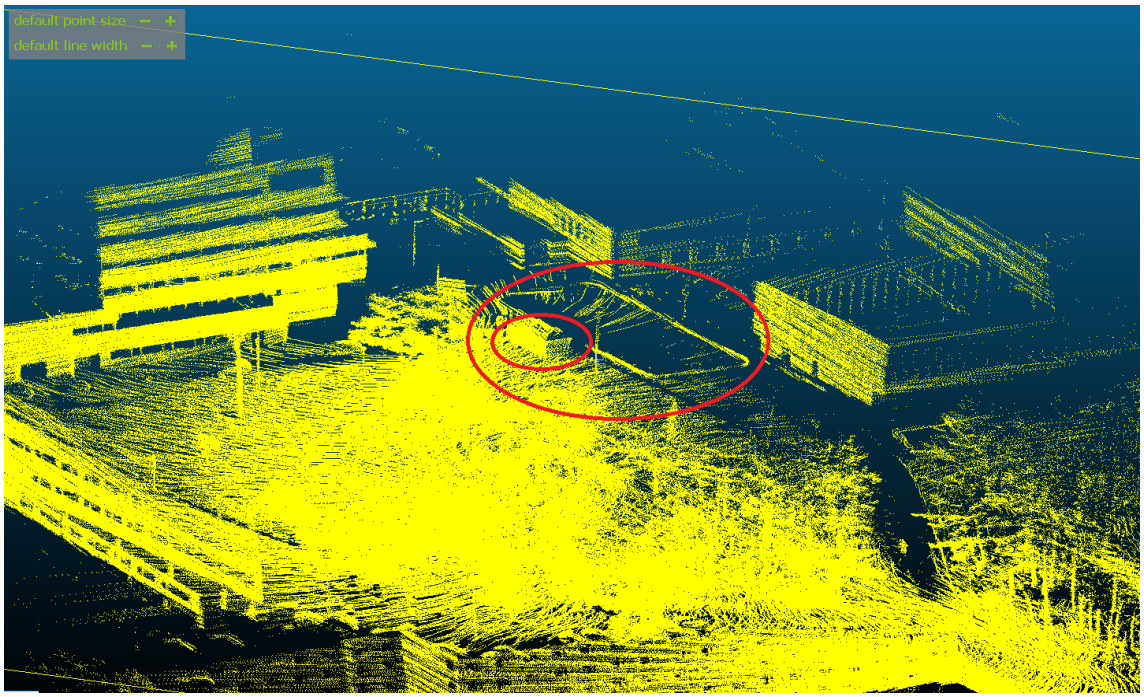
3D kaartide käsitsi liitmiseks tuleb kõigepealt kaardid registreerida programmi Cloud Compare funktsiooniga Align. Selleks tuleb võtta kaks 3D kaarti, määrata kumba liigutatakse ja kumb jääb paigale. Kasutajal tuleb käsitsi määrata vähemalt 4 paari samaväärseid punkte mõlemal 3D kaardil. Kui soovida täpsemat tulemust, võib punkte valida rohkem. Kui kaardid on registreeritud, siis kaartide liitmiseks tuleb kasutada Cloud Compare funktsiooni Merge. Merge liidab kaks või enam 3D kaarti omavahel üheks 3D kaardiks.

3.3.1 Info uuendamine 3D kaardil

3D kaartide järeltöötluses on oluline roll ka kaartidelt vananenud infoga alade välja lõikamine ja uute osade lisamine. Kui näiteks on tekkinud tee äärde uus maja või hoopis mõni aed eemaldatud, siis tuleb vastava infoga ka 3D kaart kaasajastada.

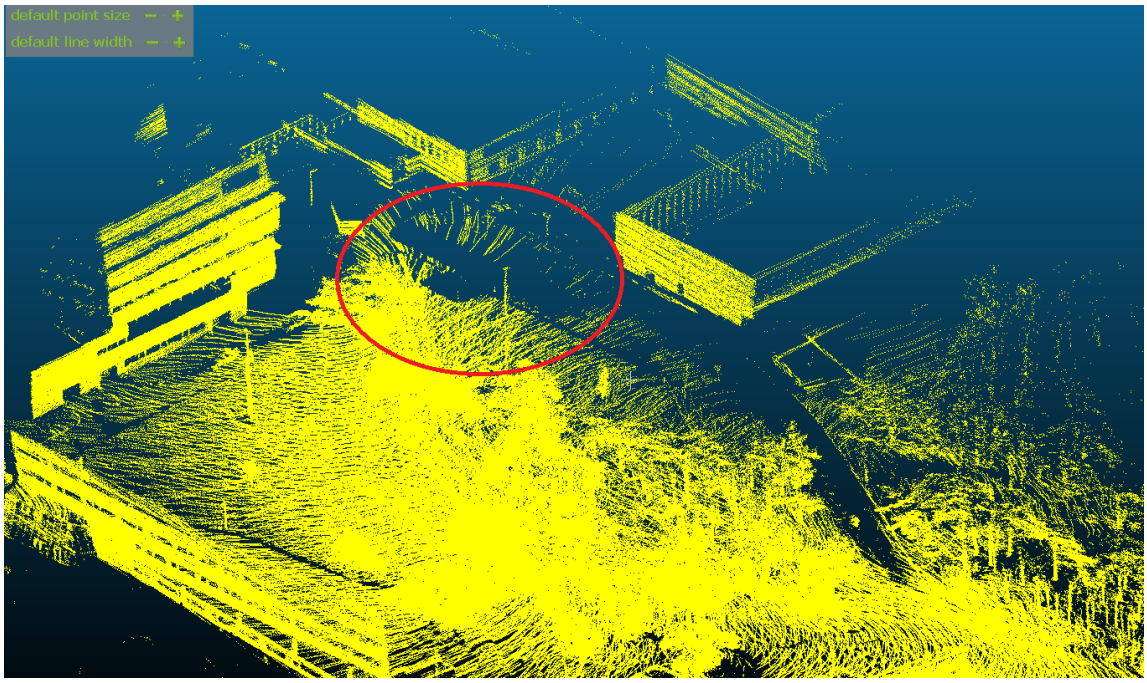
3D kaardilt soovitud ala välja lõikamiseks tuleb kasutada Cloud Compare tööriista Segment, mis võimaldab ala segmenteerida nii hulktahukana kui ristküllikuna. Pärast segmenteerimist saab kasutaja otsustada, kas vastav ala soovitakse välja lõigata või hoopis alles jätta ja ülejäänud kaart kustutada.

TTÜ kampusest tehtud 3D kaardile on jäänud ajutine saalihokiväljak koos piirdega ja kaubik (Joonis 9). Joonis 10 on näha, et väljak ning kaubik on 3D kaardilt eemaldatud. Tänu sellele, et ala on võimalik segmenteerida hulktahukana, sai väljalõike teha väga täpsena, nii et isegi väljaku kõrval olev valgustuspost ja väljaku sisene ala jäid kaardile alles. Eemaldatud ala asemele otsustasin teise 3D kaardi tükki mitte panna, kuna samast alast puhast kaarti ei ole, tavaliselt pargivad seal autod ja nii on ka teistel salvestustel samast alast autod peal. Kuna autod pargivad alati natukene erinevalt, siis autode olemasolu kahandab lokaliseerimise täpsust ja tekitavad kaardil vigu. Seega on eelistatud variant, et vananenud info kaardilt üldse puudub, kui tekitab lokaliseerimisel vigu.



Joonis 9. TTÜ kampus, kus punase ringi sees on ajutine saalihokiväljak ja kaubik.

Algselt oli kavas tuntud objektide, näiteks autode, eemaldamiseks kaardilt kasutada objektituvastust ja objektide projektsioone punktipilves, kuid autorist sõltumatutel asjaoludel ei olnud võimalik käesoleva töö valmimise ajaks koguda piisaval hulgal algandmeid koos kalibreeritud kaamerainfoga. Lisaks illustreerib ülaltoodud näide asjaolu, et kaardile võib sattuda väga erineva kujuga objekte. Seega jääb objektide tuvastamine ja tuvastamise tulemustest lähtuv kaardi modifitseerimine edasiseks tööks.



Joonis 10. Saalihokiväljak ja kaubik on 3D kaardilt välja lõigatud.

3.3.2 3D kaardi osadeks lõikamine

Suuri 3D kaartide faile on mõistlik osadeks lõigata. Sellega hoiame kokku arvutusvõimsust ja säästame mälu, kuid ei kaota täpsuses, kuna kasutusel olevate lidarite maksimaalne mõõtekaugus on 200 m.

3D kaartide osadeks lõikamiseks tuleks kasutada Cloud Compare tööriista Segment, mis vaikimisi avaneb hulknurga redigeerimisrežiimis. See tuleks ümber vahetada ristküliku redigeerimisrežiimi vastu, valides menüüst „Rectangle edition“. Kasutajal tuleb valida osa, mis soovitakse kaardist välja lõigata, seejärel teha hiirega parem-klikk ja siis valida menüüst „Segment Out“. Pärast seda, kui tegevus on kinnitatud, on vasakul pool menüüs näha kaardid alanimedega segmented ja remaining. Kui on soovi, võib kaarditükke veel omakorda lõikuda. Lõpetuseks tuleb kõik osad salvestada.

3.4 Geograafiliste koordinaatide transformatsioon

Perspektiivis soovime, et Iseauto kasutaks navigeerimiseks ja lokaliseerimiseks Eesti ristkoordinaatidega 3D kaarti. Perspektiivis seetõttu, et töö kirjutamise hetkeks ei ole globaalset navigatsiooni satelliitide süsteemi (GNSS) Iseautol veel korralikult tööle saadud. Tulevikus saaks GNSS anda auto täpse asukoha 3D kaardil, mis on oluline

lokaliseerimise alustamiseks, ning seda saab kasutada ka asukoha korrigeerimiseks autonoomse sõidu ajal.

Selleks, et 3D kaart üle viia Eesti ristkoordinaatidele, kasutan PCL pcl_transform meetodit. Eelistan seda Cloud Compare transformatsiooni meetodile, kuna PCL on kiirem. Iga 3D kaart vajab oma transformatsiooni maatriksit, kuna iga 3D kaart omab erinevat alguspunkti ja suunavektorit. Alguspunktina kasutan GNSS-i esimest sõnumit. GNSS-i informatsiooni edastamiseks kasutatakse NMEA sõnumeid. Meie kasutame täpsemalt NMEA sõnumitüüpi GNGGA, mis on järgmisel kujul:

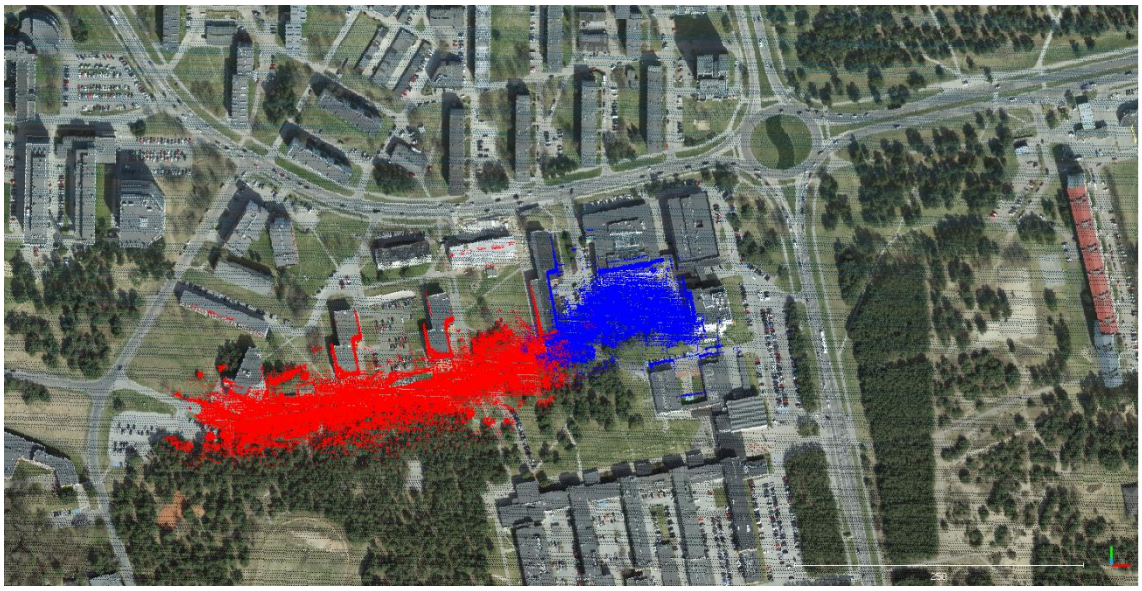
$GNGGA,151546.50,5923.74821440,N,02440.13282805,E,1,12,1.3,19.309,M,18.398,M,,*4D

GNGGA sõnumi teisendamiseks Eesti ristkoordinaadiks kasutan kaastudengi Henrik Linnamäe loodud konverterit (vt Lisa 3), mis on loodud Maa-ameti lähtekoodi [24] abil. Eelneva näitena kasutatud NMEA sõnumi konvertimine Eesti ristkoordinaadistikku andis järgmise tulemuse:

6584396.068712881, 538001.8551627101, 19.309

Seda tulemust kasutangi sisendina pcl_transform meetodis. Tulevikus peaksime ka suunavektori saama NMEA sõnumist, aga kuna see hetkel ei toimi, siis kasutan 3D kaardi paika keeramiseks Cloud Compare keeramise funktsionaalsust.

Tulemuse visualiseerimiseks ja hindamiseks kasutan Cloud Compare rakendust, kuhu on laetud nii Eesti ristkoordinaatidega 3D kaart kui Maa-ameti ortofotod (Joonis 11). Ortofoto on 2D kaart, mis koosneb pikslitest, millega on seotud geograafilised koordinaadid.



Joonis 11. Eesti ristkoordinaatidega 3D kaart ja Maa-ameti ortofotod

Äramärkimist väärib siiski asjaolu, et globaalsete koordinaatidega 3D kaartide töötlemine nõuab arvutitelt märkimisväärselt suuremat arvutusvõimsust. Näiteks Cloud Compare tungivalt soovitab mitte töödelda 3D kaarte, mis on globaalsete koordinaatidega [23]. Põhjuseks on globaalsete koordinaatide suured väärtused (tavaliselt suuremad kui 10 astmes 5) ja kuna nii Cloud Compare kui PCL kasutavad andmetüübina 32 bitist float-i, siis suuremate väärtustega 3D kaartide täpsus kahaneb 1-lt kuni 10 cm-ni.

Eesti ristkoordinaatide lähtepunktid:
x = +6375 000m
y = +500 000m [25]

Seega vajab antud teema veel täiendavat uurimist, et olla kindel, et Eesti koordinaatsüsteemis 3D kaardid toovad suuremat kasu kui täpsuse võimalik kahanemine.

4 Tulemuste hindamine

4.1 Lokaliseerimine

Lokaliseerimisel sobitatakse rosbagist saadud lidarite andmeid vastu etteantud 3D kaarti. Lokaliseerimiseks kasutame Iseautol nii paremat kui vasakut lidarit.

Kaartide kvaliteedi hindamiseks kasutan Autoware lokaliseerimise algoritmi NDT matchingu skoori. NDT matching kasutab kahte ROS-i sõlme – voxel_grid_filter ja ndt_matching. Voxel_grid_filter sõlm vähendab 3D kaardil punktide arvu. Kuna töötlemata 3D kaartidel on punktid üksteisele väga lähedal, siis selleks, et hoida kokku arvutusvõimsust, on mõistlik osa punkte eemaldada. Ndt_matching sõlm edastab sõiduki jooksva asukoha ndt_pose kanalisse ja statistika (näiteks skoor, sooritus aeg, sõiduki kiirus ja kiirendus) ndt_stat kanalisse. Seda statistika kanalisse saadetavat skoori väärtust kasutamegi 3D kaardi kvaliteedi hindamiseks.

Ndt_matchingu täpsuse hinnangu arvutamisel moodustatakse kõigepealt vokselite sõrestik, kus vokseliks on etteantud küljepikkusega (vaikimisi 2 m) ruut, kus arvutatakse punktide kauguste keskmine ning dispersioon. Seejärel leitakse lidari asukoha teisendus, mis minimeerib summarset viga lidaritega mõõdetud punktipilvest moodustatud vokselite ja varem ettevalmistatud 3D kaardi vokselite vahel. Ndt_matching skoor ongi keskmine viga vokseli kohta, kus voksel on etteantud mõõtmispiirkonnas [26].

Viga võib tuleneda nii ebakorrektsest positsioneerimisest kui ka kaardi ja reaalse maailma vahelistest erinevustest, näiteks autodest või inimestest kaardil. Kuna viga jagatakse mõõdetud vokselite arvuga, siis ei mõjuta üksikute ajutiste objektide olemasolu ruumis positsioneerimist oluliselt.

Lokaliseerimise õnnestumisel mängib olulist rolli täpse alguspunkti määramine.

4.2 Tulemuste analüüs

Tulemuste hindamiseks kasutasin lokaliseerimisel sama rosbagi erinevate 3D kaartidega. Antud rosbagis on salvestatud sõit, kus auto teeb lõpus ümberpöörde. See võib

lokaliseerimise täpsust oluliselt mõjutada ja ebakvaliteetse 3D kaardi korral ei suuda auto ennast enam lokaliseerida.

Kõikidelt 3D kaartidelt, mida testimisel kasutasin, eemaldasin kõigepealt kõrvalekalded kasutades `pcl_outlier_removal` meetodit.

Hindamaks, kuidas mõjutab 3D kaardi töötlemine lokaliseerimise täpsust, tegin katseid järgmiste 3D kaartidega:

1. Töödeldud 3D kaart, kus on käsitsi liidetud kaks vasaku lidari 3D kaarti, mis on salvestatud üksteisele vastupidises suunas sõites. Liitmiseks kasutasin Cloud Compare funktsioone Align ja Merge;

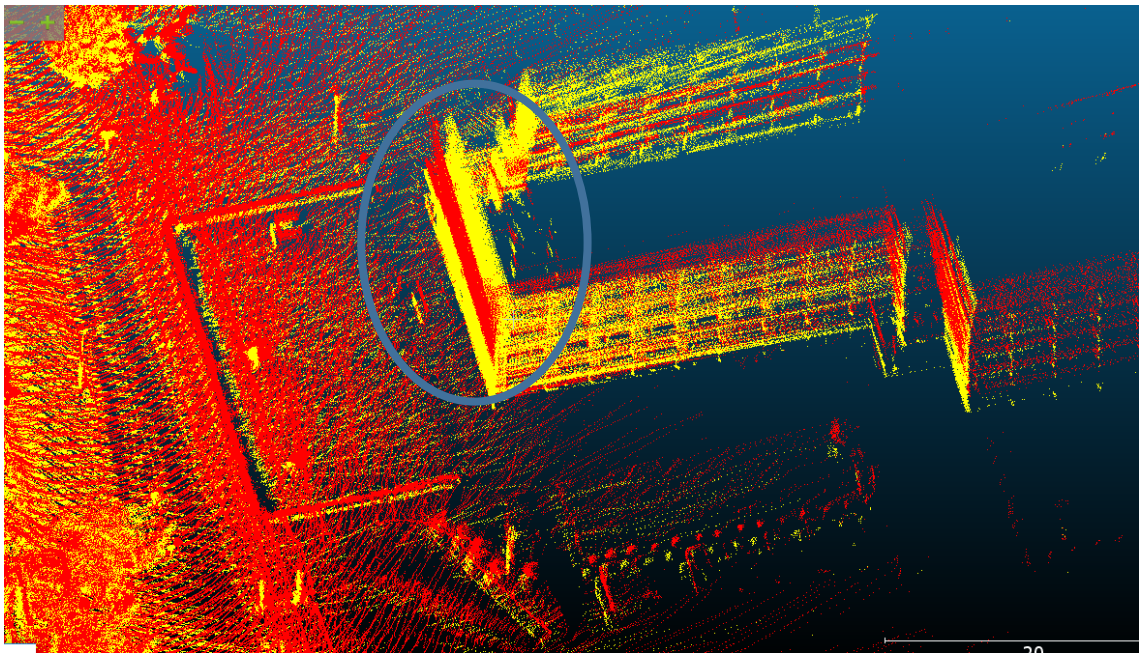
2. Töötlemata vasaku lidari 3D kaart;

3. Töödeldud 3D kaart. Sama, mis kaart nr 1, ainult pärast kaartide liitmist on veelkord kasutatud kõrvalekallete eemaldamise meetodit;

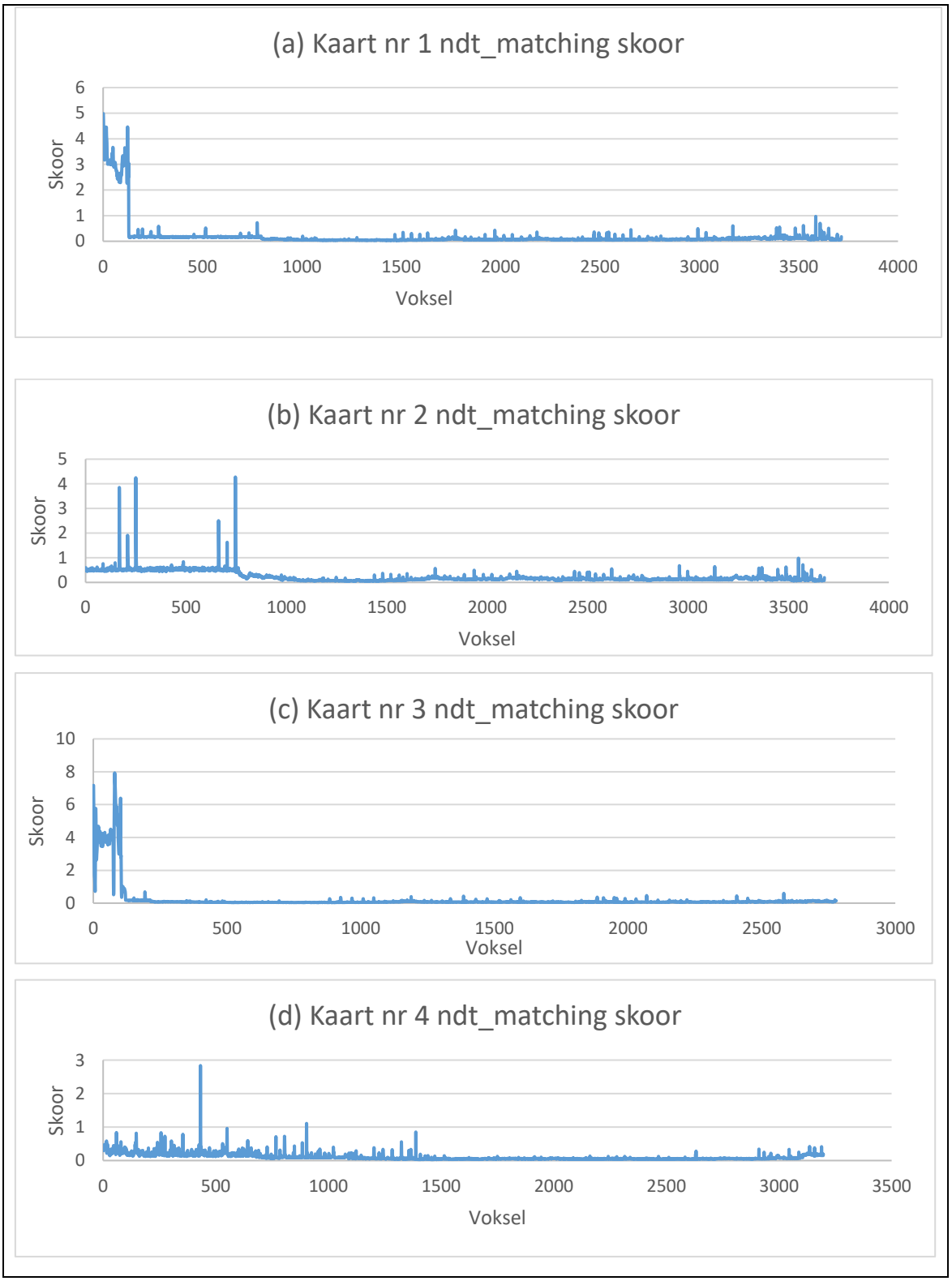
4. Töödeldud 3D kaart, kus oli liidetud parema ja vasaku lidari 3D kaart, need eelnevalt automaatselt registreeritud Cloud Compare ICP algoritmiga.

Kaardi nr 1 loomisel kasutasin kahte vasaku lidari kaarti, kuna parema lidari kaartidel on suurem kalle ning majade tipud on kõverad. Vastupidise suunaga sama lidari kaartide kasutamine annab aga sama efekti, kui kasutada parema ja vasaku lidari kaarte koos – vaateväli on laiem. Samas vastupidise suunaga kaartide liitmisel ei saa kasutada automaatset registreerimist, kuna kaardid ei joondu. Seetõttu tuligi kaart nr 1registreerida käsitsi.

Et hinnata, milline registreerimise meetod annab täpsema tulemuse, kasutasin kaart nr 4 loomisel parema ja vasaku lidari kaarti. Need kaardid registreerisin automaatse ICP algoritmiga. Joonisel 12 on näha ICP registreerimisel tekkiv punane topeltsein. Topeltsein tekib lidarite mõõtmisvigadest, mille päritolule me käesolevas töös jälile ei jõudnud. Lidarid konfigureeriti tootja kaasa antud kalibreerimisfailiga, mis on kõigile lidaritele sama. Antud töö raames lidarite kalibreerimist ei tehtud, aga võimalik on probleemi juurde tagasi pöörduda allikas [27] toodud lähenemist kasutades.



Joonis 12. ICP registreerimisel tekkiv topeltsein



Joonis 13. 3D kaartide ndt_matchingu skoorid läbitud trajektoori lõikes

3D kaartide keskmised ndt_matchingu algoritmi skoori tulemused olid järgmised:

	Keskmine skoor
Kaart 1	0,205134
Kaart 2	0,228336
Kaart 3	0,229156
Kaart 4	0,107665

Tabel 2. Ndt_matchingu keskmised skoorid

Kõikide kaartide lokaliseerimise tulemusega võib rahule jääda ja neid kõiki on võimalik kasutada autonoomseks sõiduks. Kaartidel nr 1 ja 3 on skoorid kõrgemad alguses, mis viitab, et algse asukoha määramine võttis rohkem aega. See peaks paranema, kui GNSS Iseautol tööle saab, siis saab algse asukoha määrata GNSS-i sõnumite abil.

Keskmise skoori järgi andis lokaliseerimisel parima tulemuse kaart nr 4, kus on liidetud parema ja vasaku lidari andmetest tehtud 3D kaardid, kasutades automaatset ICP registreerimise protseduuri. Automaatne registreerimine andis peaaegu kaks korda täpsema lokaliseerimise tulemuse kui käsitsi liidetud 3D kaardid.

Kõikidel kaartidel õnnestus lokaliseerimine hästi ka raja lõpus, kus auto tegi ümberpöörde. Joonis 13 on näha, et ühelgi kaardil raja lõpus skoor oluliselt ei suurenenud. Alampunktis 4.3 esitatud Joonis 18 on näha, millised on skoori tulemused ümberpööramisel, kui lokaliseerimiseks kasutatakse ebakvaliteetseid 3D kaarte.

Tulevikus, kui GNSS-i abil saab määrata auto täpse asukoha kaardil, võiks lokaliseerimiseks kasutada kaarti nr 3, kus töödeldud kaardilt eemaldati veel kord kõrvalekalded. Joonis 13 on näha, et kui algse asukoha määramisega kaasnevat viga mitte arvestada, siis kaardil nr 3 on kõige madalam skoor – peaaegu nulli lähedal. Ja skoor ei tõuse ka lõpus, kus auto tegi ümberpöörde.

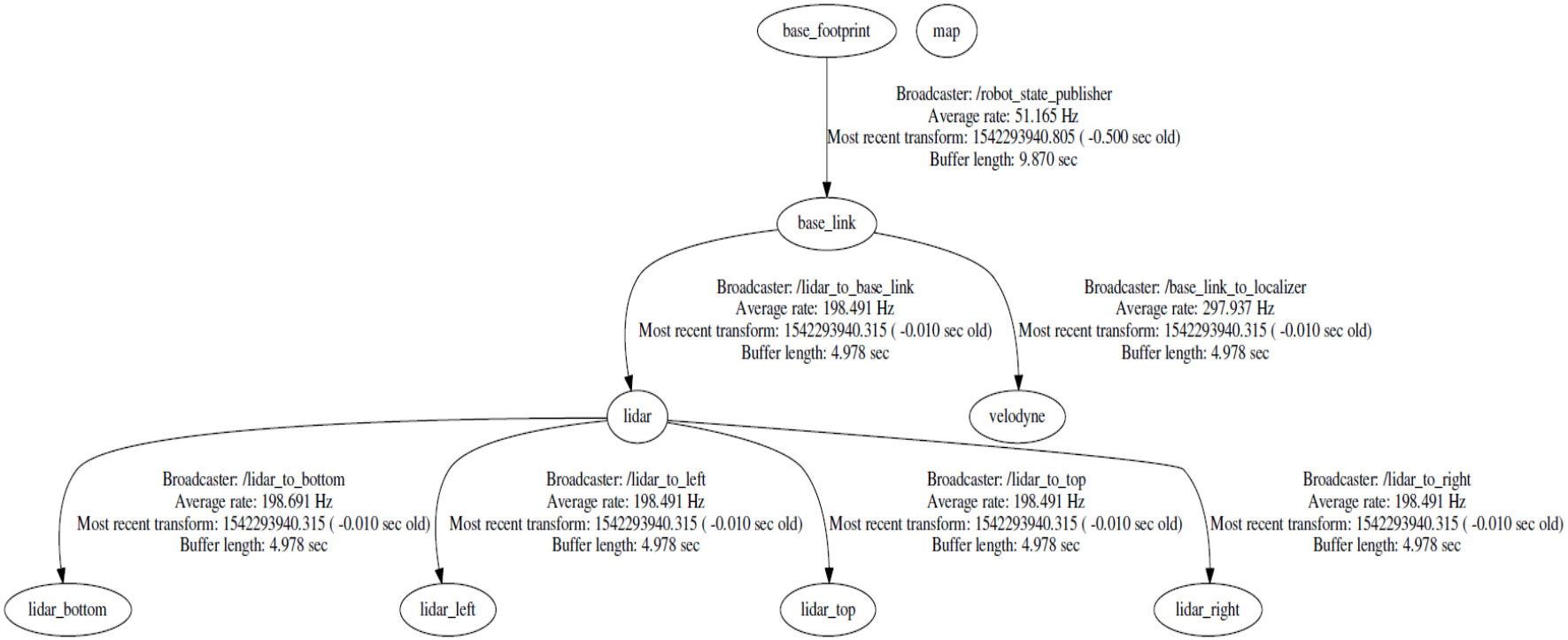
4.3 Vead transformatsiooni puu loomisel

Alapunktis 3.1 on välja toodud, milline peab olema 3D kaartide loomiseks transformatsiooni (TF) puu (Joonis 4). Selle tulemuseni jõudmine ei olnud lihtne ja

käesolevas alapunktis toon välja, mis olid meie komistuskivid ja kuidas nendest üle saime. Käesolevas alapunktis välja pakutud lahendused sobivad kasutamiseks juhul, kui lidarite asendid ei ole teada.

Meie eeldus oli, et TF võtab raamistiku kanali järgi, ja kui oleme määranud samaväärsed väärtused kanalitele `lidar`, `lidar_left`, `lidar_top` ja `velodyne`, siis tekibki õige TF puu. Sellise seadistusega saime Joonis 14 kujutatud puu.

TF puule vastavad käivitusfailid on koodinäites 2 ja 3. Koodinäites 2 on määratud lidarite asukohad teineteise suhtes. Koodinäites 3 on määratud lidari paiknemine baaslüli suhtes.



Joonis 14. vale TF puu ja selle harud

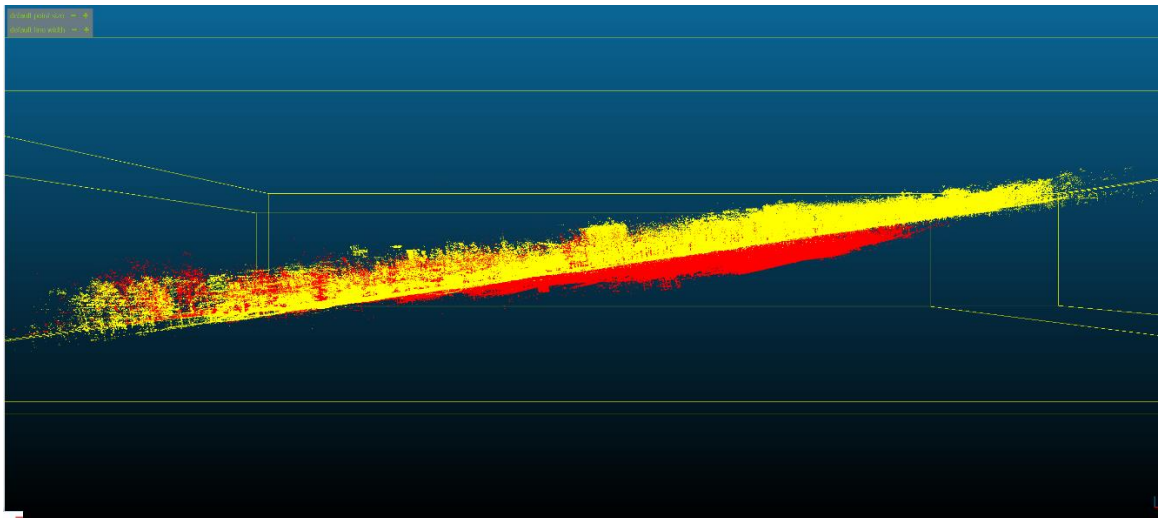
```
<launch>
    <arg        name="velodyne_calibration_file"        default="$(env
ISEAUTO_RESOURCES)/velodyne/Velo_Iseauto.yaml" />
    <node       pkg="tf"             type="static_transform_publisher"
name="lidar_to_top" args="0 0 0 0 0 0 lidar lidar_top 10" />
    <node       pkg="tf"             type="static_transform_publisher"
name="lidar_to_left" args="0 0 0 0 0 0 lidar lidar_left 10" />
    <node       pkg="tf"             type="static_transform_publisher"
name="lidar_to_velodyne" args="0 0 0 0 0 0 lidar velodyne 10" />
    <node       pkg="tf"             type="static_transform_publisher"
name="lidar_to_right"  args="-0.121917   -1.08513 -0.0630073  2.92476
3.10318 -3.00629 /lidar /lidar_right 10" />
    <node       pkg="tf"             type="static_transform_publisher"
name="lidar_to_bottom" args="0.614426 -0.499121  -1.15677  0.0174392 -
0.0110698  0.0387979 lidar lidar_bottom 10" />
</launch>
```

Koodinäide 2. Käivitusfail lidarite omavahelise paiknemise suhetega

```
<launch>
    <arg name="use_tf_static" default="false"/>
    <node   name="robot_state_publisher"   pkg="robot_state_publisher"
type="robot_state_publisher" >
          <param name="use_tf_static" value="$(arg use_tf_static)"/>
    </node>
    <node       pkg="tf"             type="static_transform_publisher"
name="lidar_to_base_link"  args="2.6  0.54  1.40  0.115  0.12  -0.03
/base_link /lidar 10" />
</launch>
```

Koodinäide 3. Baaslüli ja lidari suhte käivitusfail

Selline TF seadistus tekitas 3D kaarte loomisel probleemi – 3D kaardid on tugevalt ülespoole kaldu (Joonis 15). Mida pikem on vahemaa, seda suurem kalle.



Joonis 15. Joonisel on kujutatud kaks 3D kaarti, mis on tehtud üheaegselt. Kollane kaart on tehtud vasaku lidari poolt, punane kaart parema lidari poolt. Et selgelt näha kaartide kaldenurka, on vaade küljelt.

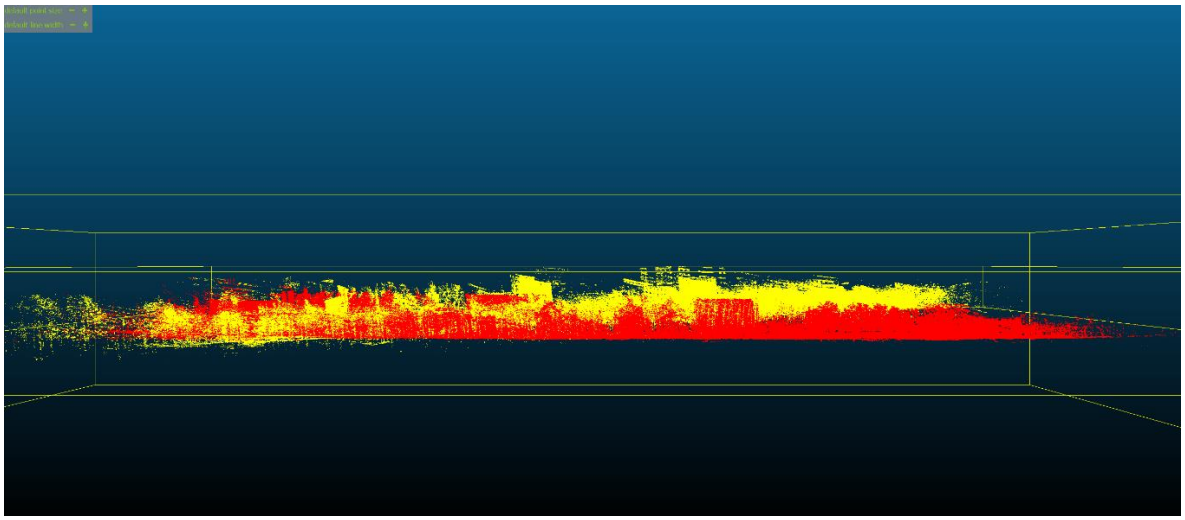
Lahendus Joonis 15 näidatud probleemile on kasutada mõlemal lidaril vastavalt tema kaldenurgale transformatsioonimaatrikseid. Transformatsiooni rakendatakse kõigile valitud üksustele ja nende lastele rekursiivselt. Transformatsiooni maatriks on 4x4 maatriks, mis koosneb 3x3 rotatsioonimaatriksist (ülemise osa 3 esimest veergu) ja 3D vektorist (ülemise osa neljas veerg) ning viimane rida koosneb alati kolmest 0-st ja ühest 1-st. Transformatsiooni eesmärk on eemaldada kaardilt kalle ja tuua z-telje alguspunkt nulli. Lisaks peab vastavaid maatrikseid saama kasutada erinevate salvestuste korral.

Vasaku lidari transformatsioonimaatriks:

1	0	0.113	-1.324
0	1	0	0
-0.113	0	1	0
0	0	0	1

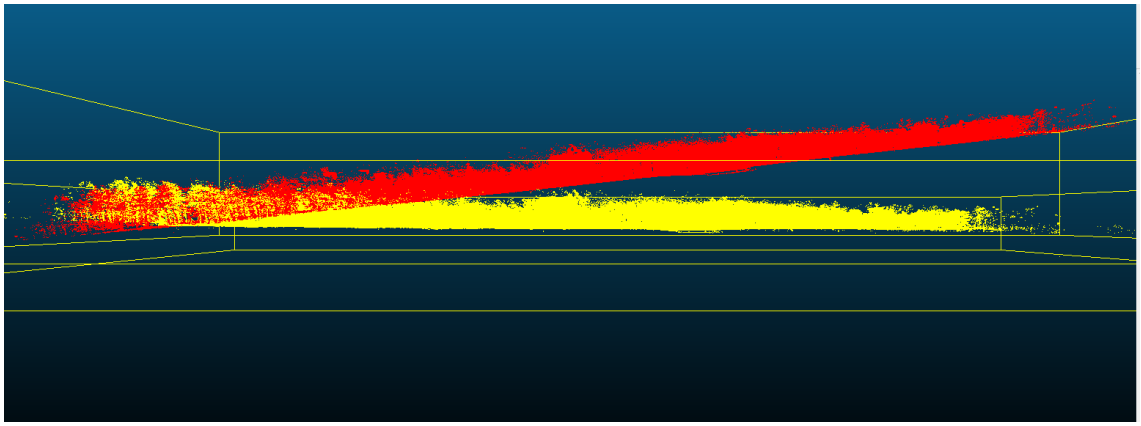
Parema lidari transformatsioonimaatriks:

1	0.014	0.152	-2.235
0	1	-0.091	1.808
-0.152	0.09	1	0
0	0	0	1



Joonis 16. Joonis 15 kujutatud punktipilved peale programmiga Cloud Compare tehtud transformatsiooni.

Võrreldes 3D kaartide kaldenurka korrektse ja vale TF puu korral, siis tulemus on näha Joonis 4. Mõlema kaardi loomiseks kasutasime sama rosbagi. Kollase kaardi loomisel kasutasime käivitusfailina koodinäidet 1. Tulemuseks oli praktiliselt sirge 3D kaart, kus kaardi alguse ja lõpu kõrguste vahe on 2 m. Punase kaardi loomisel kasutasime käivitusfailidena koodinäidet 2 ja 3. Selle 3D kaardi kõrguste vahe oli ligi 35 m.



Joonis 17. Kollane kaart on loodud korrektse TF-ga, punane kaart on loodud vale TF-ga.

Kuna kaardi loomisel kuskilt vigu ega hoiatusi läbi ei jooksnud, siis arvasime oma seadistuse õige olevat. Tulemuste hindamiseks kasutasime Autoware lokaliseerimise algoritmi ndt_matching skoori, mis on kahe sobitatud punkti vaheline keskmine kaugus.

Katseid tegin järgmiste 3D kaartidega:

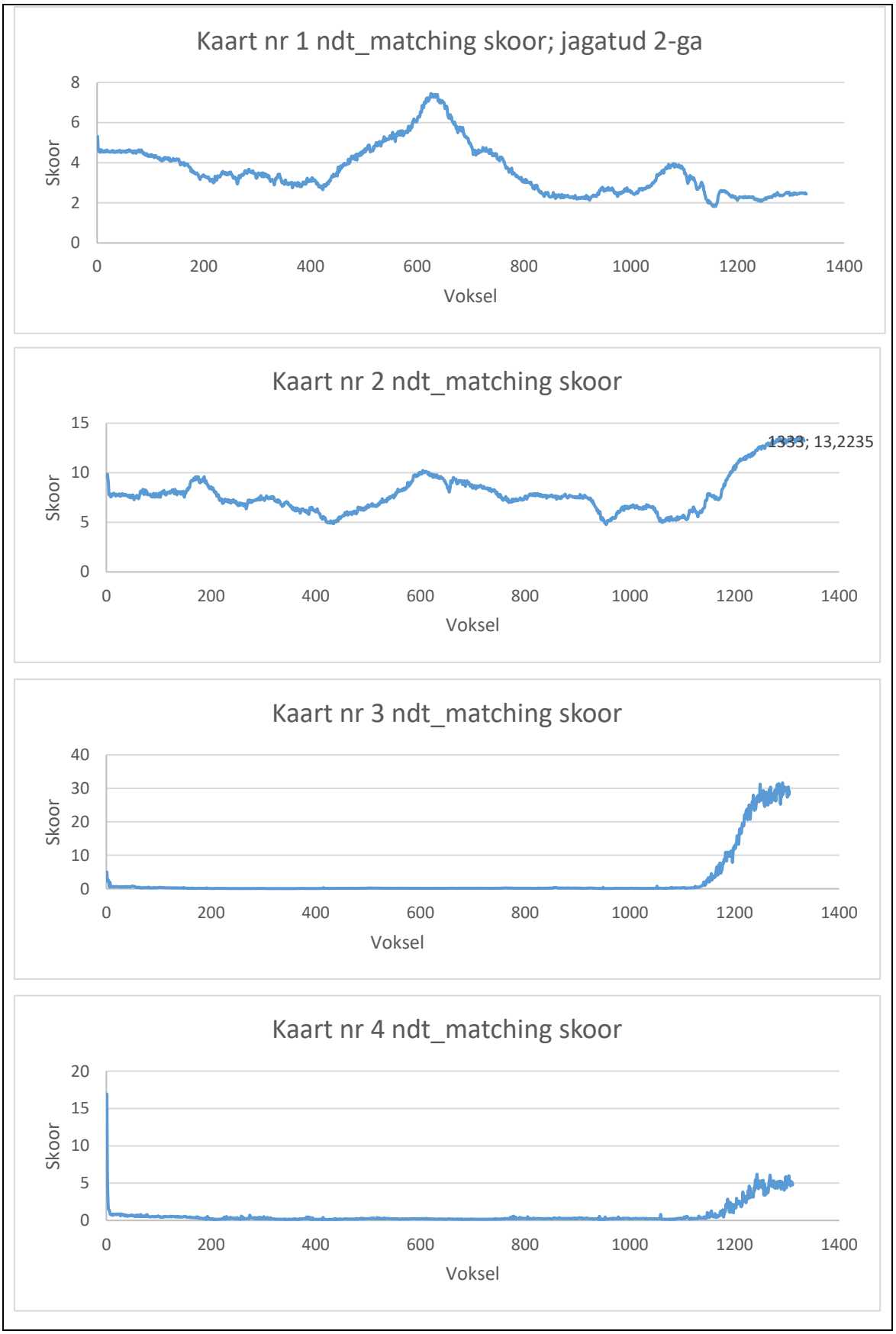
1. Töödeldud 3D kaart, kus oli liidetud parema ja vasaku lidari 3D kaart, need eelnevalt registreeritud Cloud Compare ICP algoritmiga. Lisaks olid mõlemalt 3D kaardilt eemaldatud kalle, kasutades transformatsiooni maatrikseid ning saadud tulemust töödeldud Cloud Compare müra kõrvaldamise filtriga;

2. Töödeldud vasaku lidari 3D kaart, millelt eemaldasin kalde kasutades vasaku lidari transformatsiooni maatriksit;

3. Töötlemata vasaku lidari kaart, millel on sama alguspunkt kui rosbagi salvestusel;

4. Töötlemata vasaku lidari kaart, mille alguspunkt on rosbagi salvestuse lõpp-punkt;

Kuna ndt_matchinguga loodav 3D kaart on kaldu allapoole ja kaardistamisel ndt_mappinguga tekkiv 3D kaart on kaldu ülespoole, siis võib eeldada, et lokaliseerimisel annab parema tulemuse kaartide kombinatsioon, kus kalle on sama-suunaline.



Joonis 18. Vale TF-ga loodud 3D kaartide ndt_matchingu skoorid

3D kaartide keskmised ndt_matchingu skoori tulemused olid järgmised:

	Keskmine skoor
Kaart 1	3,238
Kaart 2	7,885
Kaart 3	2,475
Kaart 4	0,669

Tabel 3. Vale TF-ga loodud 3D kaartide ndt_matchingu keskmised skoorid

Lokaliseerimisel andis parima tulemuse 3D kaart nr 4, mida ei olnud üldse töödeldud ega isegi kallet eemaldatud. Üldiselt ei ole see tulemus üllatav, kuna lihtsustatult öeldes suudab kõver 3D kaart end kõige paremini sobitada kõverasse keskkonda. Ettevaatlikuks teeb skoori tulemus raja lõpus, kus skoor tõuseb 10-20 korda. Seda põhjustab auto tagasipööramine, mille tagajärjel lokaliseerimise täpsus kahaneb märgatavalt. Kõige halvemini mõjub auto tagasipööramine lokaliseerimisel kaart nr 3-ga. Kaart nr 1 on küll kõige stabiilsem, aga keskmine skoor on autonoomseks sõiduks liiga suur.

Üldine hinnang vale TF puuga loodud 3D kaartidega lokaliseerimisele on, et autonoomseks sõiduks neid kaarte kasutada ei saa. Lokaliseerimise täpsus on neil kas liiga madal või on lokaliseerimine ebastabiilne.

Kokkuvõte

Kvaliteetsete 3D kaartide loomise eelduseks on, et me oskame ette anda lidarite täpse asukoha ja kaldenurga. 3D kaartide loomisel kahe kallutatud lidariga on määrava tähtsusega õige transformatsiooni puu olemasolu. Vastasel korral võib küll 3D kaardi loomine õnnestuda, aga kaardid on sellisel juhul kaldus ja kõverad, mis omakorda põhjustab lokaliseerimise ebatäpsust ja autonoomseks sõiduks selliseid 3D kaarte kasutada ei saa.

Käesoleva magistritöö tulemusena on dokumenteeritud vajalikud sammud kahe lidariga 3D kaartide loomiseks ja välja pakutud lahendus 3D kaartide loomise automatiseerimiseks. Selleks on välja töötatud käivitusfail ja kestaskript.

Antud töös tutvustatakse nii automaatseid kui manuaalseid meetodeid 3D kaartide registreerimiseks, kõrvalekallete eemaldamiseks, liitmiseks kui ka osadeks lõikamiseks ja kaartidelt vananenud info eemaldamiseks. Töödeldud 3D kaartide vastavust pärismaailmaga on kontrollitud vastu Maa-ameti ortofotosid, kus visuaalsel vaatlusel oli näha, et majade ning teede kontuurid kattusid. Välja on pakutud ka lahendus 3D kaartide teisendamiseks Eesti ristkoordinaatidesse.

Käesolevas töös defineerisin kaartide kvaliteedi kriteeriumina lokaliseerimise täpsuse. Oodatav tulemus oli, et töödeldud kaardi veahinnangud on väiksemad, kui töötlemata kaardil. Vastav tulemus ka saadi. Testimine viidi läbi nelja erineva töötlusega 3D kaardiga. Lokaliseerimisel andis täpseima tulemuse töödeldud kaart, kus oli liidetud parema ja vasaku lidari andmetest tehtud 3D kaardid kasutades automaatset ICP registreerimise protseduuri. Lokaliseerimise täpsus säilis ka olukorras, kus auto tegi tagasipöörde.

Enne 3D kaartide registreerimist eemaldatakse kaardilt kõrvalekalded PCL `pcl_outlier_removal` meetodiga. Lokaliseerimise täpsust saab parandada veelgi, kui töödeldud 3D kaardilt eemaldada uuesti kõrvalekalded. Teine võimalus lokaliseerimise täpsuse tõstmiseks on kasutada algse asukoha määramiseks GNSS-i sõnumeid.

Lühendite loetelu

2D	Kahemõõtmeline
3D	Kolmemõõtmeline
GNSS	Global Navigation Satellite System
ICP	Closest points algorithm
IMU	Inertial Measurement Unit
LIDAR	Light Detection And Ranging
NDT	Normal Distribution Transform
NDT-OM	Normal Distributions Transform Occupancy Map
NMEA	National Marine Electronics Association
PCD	Point Cloud Data
PCL	Point Cloud Library
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
TTÜ	Tallinn University of Technology
URDF	Unifie Robot Description Format
XML	Extensible Markup Language

Viited

[1] M. G. M. M. F. P. Federico Ferri, „Dynamic obstacles detection and 3D map updating,“ *IROS*, pp. 5694-5699, 2015.

[2] M. Väli, *Evaluation of multiple lidar placement*, Tallinn, 2018.

[3] I. Bogoslavskyi ja C. Stachniss, „Analyzing the quality of matched 3D point clouds of objects,“ %1 *International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, 2017.

[4] Autoware, 2 2018. [Võrgumaterjal]. Available: https://github.com/CPFL/Autoware.

[5] T. S. H. A. a. A. J. L. Jari Saarinen, „Fast 3D Mapping in Highly Dynamic Environments using Normal,“ %1 *IROS*, Tokyo, 2013.

[6] „VLP-16,“ Velodyne, [Võrgumaterjal]. Available: https://velodynelidar.com/vlp-16.html. [Kasutatud 2018].

[7] A. N. C. L. A. J. L. a. J. H. Martin Magnusson, „Evaluation of 3D Registration Reliability and Speed –,“ %1 *IEEE International Conference on Robotics and Automation*, Kobe, Japan, 2009.

[8] M. M. a. A. J. L. Todor Stoyanov, „Comparative Evaluation of the Consistency of Three-Dimensional Spatial Representations used in Autonomous Robot Navigation,“ *Field Robotics*, 2013.

[9] S. T. Y. M. S. M. M. H. Y. K. A. M. T. A. Y. F. T. A. Shinpei Kato, „Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems,“ %1 *ACM/IEEE International Conference on Cyber-Physical Systems*, 2018.

[10] M. Magnusson, „The Three-Dimensional Normal-Distributions Transform – an Efficient,“ 2009.

[11] „ISEAUTO - TTÜ & Silberauto AS,“ [Võrgumaterjal]. Available: http://iseauto.ttu.ee/.

[12] „PCL - Point Cloud Library (PCL),“ 24 11 2018. [Võrgumaterjal]. Available: http://www.pointclouds.org.

[13] 24 11 2018. [Võrgumaterjal]. Available: http://pointclouds.org/documentation/tutorials/pcd_file_format.php.

[14] „CloudCompare - Wikipedia,“ 24 11 2018. [Võrgumaterjal]. Available: http://www.cloudcompare.org/doc/wiki/index.php?title=Main_Page.

[15] P. K. F. C. P. F. a. R. S. Francois Pomerleau, „Long-term 3D map maintenance in dynamic environments,“ %1 *International Conference on Robotics & Automation*, Hong Kong, 2014.

[16] L. P. V. O.-S. E. A. A. C. A. F. S. Filipe Mutza, „Large-scale mapping in complex field scenarios using anautonomous car,“ *ScienceDirect*, p. 439–462, 2016.

[17] „Maptools,“ Tier IV, [Võrgumaterjal]. Available: https://maptools.tier4.jp/. [Kasutatud 2018].

[18] „A Ride In Ford's Self-Driving Car,“ 24 11 2018. [Võrgumaterjal]. Available: https://spectrum.ieee.org/cars-that-think/transportation/self-driving/a-ride-in-fords-selfdriving-car.

[19] F. C. R. S. François Pomerleau, „A Review of Point Cloud Registration Algorithms for Mobile Robotics,“ *Foundations and Trends in Robotics,* 2015.

[20] „rosbag,“ 11 2018. [Võrgumaterjal]. Available: http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data.

[21] „Autoware/ndt_mapping.cpp at 1.7.0 · CPFL/Autoware · GitHub,“ 12 2018. [Võrgumaterjal]. Available: https://github.com/CPFL/Autoware/blob/1.7.0/ros/src/computing/perception/localization/packages/lidar_localizer/nodes/ndt_mapping/ndt_mapping.cpp.

[22] „Autoware_ndt_matching.cpp at 1.7.0 · CPFL_Autoware · GitHub,“ 12 2018. [Võrgumaterjal]. Available: https://github.com/CPFL/Autoware/blob/1.7.0/ros/src/computing/perception/localization/packages/lidar_localizer/nodes/ndt_matching/ndt_matching.cpp.

[23] „Documentation - Point Cloud Library (PCL),“ 12 2018. [Võrgumaterjal]. Available: http://www.pointclouds.org/documentation/tutorials/statistical_outlier.php.

[24] 01 2019. [Võrgumaterjal]. Available: https://www.maaamet.ee/rr/geo-lest/files/geo-lest_function_php.txt.

[25] „L-EST97,“ 12 2018. [Võrgumaterjal]. Available: https://geoportaal.maaamet.ee/est/Andmed-ja-kaardid/Koordinaatsusteemid-ja-kaardilehtede-jaotused/L-EST97-ja-teised-koordinaatsusteemid-p173.html.

[26] „Autoware/NormalDistributionsTransform.cpp at 1.7.0 · CPFL/Autoware · GitHub,“ 01 2019. [Võrgumaterjal]. Available: https://github.com/CPFL/Autoware/blob/1.7.0/ros/src/computing/perception/localization/lib/ndt_cpu/src/NormalDistributionsTransform.cpp#L719.

[27] A. K. A. F. C.L. Glenniea, „CALIBRATION AND STABILITY ANALYSIS OF THE VLP-16 LASER SCANNER,“ %1 *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Lausanne, 2016.

[28] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

[29] H.-M. G. Erik Einhorn, „Generic NDT mapping in dynamic environments and its application for lifelong SLAM,“ *Robotics and Autonomous Systems*, pp. 28-39, July 2015.

[30] „Ford Fusion autonomous research vehicles use LiDAR sensor technology to see in the dark,“ 24 11 2018. [Võrgumaterjal]. Available: https://phys.org/news/2016-04-ford-fusion-autonomous-vehicles-lidar.html.

[31] 12 2018. [Võrgumaterjal]. Available: https://github.com/CPFL/Autoware/tree/master/ros/src/computing/perception/localization/packages/lidar_localizer/nodes/ndt_matching_monitor.

Lisa 1

```
<launch>
    <arg name="velodyne_tf_x" default="2.6" />
    <arg name="velodyne_tf_y" default="0.54" />
    <arg name="velodyne_tf_z" default="2.2" />
    <arg name="velodyne_tf_yaw" default="0.115" />
    <arg name="velodyne_tf_pitch" default="0.12" />
    <arg name="velodyne_tf_roll" default="-0.03" />

 <!--             -->
 <!-- NDT_MAPPING -->
 <!--             -->

<param name="use_sim_time" value="true" />
<param name="localizer" value="velodyne" />

<!-- TF-s and vehicle model -->
          <param name="robot_description"
textfile="/home/iseauto/iseauto-master/vehicle/iseauto.urdf"/>
    <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher" >
          <param name="use_tf_static" value="false"/>
   </node>

    <node pkg="joint_state_publisher" type="joint_state_publisher"
name="joint_state_publisher" >
   </node>

    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_base_link" args="2.6 0.54 2.2 0.115 0.12 -0.03
/base_link /velodyne 10" />
    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_top" args="0 0 0 0 0 0 velodyne lidar_top 10" />
    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_left" args="0 0 0 0 0 0 velodyne lidar_left 10" />
    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_right"  args="-0.121917   -1.08513 -0.0630073
2.92476  3.10318 -3.00629 velodyne lidar_right 10" />
    <node pkg="tf"  type="static_transform_publisher"
name="velodyne_to_bottom" args="0.614426 -0.499121  -1.15677
0.0174392 -0.0110698  0.0387979 velodyne lidar_bottom 10" />

<!-- Velodynes position relative to base_link. -->
    <param name="tf_x" type="double" value="$(arg velodyne_tf_x)" />
    <param name="tf_y" type="double" value="$(arg velodyne_tf_y)" />
    <param name="tf_z" type="double" value="$(arg velodyne_tf_z)" />
    <param name="tf_roll" type="double" value="$(arg
velodyne_tf_roll)" />
    <param name="tf_pitch" type="double" value="$(arg
velodyne_tf_pitch)" />
    <param name="tf_yaw" type="double" value="$(arg velodyne_tf_yaw)"
/>
```

```
<!-- lidar_top frame-->
    <group ns="lidar_top">
        <include file="$(find
points_preprocessor)/launch/points_concat_filter.launch">
          <arg name="output_frame" value="lidar_top" />
          <arg name="input0" value="/lidar_left/points_raw" />
          <arg name="input1" value="/lidar_right/points_raw" />
          <arg name="output" value="points_raw" />
        </include>
    </group>

<!-- rosbag to play -->
   <arg name="file" default="/home/iseauto/rosbags/itkolledz-uhikas-
20181115173412.bag" />
   <node pkg="rosbag" type="play" name="rosbag_play" output="screen"
args="--clock -d 5 $(arg file) ">
          <remap from="/lidar_left/points_raw" to="/points_raw"/>
          <remap from="/tf" to="/rosbag_tf"/>
   </node>

<!-- ndt_mapping -->
   <node pkg="lidar_localizer" type="queue_counter"
name="queue_counter" output="screen"/>
   <node pkg="lidar_localizer" type="ndt_mapping" name="ndt_mapping"
output="screen">
         <param name="method_type" value="2" />  <!-- pcl_generic=0,
pcl_anh_gpu=2 -->
         <param name="use_imu" value="false" />
         <param name="use_odom" value="false" />
         <param name="imu_upside_down" value="false" />
         <param name="imu_topic" value="/imu_raw" />
         <param name="incremental_voxel_update" value="false" />
   </node>

</launch>
```

Lisa 2

```
#!/bin/bash

FILENAME=/tmp/ndt_mapping_`date +%Y%m%d%H%M%S`
PCD=`pwd`/ndt_map`date +%Y%m%d%H%M%S`.pcd

roslaunch $ISEAUTO_HOME/launches_imiev/autoware/ndt_mapping.launch >
$FILENAME &

LAUNCHPROC=$!

LEN=0
while true ; do
        sleep 10
        NEWLEN=`wc -l $FILENAME`
        if [[( "$LEN" == "$NEWLEN" ) ]] ; then
        echo "Saving to file: $PCD"
rostopic pub /config/ndt_mapping_output
autoware_msgs/ConfigNdtMappingOutput "header:
  seq: 1
  stamp:
    secs: 0
    nsecs:         0
  frame_id: ''
filename: "$PCD"
filter_res: 0.2" &
          ROSMSGPID=$!

          sleep 10
          kill $LAUNCHPROC
          kill $ROSMSGPID
          #rm $FILENAME
          exit 0
       else
         LEN=$NEWLEN
       fi
done
```

Lisa 3

```
# pohineb Maa-ameti lahtekoodil https://www.maaamet.ee/rr/geo-
lest/files/geo-lest_function_php.txt

import csv
from math import floor, sin, pi, sqrt, cos, log
def main():
    nmea_array = []
    with open('nmea.csv', newline='') as csvfile:
        reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
        for row in reader:
            for element in row:
                fields = element.split(",")
                print("raw " + fields[2] + "," + str(float(fields[4]))
+ "," + fields[9])
                print(calculate(float(fields[2]), float(fields[4]),
float(fields[9])))

def calculate(latitude, londitude, h):
    # def lat,lad,lod,lon
    # def latd, lond

    lad = floor(latitude / 100.0)
    lat = latitude - lad * 100.0
    lod = floor(londitude / 100.0)
    lon = londitude - lod * 100.0

    latd = lad + ((latitude - lad * 100.0) / 60)
    lond = lod + ((londitude - lod * 100.0) / 60)

    m_geo_lat = latd
    m_geo_lon = lond
    LAT = (latd * (pi / 180))
    LON = (lond * (pi / 180))
    a = 6378137.000000000000
    F = 298.257222100883
    RF = F
    F = (1 / F)
    B0 = ((57.000000000000 + 31.000000000000 / 60.000000000000 +
3.194148000000 / 3600.000000000000) * (pi / 180))
    L0 = ((24.000000000000) * (pi / 180))
    FN = 6375000.000000000000
    FE = 500000.000000000000
    B1 = ((59.000000000000 + 20.000000000000 / 60.000000000000) * (pi
/ 180))
    B2 = ((58.000000000000) * (pi / 180))
    xx = (latd - FN)
    yy = (lond - FE)
    f1 = (1 / RF)
    er = ((2.000000000000 * f1) - (f1 * f1))
    e = sqrt(er)
```

```
    t1 = sqrt(((1.000000000000 - sin(B1)) / (1.000000000000 +
sin(B1))) * (
        pow(((1.000000000000 + e * sin(B1)) / (1.000000000000 - e *
sin(B1))), e)))
    t2 = sqrt(((1.000000000000 - sin(B2)) / (1.000000000000 +
sin(B2))) * (
        pow(((1.000000000000 + e * sin(B2)) / (1.000000000000 - e *
sin(B2))), e)))
    t0 = sqrt(((1.000000000000 - sin(B0)) / (1.000000000000 +
sin(B0))) * (
        pow(((1.000000000000 + e * sin(B0)) / (1.000000000000 - e *
sin(B0))), e)))
    t = sqrt(((1.000000000000 - sin(LAT)) / (1.000000000000 +
sin(LAT))) * (
        pow(((1.000000000000 + e * sin(LAT)) / (1.000000000000 - e *
sin(LAT))), e)))
    m1 = (cos(B1) / (pow((1.000000000000 - er * sin(B1) * sin(B1)),
0.500000000000)))
    m2 = (cos(B2) / (pow((1.000000000000 - er * sin(B2) * sin(B2)),
0.500000000000)))
    n = ((log(m1) - log(m2)) / (log(t1) - log(t2)))
    FF = (m1 / (n * pow(t1, n)))
    p0 = (a * FF * (pow(t0, n)))
    FII = (n * (LON - L0))
    p = (a * FF * pow(t, n))
    m_x = p0 - (p * cos(FII)) + FN
    m_y = p * sin(FII) + FE
    m_z = h
    return [m_x, m_y, m_z]
main()
```