Mobiele 3D-mapping en lokalisatie met behulp van actieve dieptesensoren

Mobile 3D Mapping and Localization Using Active Depth Sensors

Michiel Vlaminck

Promotoren: prof. dr. ir. W. Philips, prof. dr. ir. H. Luong
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

Vakgroep Telecommunicatie en Informatieverwerking
Voorzitter: prof. dr. ir. J. Walraevens
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2019 - 2020

UNIVERSITEIT
GENT

ISBN 978-94-6355-361-2
NUR 950
Wettelijk depot: D/2020/10.500/38

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Telecommunicatie en informatieverwerking

Image Processing and Interpretation
imec research group at Ghent University

Promotoren: prof. dr. ir. Wilfried Philips
prof. dr. Ir. Hiep Luong

Affiliations

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Telecommunicatie en Informatieverwerking
Onderzoeksgroep Image Processing and Interpretation (IPI), imec
Sint-Pietersnieuwstraat 41, B-9000 Gent, België

Tel.: +32-9-264.34.12
Fax.: +32-9-264.42.95

Dankwoord

Dit werk zou niet mogelijk zijn geweest zonder de steun en de hulp van vrienden, familie en collega's. Ik zou dan ook van de gelegenheid gebruik willen maken om een aantal personen in het bijzonder te bedanken.

In de eerste plaats wil ik mij richten tot mijn promotoren prof. Wilfried Philips en prof. Hiep Luong die mij de kans hebben gegeven om dit interessante onderzoek te verrichten in het domein van de computervisie. Bedankt voor jullie talrijke opmerkingen en inzichten en voor het nauwgezet nalezen van dit boek. Verder wil ik ook mijn juryleden dr. Filip Rooms, prof. Peter Lambert, prof. Peter Schelkens, prof. Peter Veelaert en prof. Philippe De Maeyer bedanken voor hun nuttige feedback.

Verder wil ik graag mijn ouders bedanken omdat ze het maar niet beu werden om mij telkens opnieuw te vragen wanneer ik dit doctoraat eindelijk eens zou afwerken. Ik bedank hen ook voor alle kansen die ze mij hebben gegegeven en voor hun onvoorwaardelijke steun gedurende mijn studies en doctoraatsopleiding.

De vrienden uit de 'Gent-groep' wil ik bedanken voor alle memorabele momenten samen: de reisjes naar toeristische en minder toeristische streken, de legendarische feestjes, de succesvolle sportieve evenementen en niet te vergeten de wekelijkse café-avonden. Hoewel deze laatste recentelijk helaas iets minder frequent georganiseerd worden, zijn ze toch van groot belang geweest om mijn gedachten voldoende te verzetten tijdens mijn doctoraat.

Mijn ex-huisgenoot Pieter en zijn verloofde Charlotte wil ik bedanken voor alle leuke momenten samen op het appartement, niet in het minst de immer gezellige aperitiefjes. Ook mijn ex-huisgenoten Matthias, Dominique, Steven en Mick bedank ik voor de uitstekende sfeer op het appartement tijdens de eerste jaren van mijn doctoraat. Ik wil ook Sarah bedanken voor de steun tijdens drukke periodes en voor de leuke ontspannende momenten na belangrijke deadlines.

Mijn ex-bureaugenoten Jan, Jonas, Simon, Dirk en Joris wil ik bedanken voor de vele geanimeerde discussies en de polls over al dan niet succesvolle Mars-landingen of presidentsverkiezingen in Amerika. Mijn huidige bureaugenoten Dimitri, Gianni en Ljiljana bedank ik voor de vele tips die ik van hen heb gekregen, zowel op persoonlijk, sportief als professioneel vlak. Ik bedank ook collega Bart voor de leuke en leerrijke trips die we samen hebben gemaakt.

Ik ben ontzettend dankbaar dat ik de afgelopen zes jaar in een aangename omgeving heb kunnen werken en in het bijzonder wil ik alle (ex)-collega's bedanken

met wie ik over de middag samen heb gelunched.

Ik wil ook een speciaal woord van dank richten aan Philippe en Davy die altijd paraat stonden om mij te helpen bij praktische problemen. Ik bedank jullie in het bijzonder voor de hulp bij het assembleren van de Liborg robot.

Tenslotte wil ik mij richten tot mijn vriendin Greet. Bedankt voor het geduld en voor de talrijke inspanningen om mij gemotiveerd te houden bij de laatste loodjes van dit doctoraat. Ik ben jou ontzettend dankbaar voor de eindeloze hulp en steun die het werk voor mij aanzienlijk hebben verlicht.

Gent, Maart 2020
Michiel Vlaminck

Table of Contents

List of Figures

List of Tables

List of Acronyms

A

APE	Absolute Pose Error

B

BA	Bundle adjustment
BIM	Building information model
BnB	Branch and Bound

C

CAD	Computer-aided design

D

DoF	Degrees of Freedom

F

FOV	Field Of View

G

GNSS	Global Navigation Satellite System

I

ICP	Iterative Closest Point
IRLS	Iteratively Reweighted Least Squares
IMU	Intertial Measurement Unit

L

LOD	Level of Detail
LCP	Largest Common Point Set

M

MAP	Maximum a posteriori
MLE	Maximum Likelihood Estimation

N

NDT	Normal Distribution Transform

R

RANSAC	RANdom SAmple Consensus
RPE	Relative Pose Error

S

SfM	Structure from Motion
SLAM	Simultaneous Localization And Mapping

T

ToF	Time of Flight
TSDF	Truncated Signed Distance Function

Nederlandse samenvatting

In de toekomst zullen steeds meer robots hun plaats in onze maatschappij opeisen. Ze zullen geprogrammeerd worden om tal van taken over te nemen van mensen, niet alleen voor huishoudelijke doeleinden, maar ook om bewakingsopdrachten uit te voeren, infrastructuur te inspecteren of om schade te beoordelen op plaatsen die moeilijk bereikbaar zijn voor de mens, zoals nauwe tunnels en rioleringen, of grote constructies zoals dammen of bruggen. Ook bij natuurrampen, zoals aardbevingen, zouden ze reddingsoperaties kunnen faciliteren om op zoek te gaan naar overlevenden in gebouwen die zijn ingestort of in brand staan.

Eén van de voornaamste vereisten voor deze robots om volledig operationeel te zijn is het creëeren en onderhouden van een 3D-model van hun omgeving om zichzelf te kunnen navigeren. Het is belangrijk dat dit 3D-mapping-proces snel genoeg is zodat de map meteen beschikbaar is. Soms zal het 3D-model niet alleen gebruikt worden om te navigeren, maar ook om een uitgebreide analyse te doen van de omgeving, bijvoorbeeld voor inspectietaken, wat restricties oplegt aan het detailniveau en de accuraatheid van het 3D-model.

Naast autonome robots zal het concept van *augmented reality* (AR) ook een cruciale rol gaan spelen in onze toekomst. Bij AR-toepassingen wordt de omgeving van de gebruiker 'uitgebreid' of 'verrijkt' met virtuele objecten. Het laat mensen toe om van op afstand met elkaar te interageren, bijvoorbeeld om samen te werken aan gemeenschappelijke opdrachten. Om in staat te zijn om virtuele objecten in de omgeving te plaatsen, moet het AR-apparaat weten hoe de omgeving er uit ziet en om die reden heeft het ook een 3D-model nodig die onmiddelijk beschikbaar is.

Om dergelijk 3D-model van de omgeving op te bouwen, moet het systeem metingen, genomen van op verschillende plaatsen, met elkaar aggregeren, aangezien één enkele scan niet de volledige ruimte in detail kan vastleggen. Het probleem van 3D-mapping is daarom nauw verwant met het probleem van lokalisatie aangezien we alle sensor-metingen doorheen de tijd met elkaar moeten relateren.

Helaas zijn bestaande, 'externe' positioneringssystemen veelal ongeschikt. Het 'global positioning system' (GPS) of andere 'Global Navigation Satellite Systems' (GNSS) zijn bijvoorbeeld onvoldoende nauwkeurig en kunnen niet worden gebruikt binnenshuis of op plaatsen waar het signaal te onbetrouwbaar is, zoals verstedelijkte gebieden waar hoge gebouwen of overhangende structuren het signaal blokkeren. Wielsensoren die de afstand meten die een robot heeft afgelegd door middel van het aantal wielomwentelingen, zijn onbetrouwbaar wanneer de wielen beginnen slippen. Bovendien is het gebruik van dergelijke systemen beperkt tot robots die zijn uitgerust met wielen en kunnen ze niet gebruikt worden in systemen die geïmplementeerd worden in drones of 'unmanned aerial vehicles' (UAV's). 'Inertial measurement units' (IMU's), die ook in staat zijn om positie en orientatie te meten doorheen de tijd, zijn dan weer ofwel te duur ofwel te sterk onderhevig aan drift. Tot slot wordt ook 'ultra-wideband'-technologie (UWB) gebruikt voor 'indoor' positionering, maar door het feit dat eerst 'ankers' geplaatst moeten worden in de ruimte beperkt het gebruik zich veelal tot omgevingen die gemakkelijk te bereiken en controleren zijn.

Al het voorgaande betekent dat er een systeem nodig is dat in staat is om 3D-mapping - en dus ook 3D-positionering - te doen gebaseerd op sensoren die in staat zijn om de omgeving te percipiëren, zoals camera's of dieptesensoren. In de wetenschappelijke literatuur wordt dit probleem ook vaak omschreven als 'simultaneous localization and mapping' of SLAM aangezien het zowel de lokalisatie van de robot alsook het mappen van de omgeving omvat. Huidige 3D-mapping-systemen zijn ofwel te traag ofwel onvoldoende nauwkeurig en robuust. Bovendien zijn veel bestaande systemen nog steeds gebaseerd op externe positioneringssystemen waardoor ze ongeschikt zijn in heel wat omstandigheden. In dit doctoraat hebben we technieken ontwikkeld die toelaten om een 3D-model op te bouwen van de omgeving in ware-tijd en louter gebruik makend van data die afkomstig is van actieve dieptesensoren.

De onderzoekscontributies van dit doctoraat focussen op drie verschillende aspecten in het domein van 3D-mapping en lokalisatie. Ten eerste hebben we technieken ontwikkeld die het 3D-mappingproces dermate versnellen dat het mogelijk wordt om de omgeving te reconstrueren in hoog detail en in ware tijd. Dit betekent dat de data geaggregeerd kan worden in een 3D-model aan de snelheid waarmee ze wordt opgenomen, zodat autonome robots zichzelf in de omgeving kunnen navigeren. Bovendien laat het AR-applicaties toe om de omgeving van de gebruiker te verrijken met virtuele objecten. Dankzij het hoge detail kan het systeem bovendien gebruikt worden voor 'live' inspectietaken of het beoordelen van schade.

Ten tweede hebben we ons geconcentreerd op algoritmen die de nauwkeurigheid van mobiele 3D-mappingsystemen verbeteren tot een niveau dat hoger ligt dan wat beschikbaar is in de academische literatuur. Bovendien hebben we technieken ontwikkeld om het detailniveau dynamisch aan te passen, wat het mogelijk maakt om de 'trade-off' tussen snelheid en nauwkeurigheid aan te passen aan de vereisten van de applicatie of omstandigheden, bijvoorbeeld in het geval dat de 3D-mapping moet gebeuren op een platform met weinig rekenkracht.

Ten derde hebben we ook ingezet op de robuustheid van het 3D-SLAM-proces om situaties te voorkomen waarbij de 'tracking' verloren geraakt door het falen van bepaalde subprocessen. Hieronder geven we verdere verduidelijking bij de voornaamste bijdragen.

Versnellen van 3D-puntenwolkregistratie De basis van onze 3D-mapping oplossing is gebaseerd op de alignatie van 3D-puntenwolken gecapteerd met actieve dieptesensoren, een proces dat ook 'puntenwolkregistratie' wordt genoemd. De meeste bestaande dieptesensoren kunnen gemakkelijk honderdduizenden punten per seconde opnemen en bestaande technieken hebben ontzettend veel tijd nodig om dergelijke puntenwolken te registreren. In dit doctoraat hebben we verschillende methodes ontwikkeld om dit proces te versnellen. Ten eerste hebben we efficiënte datastructuren bedacht, gebaseerd op hierarchische boomstructuren, die de data-associatie tussen scans of tussen een scan en het 3D-model veréénvoudigen. Ten tweede hebben we een nieuw 'multi-resolutie' registratie-algoritme ontwikkeld gebaseerd op diezelfde hierarchische boomstructuren. De resultaten daarvan werden gepresenteerd in [Vlaminck 17c]. Tot slot hebben we de algoritmen op die manier ontworpen dat ze de kracht van moderne GPU's maximaal benut.

Verhogen van de nauwkeurigheid van 3D-puntenwolkregistratie Puntenwolken gecapteerd met dieptesensoren hebben enkele eigenschappen die het moeilijk maken om hen doorheen de tijd met elkaar te aligneren, zoals hun niet-uniforme en dungezaaide puntdensiteit en hun onnauwkeurigheden door foutieve metingen. Standaardtechnieken voor registratie falen typisch voor dit soort data. Om de nauwkeurigheid te verhogen hebben we geavanceerde alignatietechnieken ontwikkeld die beter kunnen omgaan met deze 'artefacten'. We hebben een 'surface'-analysetechniek bedacht die we gebruiken om laag-niveau 'features' af te leiden van de puntenwolken. Deze eigenschappen hebben we vervolgens geïncorporeerd in het registratieproces om zowel opéénvolgende puntenwolken te aligneren alsook een puntenwolk te aligneren met de geaggregeerde puntenwolk die op dat ogenblik wordt opgebouwd. Daarnaast hebben we het mogelijk gemaakt om de geaggregeerde puntenwolk continue te verbeteren door middel van een 'surface'-reconstructietechniek. Deze laatste laat toe om de ruis in de puntenwolk te verminderen alsook om ze te resampelen, zodat de alignatie van volgende puntenwolken nauwkeuriger kan gebeuren. De resultaten van dit onderzoek werden gepubliceerd in [Vlaminck 16a] en [Vlaminck 18c].

Verbeteren van de robuustheid van 3D-puntenwolkregistratie De robuustheid van het SLAM-proces is eveneens onderhevig aan de 'artefacten' vermeld in de vorige paragraaf. Bestaande registratietechnieken kunnen leiden tot dermate slechte pose-estimaties dat de 'tracking' uiteindelijk volledig verloren gaat. Onze voorgestelde oplossing, gebruik makend van laag-niveau 'features' laat ons toe om te focussen op de meest stabiele regios van de puntenwolk, wat leidt tot meer robuste pose-estimaties. In ons werk, gepubliceerd in [Vlaminck 18c], hebben we ook een robuuste cost-functie voorgesteld die twee puntenwolken met sterk afwijkende oriëntaties succesvol kan aligneren. Aangezien we verder ook de geaggregeerde puntenwolk bijhouden, kunnen we nieuw-gecapteerde puntenwolken ook daarmee registreren. Dat laat ons toe om de pose-estimatie verkregen door het aligneren van twee opéénvolgende puntenwolken verder te verfijnen of corrigeren.

Verbeteren van de schaalbaarheid In dit doctoraat hebben we methoden bedacht om het geheugengebruik van SLAM-systemen te reduceren en ze aan te passen aan de specifieke omstandigheden of toepassingen. Om dit te bereiken hebben we een octree-gebaseerde datastructuur ontwikkeld die een 'surface'-reconstructietechniek integreert. Die laatste maakt het mogelijk om de puntenwolk te resamplen in gebieden die overmatig gesampled werden door de dieptesensor. Op die manier kunnen we 'redundante' punten verwijderen wat leidt tot een compressie van het 3D-model. Op zijn beurt zorgt dit ervoor dat het 3D-model gemakkelijk distribueerbaar is naar andere autonome systemen die samenwerken om een gemeenschappelijk model up-to-date te houden. De octree-datastructuur maakt het ook mogelijk om het detailniveau dynamisch aan te passen alsook om veranderingen in de omgeving te herkennen en door te voeren in het 3D-model.

Versnellen van 'loop closure' 'Loop closure' of het 'sluiten van lussen' is het proces waarbij eerst gedetecteerd wordt dat een plaats in het verleden al is bezocht om nadien van deze informatie gebruik te maken om de fouten die gemaakt zijn te corrigeren. Het 'loop closure'-proces aan de hand van 3D scan data is over het algemeen een tijdsrovende taak. In dit doctoraat hebben we technieken ontwikkeld voor het efficiënt detecteren van lussen, gebaseerd op een GPU-geaccellereerde puntenwolkdescriptor. Onze methode is in staat om dergelijke lussen te detecteren in 100 tot 125 milliseconden. Wat betreft het effectief sluiten van de lus en het corrigeren van de fouten, hebben we een heuristiek bedacht die de fout op het einde van de lus terug in de pose-graaf propageert, daarbij rekening houdend met de residuën van de puntenwolkregistratie. Gebaseerd op die residuën, schatten we hoeveel elke pose-estimatie heeft bijgedragen aan de finale som van pose-errors. De resultaten van dit werk werden gepubliceerd in [Vlaminck 18a].

Naast deze onderzoekscontributies heeft dit doctoraat ook bijgedragen in een aantal onderzoeksprojecten met zowel academische als industriële partners in de periode van 2014 tot 2019. In het icon-project GIPA hebben we meegewerkt aan een generisch AR-platform dat een hele reeks applicaties heeft mogelijk gemaakt, waaronder 'remote collaboration', 'remote expert aid' en 'critical infrastructure management'. In het icon-project ARIA waren onze contributies de ontwikkeling van een accuraat '3D-tracking' systeem gebruik makend van een 'time-of-flight'-camera. Bovendien hebben we een techniek ontwikkeld voor het vinden van de 3D pose van een camera ten opzichte van een beschikbaar 3D model.

Tot slot hebben we een prototype ontwikkeld van een autonome robot die we de naam 'Liborg' hebben gegeven. Deze robot is in staat om in ware-tijd een 3D-model op te bouwen van de omgeving en kan onder meer worden ingezet voor het 'live' monitoren van het opgebouwde 3D-model, bijvoorbeeld om inspectietaken uit te voeren of om schade te beoordelen in omgevingen die moeilijk toegankelijk zijn. Het werk rond Liborg werd gepubliceerd en gepresenteerd op een internationale conferentie [Vlaminck 17b].

English summary

In the future, more and more autonomous robots will take their place in our society. They will be programmed to take over a lot of tasks from humans, not only for domestic purposes but also to conduct surveillance tasks in public places or to inspect critical infrastructure or assess damages in environments that are difficult to reach for humans, such as narrow tunnels, sewer systems, caves or big construction sites. Also in the event of disasters, such as earth quakes, they could facilitate rescue operations to search for survivors in buildings that are collapsed or on fire.

One of the main prerequisites for autonomous robots to fully operate is the ability to build and maintain a 3D map of their surroundings in order to navigate themselves within the environment. Therefore, it is important that the mapping process is fast enough in order to have the map readily available. Sometimes, the 3D map is not only used for real-time navigation but also to conduct a comprehensive analysis of the scene, for example in case of inspection tasks. The latter also imposes restrictions on the level of detail and the accuracy of the 3D map.

Apart from autonomous robots, the concept of *augmented reality* (AR) will also play a crucial role in our future lives. In AR applications, the surroundings of the user are *'augmented'* or enriched with virtual objects. It allows people to remotely interact with each other, for example to collaborate on common tasks. However, in order to put virtual objects in the scene, the AR device needs to know how the users' environment looks like and therefore it also needs a 3D map that is readily available.

In order to build a 3D map of the environment, the system needs to aggregate sensor measurements taken from different positions as one single scan will never cover the whole scene. The problem of 3D mapping is thus closely related to the one of localization as we need to relate all the sensor measurements through time. However, existing positioning systems are often inappropriate. For example, Global Positioning System (GPS) or other Global Navigation Satellite Systems (GNSS) are insufficiently precise and do not work indoors or in environments where the signal is too poor. Wheel odometry sensors, which measure the distance travelled by the robot based on the number of revolutions of the wheels, are also unreliable as the wheels can start to slip. In addition, they are limited to robots that are equipped with wheels and are unsuitable for robotic systems implemented in drones or unmanned aerial vehicles (UAV). Inertial measurement units (IMUs) are also able to estimate the position and orientation over time but generally they are either too expensive or too much subject to drift. Finally, ultra-wideband (UWB) technology requires that

anchors have to be put into the scene first, which makes it less scalable and limits its use to controlled environments that are easy to reach.

All this means that a system is needed that is able to perform 3D mapping - and thus also 3D positioning - based on sensors that are able to perceive the environment, such as cameras or depth sensors. In the scientific literature, this problem is also known as *simultaneous localization and mapping* or SLAM as it involves both the precise localization of the robot and the mapping of the environment at the same time. Current 3D mapping solutions are either too slow or they are insufficiently accurate or robust. Moreover, they are often still relying on external positioning, which makes them unsuitable in a lot of circumstances. In this work, we invented techniques that allow to build a 3D map of the environment in real-time and solely using data captured with active depth sensors.

The research contributions of this work focus on three major improvements in the domain of 3D mapping and localization. First, we developed techniques to accelerate the mapping process making it possible to reconstruct the environment in high detail in real-time. This means that the data is aggregated into a 3D map at the rate it comes in, such that it can be used by autonomous robots to navigate themselves within the environment. It also allows AR applications to use the 3D map to enrich the field of view of the user with virtual objects. Thanks to the high level of detail of the reconstructions the system can also be used for real-time inspection purposes or real-time damage assessment.

Second, we focused on algorithms that improve the accuracy of mobile 3D mapping to a level above what is available in the academic literature. In addition, we developed techniques to dynamically adjust the level of detail, making it possible to tune the speed versus accuracy trade-off depending on the use case or the 'circumstances', e.g. in case of resource-constrained operating platforms. Third, we focused on the robustness of the 3D SLAM process in order to avoid situations in which the *tracking* is lost due to failures in one of the different sub-processes. Below we explain the main research contributions of this work.

Acceleration of 3D point cloud registration The core of our 3D mapping solution is based on the alignment of 3D point clouds acquired by depth sensors, a process also referred to as *point cloud registration*. Most of the modern depth modalities are able to capture hundreds of thousands of points per second and standard registration techniques take a tremendous amount of time. In this PhD, we propose several ways of accelerating this process. First, we developed efficient data structures - based on hierarchical trees - to keep track of the 3D map, facilitating data association among multiple scans or between a scan and the 3D map. Second, based on the hierarchical trees, we designed a novel multi-resolution registration algorithm. The results of this part have been presented in [Vlaminck 17c]. Finally, we carefully designed the algorithms in such a way that the power of modern GPUs is maximally exploited.

Increasing the accuracy of 3D point cloud registration Point clouds acquired from depth devices have several properties that make it difficult to match them over time, e.g. a non-uniform and sparse point density, inaccurate measurements, etc. Standard registration techniques typically fail on this kind of data. In order to increase the accuracy, we developed advanced point cloud alignment techniques that are able to cope with this large variety of 'artefacts'. Moreover, we designed a surface analysis technique that is used to derive low-level shape features from the point clouds. These features are then incorporated in the registration process, both to align point clouds acquired from consecutive scans as well as to align a point cloud with the 3D map being built. Furthermore, we made it possible to continuously improve the 3D map under construction by means of a surface reconstruction technique. This allows to reduce the noise, re-sample the aggregated point clouds and conduct the alignment of consecutive point clouds more accurately. The results of this work were published in [Vlaminck 16a] and [Vlaminck 18c].

Improving the robustness of 3D point cloud registration The robustness of the SLAM process is also subject to the point cloud 'artefacts' mentioned in the previous paragraph. Existing point cloud registration techniques can lead to such bad pose estimates that the tracking is eventually lost. Our proposed solution using low-level shape features allows us to focus on the more stable regions of the point cloud, leading to more robust estimates. In [Vlaminck 18c] we also proposed a robust cost-function that can successfully align two point clouds with a large deviation in orientation. Moreover, as we keep track of the 3D map under construction, we relate new point clouds to the entire 3D map, allowing us to 'correct' pose estimates in case the matching of two consecutive point clouds failed entirely. Finally, we developed a branch-and-bound algorithm to sample the transformation space in case the point cloud registration got stuck in a wrong local minimum.

Improving the scalability In this PhD we provided several tools to reduce the memory foot-print of the SLAM system and to make it adaptable to the circumstances and the use case. We propose a novel octree-based map integrating a form of surface reconstruction, which makes it possible to re-sample the point cloud in areas that are overly sampled. Doing so, we lower the redundancy by removing points, leading to a compression of the map. This on its turn allows the map to be easily distributable among other autonomous systems. This eventually makes it possible that multiple robots or agents work together to keep a common map of the environment up-to-date. The octree data structure behind it also facilitates dynamic adjustments of the level of detail as well as detecting and applying changes in the scene into the map.

Accelerating loop closure Loop closure is the process of detecting places that have been visited before and correcting errors that have been made along

the way. Loop closure using 3D scan data is generally a computation-intensive task. Regarding the detection, we developed a GPU-based global point cloud descriptor that is able to detect loops in 3D lidar data in approximately 100 to 125 ms. Regarding loop correction, we developed a heuristic that propagates the determined error at the end of the loop back into the SLAM pose graph, thereby taking into account residuals of the point cloud registration process. Based on these residuals, we predict how much each pose estimate accounts for in the final pose error when a loop has been detected. The results of this work were presented in [Vlaminck 18a].

Besides the above research contributions, this work also contributed in a few research projects with both academic and industrial partners during the period of 2014 until 2019. In the icon project GIPA, we contributed to a generic AR platform able to serve a whole range of applications including remote collaboration, remote expert aid and critical infrastructure management. In the icon project ARIA, our main contributions were the development of an accurate 3D position tracking system using a time of flight (ToF) camera. We also developed a system for finding the 3D pose of a (head-mounted) camera relative to a known BIM model available in a remote database.

Finally, we developed a prototype of an autonomous robot, which we called 'Liborg'. Liborg is able to perform real-time 3D mapping of the scene on its embedded computer, allowing a whole range of applications such as live monitoring of the acquired 3D model, for instance to perform inspection or assess damages in areas that are difficult to reach. The work on Liborg has been published and presented at an international scientific conference [Vlaminck 17b].

1 Introduction

In the world of tomorrow more and more autonomous robots will take over a plethora of tasks from humans. Think about domestic robots cleaning our houses, cooking our meals, doing our dishes or ironing our clothes. Also in public places, more and more robots will emerge in the future. They will clean conference halls, shopping malls, swimming pools or conducting surveillance tasks in train stations or on airports. They could also be deployed to inspect critical infrastructure or assess damages in environments that are difficult to reach for humans such as narrow tunnels, sewer systems, caves or big constructions such as dams or bridges. Also in the event of disasters, such as earth quakes, they could facilitate rescue operations to search for survivors in buildings that are collapsed or on fire.

For all the aforementioned applications, one of the main prerequisites is to give robots the ability to build and maintain a 3D map of their surroundings in order to navigate themselves within the environment. It is thus necessary that the mapping process is fast enough in order to have the map readily available. Sometimes, the 3D map is not only used for real-time navigation but also to conduct a comprehensive analysis of the scene, for example in case of inspection tasks. Therefore, the level of detail and the accuracy of the 3D map is also important. This is especially the case for so called *'digital twins'* of spaces, buildings or entire cities, which are more and more demanded. These digital twins serve as replicas and are used to conduct urban planning or organize events in case of city models or to automate and optimize processes in industrial settings.

Apart from autonomous robots or digital twin production, the concept of *aug-*

mented reality (AR) will also play a crucial role in our future lives. In AR applications, the surroundings of the user are *'augmented'* or enriched with virtual objects. It allows people to remotely interact with each other, for example to collaborate on common tasks. However, in order to put virtual objects in the scene, the AR device needs to know how the users' environment looks like in order to render them in a veracious way. Also for AR applications a system is thus needed to map the surroundings of the user in 3D and also in this case, the system needs to be fast enough.

1.1 Problem statement

In order to build a 3D map of the environment, the system needs to aggregate sensor measurements taken from different positions, as one single scan will never cover the whole scene due to the limited resolution and field of view of the sensor. The problem of 3D mapping is therefore closely related to the one of localization as we need to relate all the sensor measurements through time. However, external positioning systems are insufficiently precise and do not work indoors. The Global Positioning System (GPS) or other Global Navigation Satellite Systems (GNSS) can not be used indoor or in environments where the GPS signal is too unreliable, for example in urban areas where large structures or overhanging objects cause the GPS signal to be lost or highly inaccurate at times. In addition, GPS is not able to give information about the orientation of the sensor.

Another way of estimating the position is by using wheel odometry sensors that measure the distance travelled by the robot based on the number of revolutions of the wheels. However, this positioning system fails when the wheels start to slip. In addition, wheel odometry is self-evidently limited to robots that are equipped with wheels and cannot be used in case the robot has caterpillar tracks. As the future of autonomous systems will also manifest itself in the form of unmanned aerial vehicles (UAV) or even unmanned underwater vehicles (UUV), this way of odometry is not widely applicable anyway.

Nowadays, more and more indoor positioning systems are based on ultra-wideband (UWB) technology. However, before the position of users or robots can be tracked, anchors have to be put into the scene first, which makes it less scalable and limits its use to controlled environments that are easy to reach. Finally, inertial measurement units (IMUs) are also able to estimate the position and orientation over time. The main problem with IMUs is that they are either too expensive or too much subject to drift.

All this means that we need a system that is able to perform 3D mapping - and thus also 3D positioning - based on sensors that are able to perceive the environment, such as cameras or depth sensors. In the scientific literature, this problem is also known as *simultaneous localization and mapping* or SLAM as it involves both the precise localization of the robot and the mapping of the environment at the same time. In fact, the two tasks are interdependent and the problem can be seen as a typical chicken-or-egg problem. It is impossible to localize yourself without a map, but in order to build a map, you need to know - at all times - where you are in order to relate your observations over time.

Until a few years ago, most SLAM systems were limited in the sense that they were only able to provide 2D maps of the environment. The algorithms nor the hardware were capable of building large-scale 3D maps in real time. Also, the sensors able to acquire depth information were not readily available. Obviously, 2D maps are limited, especially in the case where the robot (or drone) is not travelling along a flat surface. With the advent of novel affordable depth sensing modalities with higher resolutions and accuracy, the field of 3D mapping started to boom.

The process of 3D mapping can be subdivided into three major parts. The first one comprises the *'matching'* of consecutive sensor measurements with each other in order to estimate the current position and orientation - together denoted as the *pose* - of the robot. This estimation is based on *dead reckoning*, in which the current pose is calculated using a previously determined pose and the estimated 'pose difference' between the last two sensor readings. The second part deals with the aggregation of all the sensor data into one coherent 3D map. This *scan-to-map* matching can further improve the estimation of the current pose. It also deals with filtering out noise or inaccurate measurements. The third part, finally, is responsible for recognizing places that have been visited before and for using this information to correct errors that have been made along the way. This part is also referred to as *loop closure*, consisting of both loop detection and loop correction.

Sometimes, a *map* of a scene or room is already present, for example in case of industrial sites or public buildings. In those cases, it can still be desirable to relate the position of a person as well as the direction he is looking at to the known 3D model of the scene. This is especially useful in AR-applications where visual cues about the objects in the scene can be overlaid in the field of view of the user.

1.2 Contributions

1.2.1 Research

The research contributions of this work focus on three major improvements in the domain of 3D mapping and localization. First, we developed techniques to accelerate the mapping process making it possible to reconstruct the environment in high detail in real-time. This means that the data is aggregated into a 3D map at the rate it comes in, such that it can be used by autonomous robots to navigate themselves within the environment. It also allows AR applications to use the 3D map to enrich the field of view of the user with virtual objects. Thanks to the high level of detail of the reconstructions the system can also be used for real-time inspection purposes or real-time damage assessment.

Second, we focused on algorithms that improve the accuracy of mobile 3D mapping to a level above what is available in the academic literature. As generating highly accurate and detailed 3D maps requires more processing power than reconstructions with a lower level of detail while they are not always needed, we also developed techniques to dynamically adjust the level of detail. Doing so, we made it possible to tune the speed versus accuracy trade-off depending on the use case or the 'circumstances', e.g. in case of resource-constrained operating platforms. Third, we focused on the robustness of the 3D SLAM process in order to avoid situations in which the *tracking* is lost due to failures in one of the different sub-processes. In the following paragraphs we zoom in on the most important research contributions that formed the basis for the increase of accuracy, robustness and the reduction of resources, both in terms of processing times and memory consumption.

Acceleration of 3D point cloud registration

As will be described later in this dissertation, the core of our 3D mapping solution is based on the alignment of 3D point clouds acquired by depth sensors, a process also referred to as *point cloud registration*. Most of the modern depth modalities are able to capture hundreds of thousands of points per second and standard registration techniques take a tremendous amount of time. In this work, we propose several ways of accelerating this process. First, we developed efficient data structures - based on hierarchical trees - to keep track of the 3D map, facilitating data association among multiple scans or between a scan and the 3D map. Second, we designed a novel point cloud registration algorithm based on those hierarchical trees. More specifically, we propose a multi-resolution scheme, in which the registration is conducted on different levels of detail, starting from a *coarse* to a more *fine-grained* representation of the point cloud. The results of this work have been presented in [Vlaminck 17c]. Finally, we carefully designed the algorithms in such a way that the power of modern GPUs is maximally exploited.

Increasing the accuracy of 3D point cloud registration

In chapter 3, it will become clear that point clouds acquired from depth devices have several properties that make it difficult to match them over time, e.g. a non-uniform and sparse point density, inaccurate measurements, etc. Standard registration techniques typically fail on this kind of data. In order to increase the accuracy, we developed advanced point cloud alignment techniques that are able to cope with this large variety of 'artefacts'. The main idea is to transform the point cloud into a representation that models the underlying surface. We designed a surface analysis technique that is used to derive low-level shape features from the point clouds. These features are then incorporated in the registration process, both to align point clouds acquired from consecutive scans as well as to align a point cloud with the 3D map being built. Furthermore, we made it possible to continuously improve the 3D map under construction by means of a surface reconstruction technique. This allows us to reduce the noise, re-sample the aggregated point clouds and conduct the alignment of consecutive point clouds more accurately. The results of this work were published in [Vlaminck 16a] and [Vlaminck 18c].

Improving the robustness of 3D point cloud registration

The robustness of the SLAM process is also subject to the point cloud 'artefacts' mentioned in the previous paragraph. Existing point cloud registration techniques can lead to such bad pose estimates that the tracking is eventually lost. Our proposed solution using low-level shape features allows us to focus on the more stable regions of the point cloud, leading to more robust estimates. In [Vlaminck 18c] we also proposed a robust cost-function that can successfully align two point clouds with a large deviation in orientation. Moreover, as we keep track of the 3D map under construction, we relate new point clouds to the entire 3D map, allowing us to 'correct' pose estimates in case the matching of two consecutive point clouds failed entirely. Finally, we adopt a branch-and-bound algorithm to sample the transformation space in case the point cloud registration got stuck in a wrong local minimum.

Improving the scalability

In this work we provided several tools to reduce the memory foot-print of the SLAM system and to make it adaptable to the circumstances and the use case. We propose a novel octree-based map with integrated surface reconstruction, which makes it possible to re-sample the point cloud in areas that are overly sampled. Doing so, we lower the redundancy by removing neighbouring points, leading to a compression of the map. This on its turn allows the map to be easily distributable among other autonomous systems. This eventually makes it possible that multiple robots or agents work together to keep a common map of the environment up-to-date. The octree data structure behind it also facilitates dynamic adjustments of the level of detail as well as detecting and applying changes in the scene into the map.

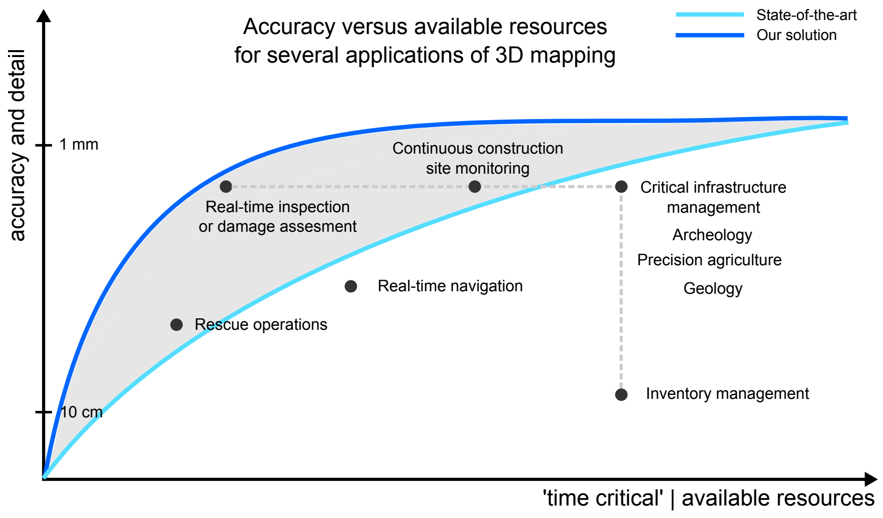


Figure 1.1: Applications of 3D mapping plotted in function of the required accuracy and level of detail as well as to which extent they are time-critical. The gray area indicates which applications are facilitated by our work compared to the state-of-the-art. Our contributions enable applications of autonomous drones or robots that go beyond 'self-navigation' including real-time inspection, damage assessment or rescue operations.

Accelerating loop closure

Recall that loop closure is the process of detecting places that have been visited before and correcting errors that have been made along the way. This loop closure using 3D data is generally a computation-intensive task. Regarding the detection, we developed a GPU-based global point cloud descriptor, that is able to detect loops in 3D lidar data in approximately 100 to 125 ms. Regarding loop correction, we developed a heuristic that propagates the determined error at the end of the loop back into the SLAM pose graph, thereby taking into account residuals of the point cloud registration process. Based on these residuals, we predict how much each pose estimate accounts for in the final pose error when a loop has been detected. The results of this work were presented in [Vlaminck 18a].

1.2.2 Impact on applications

The research contributions from the previous section facilitate a whole range of applications that go beyond 'self-navigation' for autonomous drones or robots. In Figure 1.1, a graph is depicted showing for several applications of 3D mapping its desired level of accuracy and detail as well as to which extent the 3D mapping is time-critical, thereby taking into account how much resources are available on the ‘corresponding’ processing platform, e.g. embedded computer (less resources) versus a powerful computer in the cloud (more resources). The available resources correspond to the total amount of resources on the processing platform minus the ones needed for other subtasks of the application.

Applications running on resource-constrained platforms are thus situated near the origin of the x-axis, whereas applications running on powerful servers in the cloud are situated at the other end of the x-axis. The area close to the origin of the y-axis corresponds to a level of detail of a few centimeters. The other side, where applications such as construction site monitoring, real-time automated inspection and critical infrastructure management are situated, corresponds to a level of detail of a few millimeters.

The trade-off ‘accuracy versus processing time’ for the state-of-the art is indicated by the cyan line, whereas for our own solution it is represented by the blue line. The gray zone in between the two curves denotes which applications are facilitated by our contributions. In the following, we will highlight the main applications and their mutual relation.

For *inventory management*, it is not necessary to have a sub-centimeter accurate 3D model. If the goal is to obtain automated stock management in a factory or warehouse, for example, a rough 3D model with centimeter resolution is sufficient to count the number of boxes or palettes that are present. Also in the case of real-time navigation for robots in controlled environments, it is not necessary to have every object in the scene in sub-centimeter detail, as long as the rough convex hulls of the objects are precisely located. However, real-time navigation is far more time critical than inventory management. For the latter application, the time per frame that can be spend to build or update the 3D map is a matter of seconds or even minutes rather than milliseconds for real-time navigation. On top of that, 3D mapping in view of real-time navigation usually has to be conducted on a resource-constrained platform, such as an embedded computer in a robot or drone. Inventory management on the other hand can be executed on a powerful computer with lots of processing power and memory.

In terms of accuracy and detail, the *rescue application* is situated in between real-time inspection and inventory management. They do not need as much detail as for inspection purposes, but still more than for inventory management as it should still be possible to analyse how destroyed buildings look like, in order to find survivors. This analysis requires additional resources from the system and hence there are less resources available for the actual 3D mapping.

Finally, there are applications for which the accuracy and detail is of utmost importance, typically in the range of 1 to 10 millimeters. Some of them are not time-critical and can be conducted off-line, such as *criticial infrastructure management*, in which civil engineers for example need to verify whether infrastructure starts to degrade, such as crack forming in concrete. The size of those cracks is typically a few millimeters. Other examples are the mapping of *archaeological* or *geological* sites.

For the latter applications, current 3D mapping solutions can deliver sufficiently accurate 3D models in case the scene can be captured using static scanning and in case enough resources are available. However, sometimes high precision and detail is needed while the time to compute the 3D map is limited. One example is continuous *construction site monitoring* in view of the *'built-as-planned'* use case. For that application, the goal is to map the construction site at fixed time intervals, in order to check if there were made any mistakes. If so, the errors can be detected early on and can be corrected in a reasonable amount of time and at a reduced cost.

1.2.3 Projects

The work of this PhD dissertation was conducted as part of a number of projects with both academic and industrial partners during the period of 2014 until 2019. More specifically, two projects in the domain of augmented reality were the seedbed of this research: the projects GIPA and ARIA. In both projects we investigated the use of depth modalities to map the environment of the user in view of augmenting it with visual information. In addition, we participated in a development cooperation with Vietnam on the topic of guiding visually impaired people. The outcomes of the project GIPA have also lead to the development of *Liborg*, a robot able to generate highly accurate 3D point cloud models in a very fast way.

1.2.3.1 GIPA

The ICON project GIPA or *Generic Interoperability Platform for Augmented reality applications* took place in 2014 and 2015. The aim was to develop a generic AR platform able to serve a whole range of applications including but not limited to remote collaboration, remote expert aid, serious gaming and training, critical infrastructure management and disaster or crisis support. As an example use case, we focused on the application of *'as-built-as-planned'*, in which the goal is to give architects the ability to verify whether or not a construction is built according to their plans, cfr Figure 1.2. The main partner in this use case was Sweco Belgium[1], formerly known as Grontmij.

To achieve the goal, a system had to be designed that is able to obtain reconstructions from large-scale environments with sub-centimeter accuracy in a 'reasonable'

[1]`https://www.swecobelgium.be/en/`

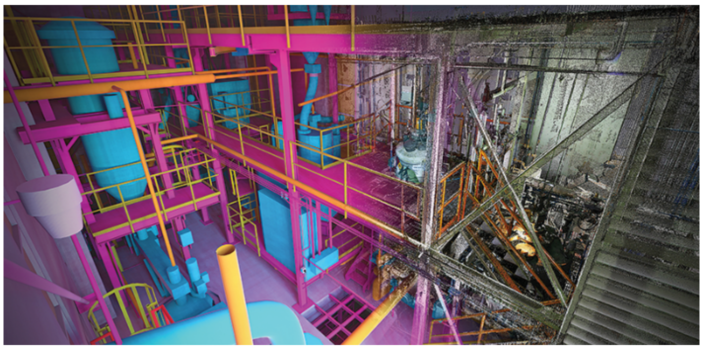


Figure 1.2: During the GIPA project, we focused on the application of *'as-built-as-planned'*. In that use case, an architect is given the ability to go on site and verify whether or not everything is built according to the plans.

amount of time. As mentioned before, the problem of 3D reconstruction or mapping is closely related to the one of localization. The main challenge here is that the system had to be independent from any external positioning system such as GPS or any other Global Navigation Satellite System (GNSS) as it also had to work indoor or in environments where the GPS signal is too unreliable. Typical examples are industrial or chemical plants with lots of overhanging pipelines, causing the GPS signal to be lost or highly inaccurate at times.

In summary, the software had to be able to perform 3D mapping - and thus also 3D positioning - based on data originated from depth sensors. In the end, we succeeded to build a system, fully independent of external positioning devices, that is able to deliver 3D point cloud models on the fly (in near real-time) with sub-centimetre accuracy. The results and outcomes of this research were presented and published at two international conferences [Vlaminck 16b; Luong 16] and in an international journal [Vlaminck 16a].

For the second part of the use case, algorithms had to be developed to match the reconstructed 3D model - initially represented by a point cloud - with an existing *computer-aided design* (CAD) model or *building information model* (BIM) that is available in a remote database. The latter process is often referred to as the *scan2BIM* problem. It had to be possible to overlay this CAD or BIM model in the view of an architect wearing a head-mounted device, thereby pointing out potential anomalies. Figure 1.2 depicts the idea of the as-built-as-planned use case. At the end of the project, we could successfully fit existing geometric primitive models - such as cylinders and planes - to the scanning data in real-time. In doing so, we made it possible to verify whether or not the constructions on site were correctly carried out.

1.2.3.2 ARIA

The ICON project *Augmented Reality for Industrial Applications* was the follow-up project of GIPA. This time, the goal was to develop a guidance application for maintenance tasks on complex industrial machinery. By means of proof of concept we focused on the replacement of an oil filter in a cooling container of the company Evonik in Antwerp. More specifically, we made it possible for a layman to be guided using visual cues while wearing a head mounted device. Our main contributions in the project were the development of an accurate 3D position tracking system that uses depth sensing, more precisely the output of a time of flight (ToF) camera. We developed a system for finding the 3D pose of the camera relative to a known BIM model available in a remote database. The latter allowed to query additional (semantic) information of the components of the machine and to overlay this information in the field of view of the user. As this application is even more time-critical than the one in GIPA, we further investigated how to speed-up the 3D pose tracking in order to make it real-time. Our improved algorithm was presented at an international conference [Vlaminck 17c]. The final tracking approach incorporating the matching with an existing BIM model, was presented at another international conference [Vlaminck 17a].

1.2.3.3 Guidance of visually impaired people

During 2014 and 2015 we participated in two development cooperation projects with Vietnam funded by Nafosted-FWO and VLIR-UOS. The goal in those projects was to built a system to improve the mobility of visually impaired people by providing them a navigation system to easily find their way to their destination. To this end, algorithms were developed to make a 3D reconstruction (or map) of the local neighbourhood of the blind person. This 3D map is then used to point out potential obstacles on their way and to give them instructions on how to navigate. This work has led to a working prototype that was eventually tested in the Nguyen Dinh Chieu blind school in Vietnam. The main results and outcomes are described in the publications [Vlaminck 13; Vlaminck 16c].

1.2.4 The Liborg mapping platform

After the project GIPA, there was a strong interest from industry in the automated process of 3D reconstruction in all kinds of environments and applied to different use cases. Besides the development of efficient 3D mapping algorithms, we also needed an efficient acquisition platform to easily collect data from. Soon after the GIPA project ended, we started with the development of a rover that can map its environment. In a later stage, we also made it possible to perform live mapping of

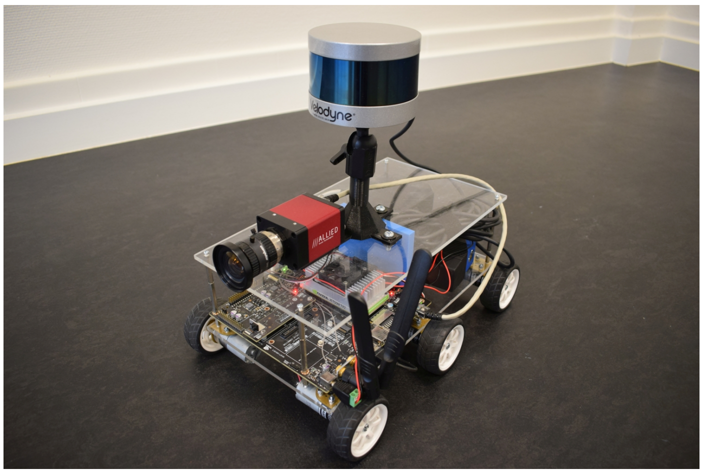


Figure 1.3: Liborg, our developed robot able to perform live 3D mapping of the environment.

the scene on the embedded computer of the robot, i.e. the NVIDIA Jetson TX2.

In addition, we developed a system to transmit the reconstructed 3D model to a remote computer or server. The latter allows a number of additional applications for which the robot can be used such as live monitoring of the acquired 3D model, for instance to perform inspection or assess damages in areas that are difficult to reach. We made it possible to control the robot using a remote control and thus to define the area to be mapped. Recently, we started the development of a truly autonomous system in a way that it will be possible for the robot to perform navigation based on its own acquired 3D model. This will allow the robot to perform the entire 3D mapping of a scene without any manual intervention.

The main sensor on the rover is a lidar scanner, hence the name *Liborg*. At a later stage we integrated a regular camera on the robot to be able to combine the 3D geometric information of the scene with visual data. To that end, a synchronisation module was developed along with calibration algorithms to relate the data of both sensors. The work on Liborg has been published and presented at an international scientific conference [Vlaminck 17b]. The robot was also demonstrated at the international technology forum (ITF) of imec in 2017 and 2018 as well as during the anniversary exhibition of Ghent University in 2017. The development of Liborg was also highlighted in the Imec magazine of October 2018 [Vlaminck 18b].

1.3 List of publications

1.3.1 Publications in international journals

- Michiel Vlaminck, Hiep Luong, Werner Goeman, and Wilfried Philips. “3D Scene Reconstruction Using Omnidirectional Vision and LiDAR: A Hybrid Approach”. In: *Sensors* 16.11 (2016)
- Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Have I Seen This Place Before? A Fast and Robust Loop Detection and Correction Method for 3D Lidar SLAM”. in: *Sensors* 19.1 (2018)

1.3.2 Publications in international conferences

- Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Surface-based GICP”. in: *Conference On Computer and Robot Vision (CRV)*. Toronto, Canada: IEEE, 2018
- Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “A markerless 3D tracking approach for augmented reality applications”. In: *2017 International Conference on 3D Immersion (IC3D)*. Brussels, Belgium, Dec. 2017, pp. 1–7
- Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Liborg: a lidar-based robot for efficient 3D mapping”. In: *SPIE Applications On Digital Imaging*. Vol. 10396. 1. San Diego, USA: SPIE, 2017
- Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Multi-resolution ICP for the efficient registration of point clouds based on octrees”. In: *International Conference On Machine Vision Applications (MVA)*. Nagoya, Japan: IAPR, 2017, pp. 334–337
- Hiep Luong, Michiel Vlaminck, Werner Goeman, and Wilfried Philips. “Consistent ICP for the registration of sparse and inhomogeneous point clouds”. In: *International Conference on Communications and Electronics (ICCE)*. Ha Long, Vietnam: IEEE, 2016, pp. 262–267
- Michiel Vlaminck, Hiep Luong, Werner Goeman, Peter Veelaert, and Wilfried Philips. “Towards online mobile mapping using inhomogeneous lidar data”. In: *Intelligent Vehicles Symposium (IV)*. Gothenburg, Sweden: IEEE, 2016, pp. 845–850
- Michiel Vlaminck, Hiep Luong, Hoang Van Nam, Hai Vu, Peter Veelaert, and Wilfried Philips. “Indoor assistance for visually impaired people us-

ing a RGB-D camera". In: *Southwest Symposium on Image Analysis and Interpretation (SSIAI)*. Santa Fe, NM, USA: IEEE, 2016, pp. 161–164

- Michiel Vlaminck, Ljubomir Jovanov, Peter Van Hese, Bart Goossens, Wilfried Philips, and Aleksandra Pizurica. "Obstacle detection for pedestrians with a visual impairment based on 3D imaging". In: *International Conference on 3D Imaging (IC3D)*. Liege, Belgium: IEEE, 2013, pp. 6–12

1.4 Outline

This dissertation is subdivided in nine chapters. In chapter 2, the principles of SLAM are explained. We elaborate on the questions why SLAM is needed and why it is not yet solved. We review the open challenges and outline the future of SLAM. Chapter 3 is devoted to the topic of point cloud registration as a solution to the problem of scan matching in SLAM systems. It describes the challenges that come with data originated from lidar scanners and ToF cameras. It also explains why the methods in the academic literature fall short on this kind of data. In chapter 4 we investigate more advanced techniques in view of making the pose estimation and mapping more accurate. Chapter 5 describes the topic of efficient map alignment and surface reconstruction and explains how they can both be exploited to further improve the alignment process and to make the mapping process faster. Chapter 6 deals with the generation of globally consistent maps and consists of two partial problems. The first one deals with the detection of loops in scanning data whereas the second one focuses on the correction of the drift error once a loop has been detected. Chapter 7 describes several use cases that we have been working on. Chapter 8 elaborates on localization using a hand-held ToF camera for use in AR applications. Finally, some concluding remarks and a look on the future are made in Chapter 9.

2 Simultaneous localization and mapping

This chapter will explain the concepts of the *simultaneous localization and mapping* problem, often abbreviated to SLAM. In summary, SLAM is the problem wherein a robot or autonomous vehicle has to build a map of an unknown environment in which it has to localize itself at the same time. For many applications of SLAM the main objective is to perform navigation for which a rough map of the environment is sufficient. Other applications however require a detailed map that can be used for comprehensive analysis, such as assessing damages in tunnels or detecting changes in natural phenomena such as caves.

To solve the SLAM problem, two questions need to be addressed. The first one is: *What does the environment look like?* It goes without saying that there are many ways of representing a scene and the answer to the question will largely depend on the application. In case of autonomous robots - where the primary goal is navigation - the map could be a collection of objects for which only the rough bounding boxes are stored. If the robot is only moving on the ground surface, the problem could be solved in 2D by maintaining a floor plan where obstacles are indicated. In case the goal is to assess damages in complex structures, a simple floor plan of the scene will of course not be sufficient.

The second question is: *Where are we?* The answer to this question only makes sense in case we have either a map of the environment or a reference system to which we can relate our current position. However, while conducting SLAM, the

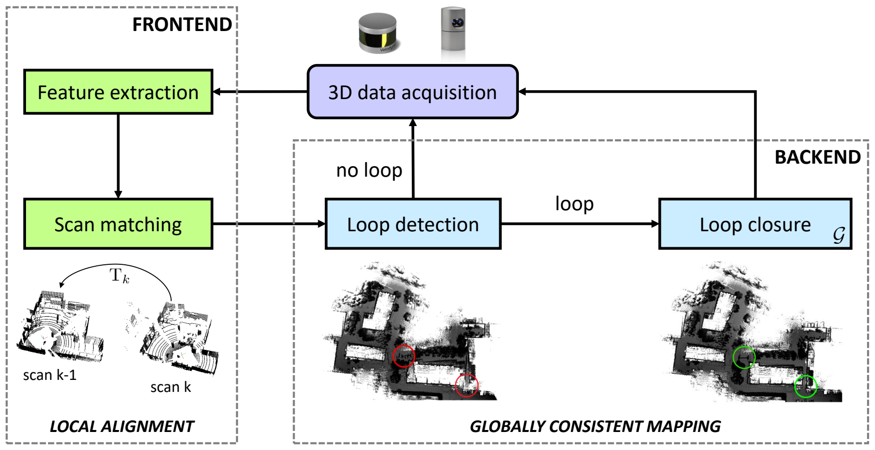


Figure 2.1: A SLAM system typically consists of two main components. The *local alignment* matches the current measurement (a scan) with the previous one and is often denoted as the *front-end* of the SLAM system. The *globally consistent mapping* module on the other hand deals with the reduction of drift, which is inherent to local scan matching. It usually integrates the detection and closing of loops in the sensor data and is also referred to as the *back-end* of the SLAM system.

goal is exactly to build this map, so by definition it is not present yet. Moreover, to build such map we need to know where we are, in order to be able to match our observations over time. This is leading to a typical interdependence or *chicken-or-egg* problem: to find out where we are we need a map of the environment, but as we don't have one yet, we need to build it and therefore we need to know where we are. This causality dilemma is one of the reasons why the SLAM problem is difficult to solve.

2.1 Components of a SLAM system

A full SLAM system typically consists of two main components. The first component is the *local alignment* comprising the matching of two consecutive measurements - referred to as frames or scans - while the second component is loop closure, enforcing the map to be *globally consistent*. The latter part is designed to cope with *drift* or error accumulation, which is inherent to sequential scan matching. No matter how accurate the local alignment is done, small errors will always be present and their accumulation will eventually lead to position estimates that are substantially deviating from the actual position. As a result, a second component is necessary to correct this drift, usually integrated using some kind of loop detection and correction algorithm. The two components are often referred to as the front-

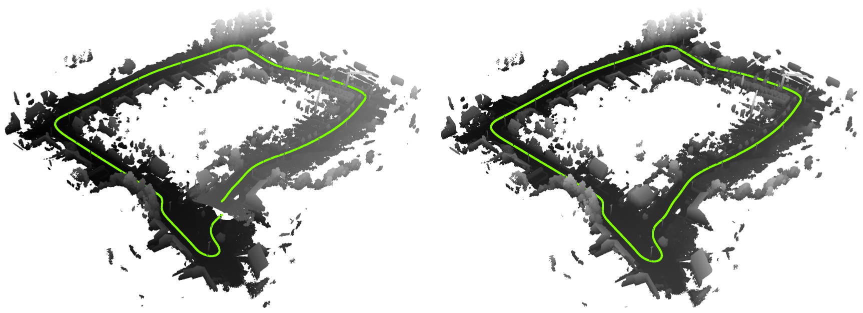


Figure 2.2: The accumulation of (small) errors through time can lead to large deviations in position estimates, known in the literature as the *drift* problem. The green line indicates the estimated trajectory of a mobile mapping van that was driving in a loop. In the left image, the loop is not correctly closed and the green line is not connected. One can see that the drift here is mainly occurring in the z-direction perpendicular to the ground plane (difference in shade denotes difference in elevation). The right image depicts the result after loop closure. The start- and endpoint are now connected.

and the back-end of the SLAM system.

Figure 2.1 depicts a schematic drawing of the components of a SLAM system. Usually, the front-end also comprises some kind of preprocessing to reduce measurement noise or to convert the data into a more robust representation. Figure 2.2 shows the principle of closing a loop: the left reconstruction is the result before whereas the right one is the result after loop closure. As can be seen in the left image, the green line - indicating the position of the vehicle through time - is not connected while the mobile mapping van stopped at the exact same position from where it has started. In the right image, the loop was closed and the green line is perfectly connected at the start- and endpoint of the acquisition.

It is important to note that loop closing is a necessary component in a full SLAM system as it is inevitable that small errors are made. When closing the loop, the accumulated error can be largely reduced. However, it does not mean that after loop closure, the map is fully accurate at the 'local' level. The quality of the scan matching still has its influence on the final obtained point cloud as is graphically illustrated in Figure 2.3. The reconstructed 'map' in the top row - metaphorically represented by a puzzle - still suffers from the small misalignments that were made during the pairwise alignment step. On the other hand, the reconstruction in the bottom row is highly accurate thanks to very precise pairwise alignments. In summary, one can say that loop closure enforces the error between any two points in the reconstruction to be bounded (as the drift is taken away), but it does not assure that the point cloud on the local level is fully accurate.

The quality of the pairwise alignment can be further improved when we keep

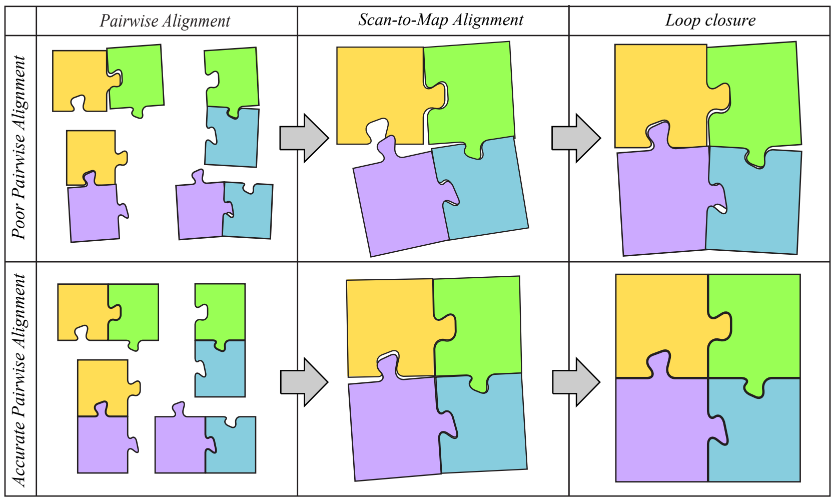


Figure 2.3: The three consecutive steps in a SLAM system: 1) pairwise alignment, 2) scan-to-map alignment and 3) loop closure. The two rows show that the quality of each of the three steps is important. In case the pairwise (or scan-to-map) alignment is inaccurately conducted, the loop closure algorithm can not fully 'recover' from that and discrepancies will remain in the final map. This is shown in a metaphorical manner by the puzzle pieces in the right upper corner not being perfectly fitted.

track of the currently built map. In that case we can relate (or align) every new scan with a larger part of the scene which alleviates the search for corresponding points and which on its turn leads to more robust and accurate alignments. This so called scan-to-map alignment is graphically depicted in the second column of Figure 2.3. While keeping track of the 3D model under construction, we can continuously improve it by estimating the underlying surface using all the points that have been captured so far. The only drawback of this approach is the size of the map potentially becoming too large to conduct the alignment in a reasonable amount of time.

The research on point cloud alignment or *registration* is described in detail in chapter 3. The methods regarding scan-to-map alignment are extensively described in chapter 5. Finally, the loop closure techniques to obtain globally consistent maps are explained in chapter 6.

2.2 Problem formulation

Since the seminal paper of Lu and Milios [Lu 97], the *de facto* standard formulation of a SLAM system is as a maximum a posteriori (MAP) estimation. Numerous

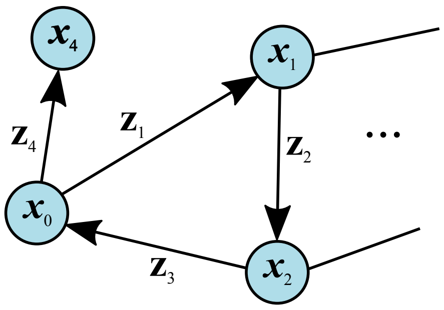


Figure 2.4: In SLAM, the formalism of *factor graphs* is often employed to reason about the interdependence among variables.

articles have been published since then, presenting improvements over the efficiency and robustness of the optimization underlying the problem. Suppose that we want to estimate the trajectory $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_k, \ldots, \mathbf{x}_M)$ of a robot or moving vehicle, as a discrete sequence of poses $\mathbf{x}_k = (x_k, y_k, z_k, \theta_x, \theta_y, \theta_z)$. At each pose $\mathbf{x}_k$ the sensor, either a lidar scanner or ToF camera, is capturing a point cloud $\mathcal{P}_k$. The estimation of the poses is based upon a series of M 'measurements' $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_k, \ldots, \mathbf{z}_M)$ such that each measurement can be expressed as a function of $\mathbf{x}$. For now, we define the measurement equation as the 'difference' between two consecutive point clouds and for simplification, it is assumed to be linear. The difference $\mathbf{z}_k$ is based on the estimated point correspondences in two consecutive point cloud $\mathcal{P}_k$ and $\mathcal{P}_{k-1}$, derived in the scan matching process. Its actual derivation will be explained in chapter 6.

While the true difference $\mathbf{z}_k$ is exactly what we are looking for, we can only make an observation $\hat{\mathbf{z}}_k$ of the true underlying difference, modelled as $\hat{\mathbf{z}}_k = \mathbf{z}_k + \mathbf{e}_k$ where $\mathbf{e}_k$ accounts for random measurement noise and estimation errors from the scan matching process. In MAP estimation, we estimate $\mathbf{x}$ by computing the trajectory $\mathbf{x}^*$ that maximizes the posterior $p(\mathbf{x}|\mathbf{z})$ (the *belief* over $\mathbf{x}$ given the measurements):

$$\mathbf{x}^* \doteq \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x}|\hat{\mathbf{z}}) = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\hat{\mathbf{z}}|\mathbf{x})p(\mathbf{x}), \tag{2.1}$$

where the second equality follows from the Bayes theorem. In this equation, $p(\mathbf{z}|\mathbf{x})$ is the likelihood of the measurements $\mathbf{z}$ given the assignment $\mathbf{x}$ and $p(\mathbf{x})$ is a prior probability over $\mathbf{x}$. When we consider $p(\mathbf{x})$ to correspond to a constant (uniform) distribution, it can be dropped from the optimization. In that case the MAP estimation reduces to *maximum likelihood estimation*. Assuming that the measurements $\mathbf{z}$ are independent, i.e. the corresponding noises or errors are uncorrelated, eq. 2.1 factorizes into:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x}) \prod_{k=1}^{M} p(\hat{\mathbf{z}}_k|\mathbf{x}) \tag{2.2}$$

$$= \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x}) \prod_{k=1}^{M} p(\hat{\mathbf{z}}_k|\mathbf{x}_k), \tag{2.3}$$

where $\mathbf{z}_k$ only depends on $\mathbf{x}_k$. We assume that the measurement noise or error $\mathbf{e}_k$ is a zero-mean Gaussian with information matrix $\mathbf{\Omega}_k$ (inverse of a covariance matrix). Using $\mathbf{e}_k = \hat{\mathbf{z}}_k - \mathbf{z}_k$, the measurement likelihood of 2.3 then becomes:

$$p(\hat{\mathbf{z}}_k|\mathbf{x}_k) \propto \exp(-\frac{1}{2}\mathbf{e}_k^\mathsf{T}\mathbf{\Omega}_k\mathbf{e}_k) \tag{2.4}$$

and similarly for the prior:

$$p(\mathbf{x}) \propto \exp(-\frac{1}{2}\mathbf{e}_0^\mathsf{T}\mathbf{\Omega}_0\mathbf{e}_0). \tag{2.5}$$

As maximizing the posterior is the same as minimizing the *negative log-posterior*, the MAP estimate becomes:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} - \log\left(p(\mathbf{x}) \prod_{k=1}^{m} p(\hat{\mathbf{z}}_k|\mathbf{x}_k)\right) \tag{2.6}$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{k=0}^{M} \mathbf{e}_k^\mathsf{T}\mathbf{\Omega}_k\mathbf{e}_k, \triangleq \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{F} \tag{2.7}$$

which is a non-linear least squares problem.

Often times, the formalism of *factor graphs* is employed to reason about the interdependence among variables. A factor graph is a graphical model that provides an insightful visualization of the problem. Figure 2.4 shows an example of a factor graph underlying the SLAM problem whereas Figure 2.5 depicts the relation between the expected 'difference' $\mathbf{z}_k$ and the observation $\hat{\mathbf{z}}_k$. Because of this graphical representation, the maximum a posteriori SLAM formulation is also called *graph-SLAM*. Sometimes the term *smoothing* is used to emphasize the difference with traditional (non-linear) filtering approaches, such as SLAM based on particle filters or the extended Kalman filter (EKF). The MAP estimation has been proven to be more accurate and more efficient than their counterparts based on Bayesian filtering.

2.3 Relation to other research domains

Several research fields are closely related to SLAM, including *Structure from Motion* (SfM) and photogrammetry. The latter two domains involve the reconstruction of a

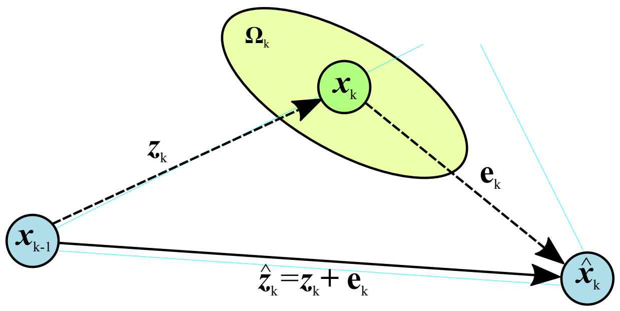


Figure 2.5: Principle of an edge connecting two vertices $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$ in the graph SLAM formulation. While $\mathbf{z}_k$ is the correct transformation between the two poses, we can only derive an estimate $\hat{\mathbf{z}}_k = \mathbf{z}_k + \mathbf{e}_k$ on the transformation.

scene by moving a regular camera in 3D space. The goal there is to obtain either a large *stitched* image - constructed by concatenating multiple single images - or in some cases a colored 3D model. Both SfM and photogrammetry are based on images and hence projective geometry. On the other hand, SLAM is not limited to the output of a camera but can also involve data from lasers, wheel odometry, inertial measurement units (IMUs) or even radar. In the case of SfM, eq. 2.7 can be rewritten as follows:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{\langle i,j \rangle \in \mathcal{C}} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^\mathsf{T} \boldsymbol{\Omega}_{ij} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}), \tag{2.8}$$

where $\mathcal{C}$ is the set of images, $\mathbf{x}_i$ and $\mathbf{x}_j$ the parameters, $\mathbf{z}_{ij}$ and $\boldsymbol{\Omega}_{ij}$ respectively the mean and the information matrix of a constraint relating the parameters $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ a vector error function that measures how well the parameter vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ satisfy the constraint $\mathbf{z}_{i,j}$.

Solving this equation constitutes the final step in the SfM process and is usually referred to as *bundle adjustment* (BA). Finally, SLAM is also related to *odometry*, referred to as *visual* odometry in case cameras are used. Whereas SLAM also involves map building, visual odometry only comprises the estimation of the position and orientation of the camera.

2.4 Applications

One of the main applications of SLAM has yet been mentioned: autonomous robots. This group of applications is however very broad. For example, it includes domestic robots providing services to elderly people for example and robots that can be used to clean or supervise public places such as train stations and conference halls. It

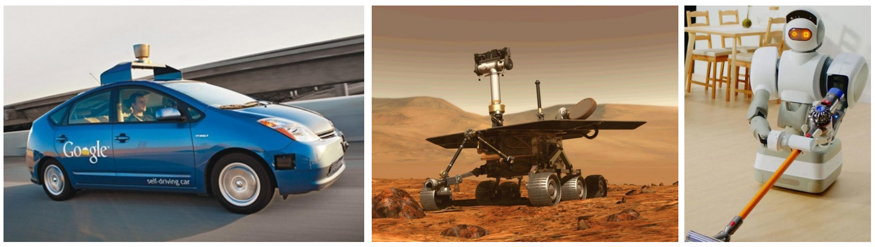


Figure 2.6: Three examples of SLAM applications: self-driving cars, Mars rovers and domestic or elderly care robots.

could also involve robots exploring difficult to reach areas such as tunnels, sewer systems, caves, etc. The goal there can be to inspect the infrastructure, assess damages or simply to study the evolution of natural phenomena. The goal could also be to perform rescue operations in disaster cases such as earth quakes or floodings. In those scenarios, the 'robot' might be better equipped with caterpillar tracks or it might be integrated in a drone or mounted on a boat. A few examples of SLAM applications are depicted in Figure 2.6.

Besides autonomous robots, there are other applications that require a 3D map (or model) from the scene and that are as time critical as autonomous robots. A clear example is *augmented reality* (AR). In AR, the purpose is to augment a real-world scene by means of virtual objects that are projected in the field of view of the user. In order to be able to augment the environment, one needs to know how it looks like and thus a 3D model is needed.

A third, perhaps slightly less obvious application is the guidance of visually impaired people in navigating themselves in undiscovered environments. Also there, the assistance system needs some kind of awareness on how the environment around the user is built up, in order to indicate potential obstacles. These three application domains were the main focus of this PhD thesis and even tough there are a few differences between them, the larger part is common and as a result many techniques and ideas are interchangeable.

2.5 Sensors

The main difference between the field of autonomous robots and AR is the type of sensors that can or should be used to conduct SLAM. As mentioned in the previous sections, one way of building a map of the environment is by using cameras. However, a lot of limitations pop up while using these devices. Images taken from the same scene can vary a lot depending on the amount of light that is present in the scene. Often times images are overexposed or they suffer from shadows or specular

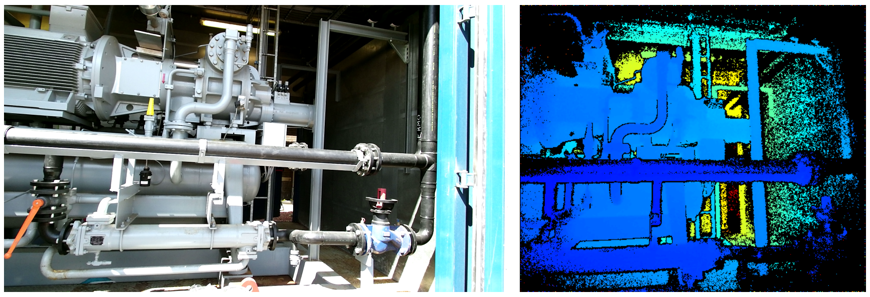


Figure 2.7: Color image (left) and corresponding depth map (right) captured by a Kinect 2 device. Blue pixels in the depth map represent points that are nearby whereas yellow or red pixels denote points that are further away. Missing depth values - represented by black pixels - can occur because the object is either too close or too far away or in case the surface is reflecting too much light.

light properties of objects. Also, they can not be used in dark environments such as tunnels, sewers or caves. It is therefore no surprise that new developments in SLAM were often triggered by the advent of novel 3D sensing modalities. In case of autonomous robots, the availability of laser range finders enabled the creation of real-time SLAM systems. Regarding AR, the advent of handheld RGB-D devices accelerated the development of immersive AR applications.

In this work we focused on different sensing technologies that are able to perceive depth, including *structured light*, *time-of-flight* (ToF) and *pulsed lidar*. All of these modalities have different specifications regarding the range at which they can still capture depth as well as the accuracy of their measurements. If the goal is to obtain a high resolution 3D reconstruction for inspection purposes, one should obviously opt for a sensor that is highly accurate even when it comes with a higher weight, power consumption or cost. In case of AR applications, it is more important that the sensor can be integrated in a *head mounted device* (HMD) and therefore it should be very lightweight and low in power consumption. On the other hand, AR applications do not need depth information for areas further than a few meters away and the accuracy of the system is subordinate to its reliability and robustness.

The early *hand-held* depth sensors on the market were based on structured light. Examples are the Microsoft Kinect and Softkinetic's DepthSense. Unfortunately, a severe drawback of this technology is that it performs poorly in the presence of abundant light, which is always the case for outdoor environments. For that reason, Microsoft later on developed the second version of the Kinect, which is now based on ToF. As a consequence, both the accuracy and resistance against high illumination are improved, but it still performs badly in direct sunlight. Microsoft eventually integrated the same ToF sensor in their AR device Hololens. Figure 2.7 shows the output of the Kinect 2, comprising a color image with a resolution of

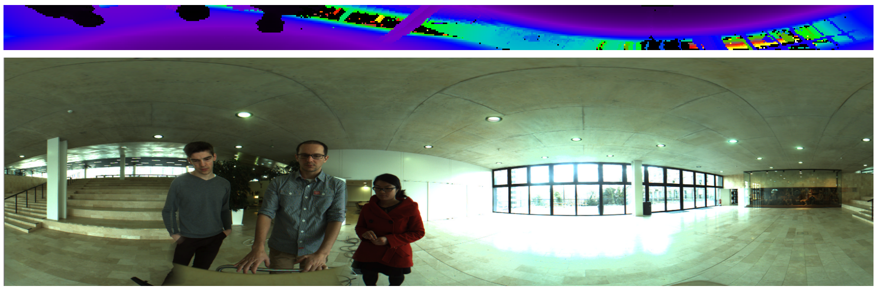


Figure 2.8: Depth image derived from the output of the Velodyne HDL-32E in the UFO building at Ghent University. The vertical resolution of this depth image is 32 in accordance with the number of lasers and the vertical FOV covers approximately 40°. As the scanner is spinning its head, the horizontal FOV comprises 360°. The picture at the bottom was captured with a Ladybug camera system and is shown to clarify the depth image.

1920×1080 and a depth map with a resolution of 512×424. The range of the ToF sensor approximates 4 m, which is sufficient for AR applications, but is too limited for navigation in autonomous robots or building 3D models of large scale environments.

For the latter use cases, pulsed lidar technology is more appropriate. These devices typically consist of a spinning head containing multiple collinear lasers. The left image of Figure 2.9 depicts three models (left to right: VLP-16, HDL-32 and HDL-64) of the Velodyne lidar scanners containing respectively 16, 32 and 64 lasers. The range of these scanners is approximately 100 m for the former two and 120 m for the latter. As they are continuously spinning their head, they produce depth measurements with a horizontal FOV of 360°. The vertical FOV of these scanners is respectively 30°, 40° and 26.9°. An example depth image acquired by the Velodyne HDL-32E is shown in Figure 2.8. The right image in Figure 2.9 depicts two lidar scanners from the competitive company Ouster, i.e. the OS-1 and OS-2.

Finally, a new imaging technology that recently receives a lot of attention is *gated imaging*. In gated imaging, a camera with tightly controlled opening and closing times of the shutter is used in combination with a high power pulsed light source, often a laser. The pulsed source is synchronized with the sensor exposure time, which makes the latter able to integrate a specific segment of the reflected light and hence to image a slice at a specific depth in a scene. In this way, one can choose to observe, for instance, only slices located at more than 1km from the camera.

Usually, gated imaging is used to enhance image contrast for example in the

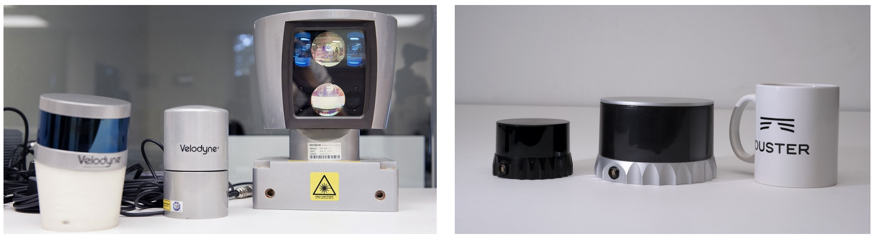


Figure 2.9: Velodyne and Ouster are currently the leading companies developing state-of-the-art 3D scanners based on pulsed lidar technology. Throughout this work, experiments have been conducted with the three different models from Velodyne depicted above, which consist of respectively 16, 32 and 64 lasers.

case of mist or other clutter. The shutter of the camera is kept closed, i.e. the camera is not integrating light, during the time that the clutter is incident. The shutter is opened only at the time of incidence of the light from the object of interest. This type of cameras can thus also be used to capture consecutive images at different depths, opening new opportunities in the domain of 3D mapping. Currently, the level of accuracy of gated imaging is in the range of a few meters, but one can expect this to increase in the future.

2.6 Evaluating the performance of SLAM systems

Many metrics have been used in the literature to evaluate the performance of SLAM systems. In general we can distinguish two different approaches. The first one compares the estimated trajectory of the mobile observer with the real, 'ground truth', one. In reality it will however never be possible to rely on the true ground truth trajectory. Therefore, measurements from an alternative - more accurate - device are often used to compare against. The second approach on the other hand compares the reconstructed 3D model with a 'ground truth' model. The same remark can be made here, in the sense that it is not always easy to obtain a correct 3D model of a scene, as it is exactly the purpose of SLAM systems to generate these. Sometimes, there exists a CAD model that was generated to construct the building or infrastructure. For research purposes, one could also opt to create a 3D (point cloud) model through *static laser scanning*. In the latter approach, measurements are taken from fixed, known, positions and hence, the result is only affected by the accuracy of the laser scanner, which in some cases can be as good as 1 or 2 mm.

When there is no ground truth information available, the last resort is to record a data sequence in which the sensor is returning to the same position from where it has left. In that case, one can consider the Euclidean distance between the first and last position, defined as the translational components of the pose, as the error. In order to make sure that a perfect loop is created, one can additionally 'add' the first measurement of the scanner (referred to as a 'scan' or 'frame') at the end of the data sequence. Doing so, we 'mimic' the fact that the robot is returning to the exact same location from where it has left, thereby eliminating manual 'navigation' errors.The latter approach is not the best way of expressing the accuracy of a SLAM system, but it can still be useful to compare different techniques with each other or to assess the contribution of a part of the solution.

In this work, we will use a combination of these evaluation methods. Regarding the comparison of trajectories, we will use two different error metrics, being the relative pose error (RPE) and the absolute pose error (APE), which are explained below. Both errors are extensively used in papers that benchmark several SLAM systems such as [Geiger 12], [Sturm 12] and [MurArtal 15].

2.6.1 Relative pose error

In a nutshell, the RPE measures the local accuracy of the trajectory, in terms of translational and rotational error, over a fixed distance (or time) interval. To this end, the trajectories are quantised in intervals of length Δ, for example 100 m. The sensor pose matrix $\mathbf{P}_i = (\mathbf{R}_i, \mathbf{t}_i) \in \mathbb{R}^{4\times4}$ corresponding with scan or sweep i represents the *first* sensor pose of the interval, whereas $\mathbf{P}_j = (\mathbf{R}_j, \mathbf{t}_j) \in \mathbb{R}^{4\times4}$ represents the *last* sensor pose of the interval. The matrices $\mathbf{R}_i$ and $\mathbf{R}_j$ represent two rotation matrices and the vectors $\mathbf{t}_i$ and $\mathbf{t}_j$ denote two translation vectors.

For $\mathbf{P}_i$ and $\mathbf{P}_j$ the following condition is valid: $||\mathbf{t}_i - \mathbf{t}_j||_2 \approx \Delta$. We now define the *difference* of the two ground truth poses $\mathbf{P}_i$ and $\mathbf{P}_j$ and estimated poses $\hat{\mathbf{P}}_i$ and $\hat{\mathbf{P}}_j$ as $\mathbf{D}_{i,j} = \mathbf{P}_i^{-1}\mathbf{P}_j$ and $\hat{\mathbf{D}}_{i,j} = \hat{\mathbf{P}}_i^{-1}\hat{\mathbf{P}}_j$ respectively. The relative pose error $\mathbf{E}_{i,j}$ is then given by:

$$\mathbf{E}_{i,j} = \left[\begin{array}{c:c} \mathbf{R}^e_{i,j} & \mathbf{t}^e_{i,j} \\ \hdashline \mathbf{0}_3^\top & 1 \end{array}\right] = \hat{\mathbf{D}}^{-1}_{i,j}\mathbf{D}_{i,j}. \tag{2.9}$$

The rotational error $r_{i,j}$ and translational error $t_{i,j}$ between the two poses are subsequently given by:

$$d_{i,j} = \frac{1}{2}(\mathrm{tr}(\mathbf{R}^e_{i,j}) - 1), \tag{2.10}$$

$$r_{i,j} = \frac{1}{\Delta}\mathrm{acos}(\max(\min(d_{i,j}, 1), -1)), \tag{2.11}$$

$$t_{i,j} = \frac{1}{\Delta}||\mathbf{t}^e_{i,j}||_2. \tag{2.12}$$

The rotational error $r_{i,j}$ is expressed in deg/m whereas the translational error $t_{i,j}$ is expressed in $\%$ and they can be computed for several $\Delta_k \in \mathbf{\Delta}$. The average absolute value of the rotational error E_r and the average translational error E_t are finally given by:

$$E_r(\mathbf{\Delta}) = \frac{1}{M} \sum_{\Delta_k} \sum_{(i,j) \in \tau_k} |r_{i,j}| \tag{2.13}$$

$$E_t(\mathbf{\Delta}) = \frac{1}{M} \sum_{\Delta_k} \sum_{(i,j) \in \tau_k} t_{i,j}. \tag{2.14}$$

The constant M represents the number of segments over all Δ_k's that are considered and τ_k denotes the set of indices (i, j) of the segments corresponding to Δ_k. In the Kitti benchmark [Geiger 12], for example, $\mathbf{\Delta} = \{$100 m, . . . , 800 m$\}$ and $M = 8$.

The RPE is useful as it mitigates somehow the influence of outliers in the estimated poses. More specifically, as pose estimates in SLAM are the result of dead reckoning, even the smallest (rotational) error is taken along the whole future trajectory. Therefore, it is useful to consider different Δ_k to investigate how robust the system is to outlier pose estimates, often the result of abrupt movements of the sensor or moving objects in the scene. Suppose that a large rotational error is made at a certain time, it will be taken along the whole future trajectory. If we quantise the trajectory in intervals of length Δ_k, we can more accurately verify whether the final error is due to one large error or many smaller ones.

2.6.2 Absolute pose error

The second error metric is denoted as the *absolute pose error* (APE). It is computed by comparing the absolute distances between the estimated and the ground truth trajectory. The absolute pose error $\mathbf{F}$ at time step i is computed as follows:

$$\mathbf{F}_i = \left[\begin{array}{c:c} \mathbf{R}_i^f & \mathbf{t}_i^f \\ \hdashline \mathbf{0}_3^\top & 1 \end{array}\right] = \hat{\mathbf{P}}_i^{-1}\mathbf{S}\mathbf{P}_i, \tag{2.15}$$

where $\mathbf{S}$ is the rigid-body transformation corresponding to the least-squares solution that maps the estimated trajectory $\hat{\mathbf{P}}$ onto the ground truth trajectory $\mathbf{P}$. This method is also referred to as the *'Umeyama'* method [Umeyama 91]. Eventually, we use the average L2-norm of the translational components of all pose indices, given by:

$$F = \frac{1}{N} \sum_i ||\mathbf{t}_i^f||_2, \tag{2.16}$$

where N denotes the total number of poses. The APE error metric is generally used to compare the performance of full-blown SLAM systems that have some kind of loop closure mechanism. It is less suited in case there is no loop closure present or when the observer is not returning to earlier visited places.

2.7 Open challenges

Narrowing down the use case to a few scans in a small environment and assuming a *static world*, current SLAM solutions can deliver acceptable performance. However, in case of many short-term dynamics, i.e. moving people or objects in the scene, or applications on resource-constrained platforms, many issues still need to be resolved. Not to speak about the case of *lifelong mapping* or providing long-term autonomy to robots. From that perspective, SLAM is far from solved. The term lifelong mapping refers to the process in which the map of the 'world' is continuously extended and improved. There are many challenges related to it that still remain open. In this section, we will organize them in two categories, namely the challenges related to the 'robustness' and the ones related to the 'scalability', both of them necessities to truly come to long-term autonomy. The first category roughly deals with failures and how to recover from them, while the second one involves efficient use of resources and the concept of distributed mapping.

2.7.1 Robustness

As SLAM systems are fragile in many ways, long-term operation is far from trivial. Failures can occur on the algorithmic level, but can also be hardware related. When a robot needs to operate autonomously on the long term these failures should be gracefully handled. This requires both algorithms that are *fail-safe*, but also *failure-aware*, i.e. the system needs to be aware of imminent failure and provide recovery mechanisms that can re-establish proper operation.

Robustness on the algorithmic level

One important case of algorithmic failure is the data association part of a SLAM system, also referred to as the correspondence problem. It deals with the question to which part of the reconstructed scene new sensor measurements belong to. On the one hand, there is the problem of *perceptual aliasing* in which different sensory inputs lead to the same *'signature'*, and which lead to wrong data associations. In point cloud based SLAM this is especially the case in monotone environments that lack geometrical features, such as long corridors. On the other hand, point-based registration methods also suffer from the *correspondence ambiguity* caused by the sparsity of the sensor measurements making perfect (physical) point matches impossible. Another aspect of robustness is when the robot or user is moving too fast or when there are harsh environments, e.g. in case of a drone flying in windy circumstances. Finally, in highly dynamic environments, the presence of moving objects or people in the scene makes the data association even harder.

Robustness to hardware failure

Besides the vulnerabilities on the algorithmic level, a SLAM system can also decline due to hardware failure or degradation. In case of structured light or pulsed lidar certain types of surfaces cause the emitted light to be distorted in a way that it interferes with the data acquisition process. Among these difficult surfaces are dark, shiny or transparent surfaces. Dark surfaces typically absorb light whereas shiny surfaces let the light scatter and bounce in uncontrollable directions. Finally, transparent surfaces, e.g., windows, let light go through which makes them invisible. Moreover, they will cause reflection which adds noise to the scans. Typically, the inside walls will be projected on the outside if one is scanning from inside and through a window. Also, the refraction of light causes it to be shifted, leading to a distortion of the acquired geometry. All these ‘special’ surfaces thus lead to inaccurate or missing measurements in the scanning data.

Besides the type of surface, some 3D sensing modalities, such as pulsed lidar, are also affected by bad weather conditions, often slashing the range and accuracy of the sensors. Finally, hardware failure could also be caused by malicious practices. Some studies have shown that lidar systems, for instance on driver-less cars, can be thwarted by a simple laser pointer. Furthermore, they are vulnerable to spoofing or saturation attacks causing a denial-of-service. All of the aforementioned potential hardware failures should be carefully recognized by the SLAM system. In addition, they should be gracefully dealt with in the sense that the system should not crash but try to produce a reasonable result that can later on be improved when the source of hardware failure is tackled.

In some cases the errors can be overcome on the software level, but in some cases this will not be possible. Therefore, future SLAM systems will use different sensors simultaneously such that one sensor can take over when another one is failing to produce correct measurements. Moreover, multiple different sensors can also be used to fuse the data together. This *sensor fusion*, makes the resulting information from the scene less uncertain then when the sensors are used individually.

Re-localization and recovery

No matter how hard we try to overcome the sources of failure, there will always be situations where recovery is impossible. Today, SLAM systems are generally not equipped with tools that enable recovery and hence they often require a hard reset in case of failure. Future SLAM systems should therefore integrate a way of saving the current map off-line and re-locating the sensor using a previously built map.

Time varying maps

Besides short term dynamical changes as result of moving objects or people, there could also be 'seasonal' changes. Nowadays, most SLAM systems assume a static environment where the 'world' remains unchanged. In future SLAM system there should be policies on which parts or which objects should have tracked changes and which ones should be 'forgotten'.

Automatic parameter tuning

The SLAM community today is largely affected by the *"curse of manual tuning"*. This means that many parameters in all different parts of a SLAM system require careful tuning for the system to produce satisfactory results. Among the parameters are the number of iterations while solving the underlying optimization, the thresholds for feature computation and defining correct data association and the criteria to trigger loop closure. Future SLAM systems will have to work "out of the box" and therefore they will have to integrate a way of automatic parameter tuning.

2.7.2 Scalability

The challenge of scalability is closely related to the one of lifelong mapping. The robots from the future, such as cleaning robots, or robots used for environmental or urban monitoring in ever changing environments will operate for an extended period of time and over large areas. This means that the maps they will create and maintain will become large as well. At the same time, the robotic platforms are bounded in terms of resources, both on computational and memory level. As a result, future SLAM systems should aim to keep the computation time and memory footprint low. In addition, they will have to work together and hence to be able to distribute their maps to their peers. There are many aspects in making a SLAM system truly scalable. Below, we list the most important ones.

Node and edge sparsification

Modern cameras, ToF sensors or lidar scanners usually operate at frame rates between 10 FPS to 30 FPS. Doing so, they induce many 'camera poses' or edges in the SLAM pose graph. When solving the MAP formulation, this number easily becomes intractable. Depending on the underlying solver, the complexity of this optimization is quadratic or linear in the number of variables. When visiting the same area over and over again, it is clear that keeping many nodes in the pose graph becomes less and less efficient in improving the overall map. Lifelong SLAM systems will therefore have to implement *pruning* mechanisms to reduce the number of poses in the SLAM graph. In line with this, recent studies are investigating so-called *continuous-time trajectory estimation* in view of reducing the number of parameters to be estimated over time. We will elaborate on this continuous-time SLAM in the next section.

Map representation

Besides the sparsification of the edges and the nodes in the graph, the representation of the map is another crucial part when it comes down to scalability. The map should fit in the memory of the 'mobile' observer and should facilitate quick updates. Even when memory is of less importance, for instance when the map is stored in the cloud, point-cloud based maps or volumetric maps are tremendously wasteful. Furthermore, those representations are too cumbersome to conduct spatial queries on, for example to relocate the user later on, detect changes and finally apply those changes in the map. As it would lead us too far to describe alternative map representations in detail here, we devoted an entire section in chapter 5 to efficient map representations.

Forgetting, learning, remembering

Because the world is constantly changing, there should be policies on when it is fine, if ever, to forget data. What can be forgotten and how can we track changes over time?

Distributed mapping

A research area that is still widely unexplored is the one of distributed mapping also referred to as *multi-agent* SLAM. However, as more and more intelligent robots will operate together, they should also share their knowledge with each other. There are two ways of distributed mapping: the *centralized* or the *decentralized* approach. In the former one, all robots build their own part of the map and send it to a central station that aggregates and re-distributes them. In the second one, each robot shares its own-built part of the map with all the other robots thereby trying to find a consensus among each other. One of the main unsolved issues regarding distributed mapping is the way to deal with outliers. As the robots might not share a common reference frame it is difficult to detect and reject wrong loop closures. Also, in a distributed map, the robots have to detect these outliers based on partial and local information. Finally, a limited bandwidth could impose additional challenges not to mention the case in which the communication link can be entirely down. A recent study on multi-robot SLAM can be found in [Saeedi 16].

Resource-constrained platforms

One severe limitation of today's SLAM algorithms is that they are computationally demanding. They work real-time on workstations or well-equipped laptops, but SLAM algorithms will be integrated in many smaller devices, where there is no space for an advanced chip including a large battery or fan. Think about mobile phones, micro aerial vehicles or even more exotic, robotic insects. Future SLAM algorithms should therefore foresee mechanisms to gently trade off accuracy for speed.

2.8 Future of SLAM

In this section we want to elaborate on a few recent developments which we believe will play a crucial role in the future of SLAM.

2.8.1 Continuous-time SLAM

In the previous section, the challenge of node and edge sparsification has briefly been mentioned. One way of dealing with this, is to identify relevant poses and prune the graph accordingly by removing uninformative edges, e.g. using an information-theoretic approach [Ila 10]. However, for a truly scalable solution, we believe that *continuous-time trajectory estimation* is more appropriate. In that paradigm, the full MAP problem is transformed into continuous time and temporal basis functions are used to keep the state size manageable. Another benefit of this approach is linked to the problem of a so-called *rolling shutter* which is present in typical *pulsed* or *scanning* lidar devices. The rolling shutter causes a *serial* acquisition of 3D points instead of one simultaneous scan of the environment. Since the robot or vehicle is typically moving, this rolling shutter causes the produced point clouds to be distorted or *skewed* as depicted in the left image of Figure 2.10. Sometimes, the lidar scanner is accompanied with a motion sensor, such as a wheel sensor or an inertial measurement unit (IMU), which can be used to undistort the lidar data and accumulate it into a rigid *frame*, cfr. right image of Figure 2.10.

In the work of [Alismail 14], the authors abandon the idea to treat one lidar

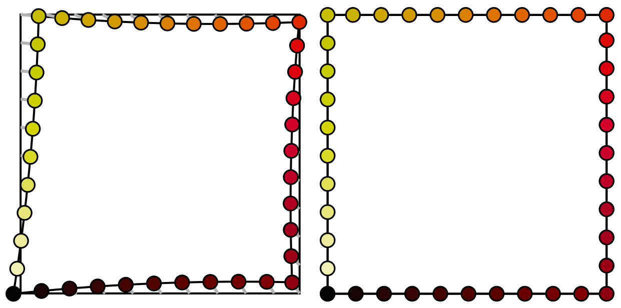


Figure 2.10: The phenomenon of a skewed point cloud (left) due to the rolling shutter of a *moving* lidar scanner. The process of de-skewing (right) aims at un-distorting the point cloud by transforming the points in a way as if they were all captured simultaneously. Image reproduced from [Alismail 14].

sweep as a single rigid frame and propose a registration technique that explicitly accounts for sensor motion, hence ending up with a continuous-time trajectory. Other researchers have investigated cubic-splines or B-splines to represent the trajectory of the robot in continuous-time [Furgale 12] . The control points or 'knots' of the spline are then representing the nodes in the graph. These B-spline methods have been successfully applied on both landmark-based as well as direct dense representations [Kim 16]. Besides for rolling shutter sensors [PatronPerez 15], B-spline methods have also been explored for event-based cameras [Mueggler 15]. Other research proposed improvements regarding 'sliding-window' B-spline formulations or proposed techniques based on non-uniform splines or wavelets. More recently, hierarchical optimization back-ends are explored [Droeschel 18], in which local multi-resolution maps are used in a multi-level graph, cfr Figure 2.11. The individual 3D scans in the local map are modelled as a sub-graph and graph optimization is performed in the local maps to account for drift and misalignments. Thereby, in each sub-graph, a continuous-time representation of the sensor trajectory allows to correct measurements between scan poses.

2.8.2 Machine learning

Nowadays, the computer vision literature is dominated by deep learning, so we can not end this chapter without mentioning it. Recently, researchers have shown that it is possible to train a deep neural network able to regress the pose of a camera directly from the images [Costante 16], hence omitting the need for standard geometry or visual odometry. However, experiments on publicly available datasets including KITTI [Geiger 12], demonstrated that the accuracy is currently below the one of traditional approaches.

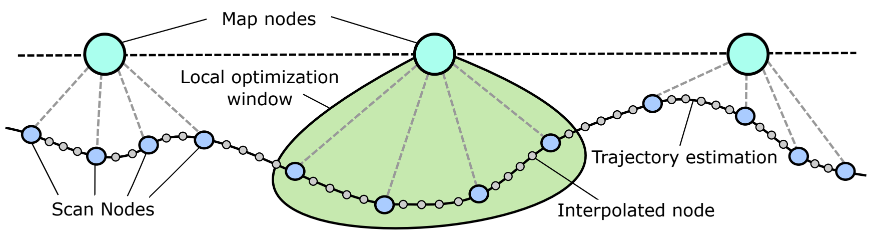


Figure 2.11: Multi-resolution continuous SLAM as presented in [Droeschel 18]. The entire map is represented by multiple *local* maps - hence hierarchical - that are locally optimized. For each local map, the trajectory is estimated continuously, making it possible to interpolate new nodes (or poses).

In [Kendall 15] the authors propose *'Posenet'*, a real-time 6-DOF camera re-localization system based on a convolutional neural network. At the same time, the authors of [Valentin 15] successfully applied regression forests to re-localize the camera. Other works involve the estimation of depth from a single image in order to use it to conduct monocular SLAM, such as *'CNN-SLAM'* presented in [Tateno 17]. Finally, in [Krishnan 15], the authors proposed Deep Kalman Filters as the first to explore the connection between deep learning architectures and recursive state estimation in large-scale graphical models. In our opinion, those recent advances do not mean that 'traditional' SLAM is dead and we consider it too soon to tell whether these studies are curiosities or will take over the traditional, well-understood principles of SLAM. In the following, we will discuss a few future directions of SLAM for which we think deep learning will nevertheless be of great influence.

The first one is related to the concept of semantic SLAM, which deals with the high-level understanding of the maps that are created. Semantic SLAM will enable task-driven decisions and anticipation based on the category of objects in the scene, including moving objects such as humans walking around. The level of detail and organization of the semantic labels will be depending on the specific task the robot has to perform. Nowadays, object recognition for the ImageNet classes can almost be treated as a black box that works well from the perspective of the robotics or SLAM researcher. Similarly, pixel-based semantic labeling reaches accuracy levels of more than 90% on the PASCAL VOC dataset [Sun 18]. These successes will definitely accelerate the deployment of semantic maps, which on its turn will ameliorate other aspects of SLAM including place recognition for use in loop closure or re-localization in case of tracking losses.

The second future research direction will be focused around practical deployment. Deep learning today is characterized by lengthy training times on supercomputers and dedicated GPU hardware to conduct inference. The question will be whether to wait for the special-purpose hardware that provides sufficient computation power for embedded devices or to investigate smaller, cheaper networks that provide ‘good enough’ results. Another issue is that future SLAM systems will typically operate in an open-world with continuous observation whereas current deep neural networks are trained on closed-world scenarios with a fixed number of classes. Also, where existing networks are trained on a large amount of data, it can not always be guaranteed that a suitable dataset exists for supervised training. A big challenge therefore remains in improving the power of deep neural networks towards ‘one-shot’ learning. The emphasis there lies on knowledge transfer, making use of prior knowledge of learnt categories and allowing to learn from minimal training examples. We believe that developments in this area will be necessary in order to enable life-long learning integrated in a continuously observing SLAM system. A final aspect is the one of incorporating prior knowledge on the expected type of objects, the characteristics of the scene or even relationships between the objects. Deep learning could be applied to distil such prior knowledge from the environment. However, the way to extract and use this information is a significant open problem, especially the way to deal with the uncertainty of estimates derived from a deep neural network.

2.9 Conclusion

In this chapter we formulated an answer to the questions whether SLAM is solved or not and why or in which situations SLAM is necessary. If we narrow down the scenarios to specific environments, use cases and limited performance, one could argue that within some boundaries SLAM is solved. However, as we enter the robust-perception age, there are still many open challenges that yet need to be overcome to truly come to robust perception and navigation for long-lived autonomous robots. From this perspective, SLAM is clearly not solved and more research on the topic is needed. The main unresolved aspects involve ‘robust performance’, ‘resource efficiency’, ‘high-level understanding’, and ‘task-driven inference’. From the point of robust performance, there is still plenty of work in making SLAM systems “fail-safe” and “self-tuning”. Resource efficiency involves techniques and data structures to maintain large-scale and time-varying maps that are easy to update and that can access the right information at the right time. In that respect we envision that future SLAM systems will be dominated by solid objects instead of point- or boundary-based systems. Regarding the time-varying maps, we furthermore need policies on which parts should have tracked changes and which parts are allowed to ‘forget’ information. High-level understanding comprises semantic labels to objects in the scene which will further ease the problem of dynamic objects or people in the scene.

3 Point cloud registration

In this chapter we will dive deeper into the problem of point cloud registration as part of a SLAM solution. In short, point cloud registration involves the alignment of two 3D point sets with partially overlapping scene content. In other words, the goal is to find the *rigid* transformation that yields the best alignment between two point clouds in order to maximize their overlap. This transformation is eventually used to merge the point clouds into one coherent coordinate system. The problem is obviously more difficult when the scene overlap decreases, but also when the point clouds originate from different acquisition modalities, or when they are acquired from very different view points. In the first case, fewer correspondences can be determined between the two point clouds. In the second case the point density or the noise properties could be deviating whereas in the third case occlusions could act as a hindrance.

In this work, we mainly focus on point cloud registration for use in SLAM applications where the same device is used during the entire process. Still, we can distinguish two different scenarios. In the first one, we want to align two point clouds that were acquired in two consecutive *sweeps* of a lidar scanner or two consecutive *frames* of a ToF camera, a problem that is usually referred to as *scan matching*. The main advantage here is that the two point clouds differ only slightly because they are acquired from only slightly different view points. Obviously, this property will only hold in case the frequency of the measurements is sufficiently high and the moving speed sufficiently slow, which we will assume in this chapter.

In the second scenario the *tracking* of the sensor is lost and we need to *relocalize*

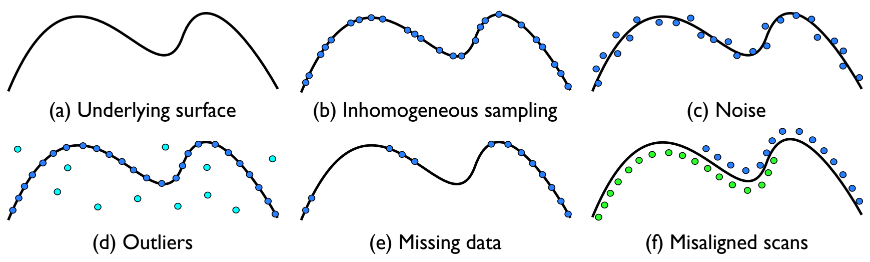


Figure 3.1: Several types of artefacts that can be present in point cloud data, here represented in 2D.

its pose. In that case, the assumption that the two point clouds are only slightly different no longer holds true. The situation can also occur when we revisit a location and we want to exploit this knowledge by *closing* the loop, thereby correcting for drift errors. In that case, the same location can be reached from different directions and hence the two acquired point clouds can have a (large) difference in orientation. In the next section we will enumerate a few challenges that come with the problem of point cloud registration. We will also explain why the methods in the earlier academic literature fall short and how we can deal with their shortcomings. The follow-up sections will logically describe how we tackled the main challenges.

3.1 Introduction

With the advent of many modern 3D sensing modalities, point cloud registration has become a prevalent part of a SLAM system. However, the registration of point clouds originating from different depth sensors poses more challenges. Figure 3.1 depicts the most common issues. Point clouds acquired by laser scanners for example often suffer from an inhomogeneous and sparse point density, due to the angle of incidence of the laser beams in combination with the proximity of objects in the scene. Other possible artefacts are due to sensor noise (inaccurate measurements) or outliers. Furthermore, it is possible that a large part of the scene is not sampled because of occlusion. Finally, in case of incremental registration, previously acquired point clouds could be imperfectly aligned. If we would use the concatenation of those point clouds later in the pipeline, we could facilitate future registration errors. Especially the latter problem is not well-addressed in state-of-the-art registration algorithms. All of the aforementioned artefacts should be carefully taken into account as they are almost always present in scan data, especially in data originated from lidar scanners.

Figure 3.2 depicts a point cloud from one of the sequences of the KITTI vision benchmark [Geiger 12], acquired using a Velodyne HDL-64E. In that point cloud

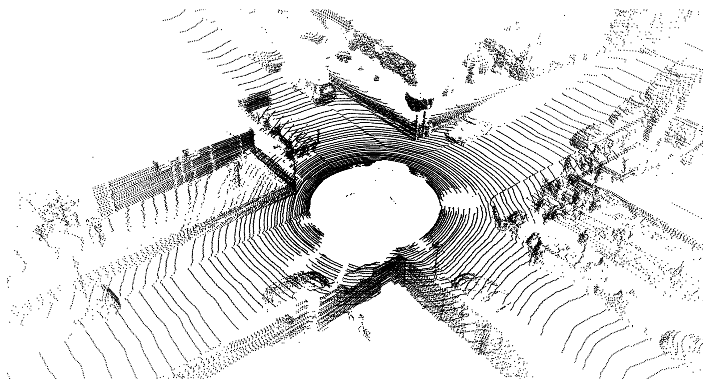


Figure 3.2: Point cloud from the KITTI benchmark [Geiger 12] acquired by a Velodyne HDL-64E lidar scanner in an urban traffic scene.

one can see rings, starting somewhere close to the origin and spreading towards the edges of the point cloud. One can also see that the distance between the rings is increasing the further we move away from the origin, which is due to the incidence angle of the laser beams. This property causes the inhomogeneous point density in lidar point clouds and makes it unlikely that the exact same physical point is sampled among consecutive scans. The latter problem gives rise to ambiguous correspondences, denoting a pair of matching points not representing the same point in the real world. This phenomenon is sometimes referred to as the *correspondence ambiguity* and jeopardizes the convergence to the correct pose estimate.

In this chapter we will propose a number of solutions to deal with the aforementioned challenges, in view of achieving more accurate pose estimates. We will thus, for now, only focus on point cloud registration for use in scan matching. The solutions are mainly based on constructing a robust cost function in a way that the knowledge about the underlying surface is maximally exploited. In the following chapters, we will come up with more advanced techniques, exceeding the scope of the actual scan matching. Among these techniques are a preliminary estimation of an optimal neighbourhood function upon which a surface reconstruction is based. This reconstruction is then used to re-project the points on the surface. It is the combination of all these improvements that in the end will account for the majority of the artefacts, from reducing the noise to eliminating the outliers and filling small gaps. Finally, the registration techniques to align two non-consecutive point clouds, e.g. in case of loop closure, will be discussed in chapter 6.

3.2 Related work

A quick review of the literature reveals that there are roughly four classes of scan matching techniques that are commonly used: 1) the *iterative closest points* (ICP) algorithm, 2) the *normal distribution transform* (NDT), 3) methods based on feature points and 4) methods based on planar surfaces. However, the majority of the scan matching approaches presented in literature are based on the ICP algorithm. The algorithm is popular as it is intuitive and offers flexibility to adjust it to specific use cases. However, it also comes with a few limitations, most of them related to the aforementioned point cloud artefacts.

As it would lead us to far to give an exhaustive list of all the different scan matching techniques presented in literature, we will highlight an anthology of the most influential works. The early SLAM literature was mainly oriented towards robotic applications in which 2D laser-range sensors were the de facto standard for quite some time. Back then, the output 'maps' were in 2D, usually obtained by vertical projection on the ground plane, and the pose of the sensor in 3D containing the (x, y)-coordinate and a rotation θ. They were further heavily depending on external positioning information from IMU's or wheel odometry sensors.

3.2.1 ICP-based solutions

In [Nüchter 04], the authors presented one of the first 3D SLAM systems based on an ICP scan matching front-end and solved using *6-degrees-of-freedom*, i.e. three positional and three orientational coordinates. They used 2 tiltable SICK 2D lasers on both ends of the robot, one put horizontally and the other one vertically. As we will see later in this chapter, one of the main drawbacks of the standard ICP algorithm is the computationally expensive correspondence estimation step that - given the iterative nature of the algorithm - has to be conducted several times. The authors therefore propose a novel search procedure, based on cached *k*d-trees to accelerate this data association step. Later on they extended their system by incorporating improved loop closure mechanisms [Nüchter 05; Nüchter 07].

In [Bosse 10], the authors also used a SICK 2D laser scanner and propose another improvement for the ICP algorithm. More specifically, they discretized the workspace into a 3D grid and summarize each voxel with shape features, similar to what we will propose in chapter 4. Later on, the authors extended their SLAM system by mounting the sensor on a spring, thereby artificially extending its 'field of view' outside its standard scanning plane [Bosse 12]. They eventually showed the use of their system in the application of mapping an underground mine [Zlot 12].

More recently, the development of off-the-shelf *multi-laser* scanners, such as the Velodyne lidar scanners, caused the 3D lidar mapping literature to boom as never before. [Moosmann 11] came up with their *Velodyne SLAM* system based on the Velodyne HDL-64E scanner. They used the ICP algorithm, but to alleviate the correspondence ambiguity, they proposed to incrementally build and maintain a *'global map'* of the environment and align each newly arrived scan with the entire map. Their map is improved every time new sensor data comes in by re-estimating the surface normals based on the new points in the map. In addition, they proposed to perform a preprocessing step to *deskew* the point cloud and to cope with degeneration caused by the rolling shutter. Hereby, they assumed that the velocity is constant during the acquisition of one scan. The authors of *Velocity ICP* [Hong 10] elaborated on this problem by proposing to estimate velocity over the ICP iterations. They compensate the distortion of a scan due to motion by re-projecting all points acquired during one sweep to the *position* of the sensor at the beginning of the sweep.

The *Generalized* ICP (GICP) algorithm presented in [Segal 09] unifies the standard - widely used - cost functions in the original ICP formulation. They claim that the well-known *point-to-point* or *point-to-plane* cost functions do not take into account the fact that the point cloud is in essence a sampling of an underlying surface and that this sampling is prone to noise. Therefore, they propose a framework that is able to incorporate structural information from both the *source* and *target* point cloud in a probabilistic way. In [Holz 14], the authors improve the GICP framework by performing an approximate surface reconstruction, similar to what we will propose in chapter 5. They show the effectiveness of their system by integrating it onto a micro aerial vehicle [Holz 15].

Finally, some studies focus on combining ICP-based scan matching with (stereo) visual odometry. For example, in [Sarvrood 16] the authors propose a solution that uses the output of visual odometry to obtain a rough estimate of the ego motion upon which consecutive point clouds are registered. The plane-to-plane cost function is used to refine the motion estimation, in combination with the outputs of an IMU.

In the end, there are many design decisions concerning the ICP algorithm, regarding the cost function, point selection, outlier rejection, weighting, correspondence estimation and so on. The authors of [Pomerleau 13] devoted a whole paper to the comparison between various baseline ICP variants on real-world data sets. The experiments showed that there is a high variability among the variants in terms of success rate and accuracy, which is largely depending on the dataset. They conclude that there is need for improved ICP methods, especially for natural, unstructured and geometrical-deprived environments.

3.2.2 NDT-based solutions

Aside from ICP, the *normal distribution transform* (NDT) [Biber 03] is another common technique that is used for laser scan matching [Magnusson 07; Sun 13; Einhorn 15]. It is based on an alternative representation of a point cloud similar to an occupancy grid. In each 3D cell, a normal distribution is stored and both scans are matched by minimizing the negative log likelihood between NDTs of scans using Newton's method. Compared to ICP-based methods, the main advantage of this technique is that there is no need for explicit correspondence estimation. The main limitation on the other hand is the fact that the accuracy of NDT is strongly related to the cell size, which is difficult to define in case of inhomogeneous point clouds. Another severe drawback is that NDT quickly degenerates in environments with limited structure.

In [Magnusson 09b], the authors present an evaluation of the 3D registration reliability and speed of both the ICP and NDT algorithm. In their experiments, NDT was shown to converge from a larger range of initial pose estimates than ICP, and to perform faster. However, the poses from which NDT converged were not as predictable as for ICP. In several cases, a scan was successfully registered from a pose estimate with large initial error but failed from a pose estimate with a smaller initial error. Also, in some cases where NDT failed, the resulting pose was worse than the result of ICP, mainly because of a larger rotation error.

In a recent work [Zaganidis 17], the authors propose to introduce semantic information - extracted from the point cloud - into the registration process in order to overcome the limitations of NDT. Experiments demonstrated that their method improves the accuracy, robustness and speed of NDT, especially in unstructured environments, making it suitable for a wider range of applications.

3.2.3 Feature-based solutions

Another way to cope with the correspondence ambiguity is to introduce the use of *features* [Siritanawan 17]. Feature-based methods have significant advantages as they concentrate on strong cues such as corners and lines and filter out irregular points such as vegetation or tree leaves. However, the main drawback is that they are in general more expensive to compute, making it difficult to achieve real-time performance. One very interesting feature-based solution has been developed in the work of [Zhang 14]. In their paper, the authors propose a method using a 2-axis lidar moving in 6-DOF. Their system incorporates two processing loops, the first one being the estimation of the observer's trajectory (odometry), the other one being the actual mapping. The latter is running at a lower frequency to guarantee real-time performance. The registration technique itself is based on the extraction of only two different - and easy to extract - *shape* features, representing sharp edges and planar surface patches. Edge points of the source cloud are associated to edge lines while planar patches are matched with other planar patches in the target cloud. In [Zhang 15], the same authors extended their system by integrating visual odometry based on data from a regular camera.

3.2.4 Plane-based solutions

Finally, in [Pathak 10; Grant 13; Xiao 13], the authors use yet another approach, based on the registration of planes. These planes can lead to very accurate alignments as they serve as strong cues and can be estimated by an accumulation of data hence reducing the noise. However, as the whole registration is solely based on the alignment of the set of planes, the lack of (large) planes in the scene can cause these methods to fail. The limitations of these methods for outdoor or *'cluttered'* environments make them less generic and hence less applicable.

3.2.5 Proposed solution

In this work, we adopt the ICP-scheme as it offers a lot of flexibility. However, we integrate it within a 'hybrid' solution in the sense that we combine ICP with feature extraction and surface reconstruction. More specifically, we incorporate low-level features describing the underlying geometry in a simple way. We exploit these properties to identify for each point an ideal neighbourhood upon which the normal vectors are computed. The low-level features are further used to guide the ICP process in such a way that the alignment uses the most stable regions, i.e. planar regions instead of irregular or scattered points. Our approach benefits from large planar surfaces while at the same time remains generic.

In this chapter, we will present an extension of the generalized ICP method, by means of an improved cost function. The whole idea behind the novel cost function is that it is more suited to incorporate information from the underlying surface. The explanation and evaluation of the feature extraction will be the topic of chapter 4.

3.3 ICP

3.3.1 The fundamentals

In a nutshell, ICP iteratively determines *corresponding* point pairs between two point clouds - the *source* and the *target* - and subsequently uses these to estimate the transformation between the two that maximizes their overlap. Thereby, the target point cloud is kept fixed, while the source point cloud is transformed to best match the target. The transformation is iteratively revised by minimizing a cost function. The algorithm was proposed more or less at the same time by both [Besl 92] and [Chen 92]. The problem is formulated as follows. Given two point clouds $\mathcal{P}^s = \{\mathbf{p}_i^s\}$ and $\mathcal{P}^t = \{\mathbf{p}_i^t\}$ - denoted respectively as the *source* and *target* point cloud - where $\mathbf{p}_i^s, \mathbf{p}_i^t \in \mathbb{R}^3$ are point coordinates, find the rigid body transformation that aligns both. The transformation matrix $\mathbf{T} = (\mathbf{R}|\mathbf{t})$ consists of a rotation matrix $\mathbf{R} \in \{\mathbb{R}^{3\times3}|\mathbf{R}^\mathsf{T}\mathbf{R} = \mathbf{I}, |\mathbf{R}| = \pm1\}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$ and is estimated as follows:

$$\operatorname*{argmin}_{\mathbf{R},\mathbf{t}} E(\mathbf{R},\mathbf{t};\mathcal{P}^s,\mathcal{P}^t) = \operatorname*{argmin}_{\mathbf{R},\mathbf{t}} \sum_{i=1}^{N} d(\mathbf{R}\mathbf{p}_i^s + \mathbf{t}, \mathbf{p}_i^t), \tag{3.1}$$

where d is a distance function and N the number of corresponding points. For simplicity we assume that $\mathcal{P}^s$ and $\mathcal{P}^t$ are indexed according to their correspondences, hence we consider $\mathbf{p}_i^s \in \mathcal{P}^s$ the corresponding point of $\mathbf{p}_i^t \in \mathcal{P}^t$ for all $i \in \{1 \dots N\}$. A simple and common used cost function to estimate this transformation is the *point-to-point* criterion:

$$E(\mathbf{R},\mathbf{t};\mathcal{P}^s,\mathcal{P}^t) = \sum_{i=1}^{N} ||\mathbf{R}\mathbf{p}_i^s + \mathbf{t} - \mathbf{p}_i^t||_2^2, \tag{3.2}$$

which represents the sum of squared ℓ_2 distances between all N corresponding pairs. Other point-to-point metrics are used as well, such as the ℓ_1 norm or least median squares (LMS) variants [Li 01].

A common problem with all these variants however is that due to the ‘sampling’ process of 3D sensors - which leads to a discretization of the 3D space - perfect point-to-point matches are nearly impossible to obtain. A more robust alternative therefore is the *point-to-plane* criterion as it relaxes this constraint by allowing point offsets along the surface normal. Hence, it minimizes the distance from all points from the source point cloud $\mathcal{P}^s$ to the tangent plane of the surface in its corresponding point in the target point cloud $\mathcal{P}^t$:

$$E(\mathbf{R},\mathbf{t};\mathcal{P}^s,\mathcal{P}^t) = \sum_{i=1}^{N} ((\mathbf{R}\mathbf{p}_i^s + \mathbf{t} - \mathbf{p}_i^t) \cdot \mathbf{n}_i^t)^2. \tag{3.3}$$

In this equation, $\mathbf{n}_i^t$ denotes the normal vector corresponding to target point $\mathbf{p}_i^t$. The idea behind the point-to-plane algorithm is that a point cloud has more structure than an arbitrary set of points in 3D space; it is actually a collection of surfaces sampled by a range-measuring sensor. In chapter 4, we will elaborate on the robust estimation of surface normals, as this will be the key to successfully applying this cost function.

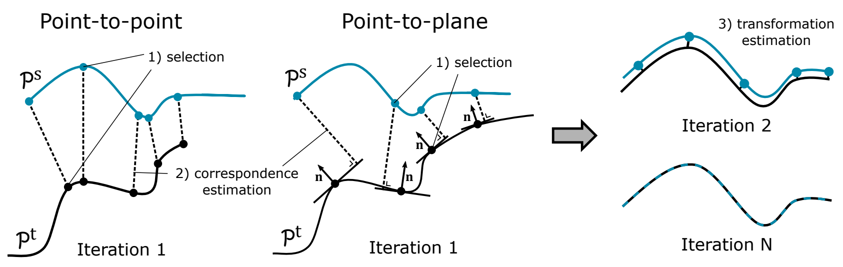


Figure 3.3: Graphical representation of the point-to-point and point-to-plane ICP algorithm. In every iteration, three main steps are executed: 1) point selection, 2) correspondence estimation and 3) transformation estimation. In every iteration, the transformation is updated until convergence has been reached and the two scans are *perfectly* aligned.

[Besl 92] proved that the ICP method terminates in a minimum as all the steps reduce the error. Some implementations define an upper threshold on the maximum distance between the two points but in that case convergence is not always guaranteed.

In Figure 3.3, a graphical representation of the ICP algorithm, for the point-to-point and point-to-plane error metric, is depicted. Each iteration of the algorithm consists of three main steps: 1) point selection (both in the source and target point clouds), 2) correspondence estimation and 3) transformation estimation. In its most simple form, all the points in the point cloud are selected and the set of closest points between the two - in Euclidean space - are taken as the corresponding pairs. The principle of the point-to-plane error metric is also demonstrated in Figure 3.3: for each point the Euclidean distance to the tangent plane of the surface in its corresponding point is minimized.

In chapter 4 we will propose a geometrical stable point selection algorithm in order to increase the robustness. Furthermore, we will propose a weighting scheme for the correspondences in order to penalize outliers. Finally, the transformation estimation involves the minimization of the cost function.

3.3.2 Closed-form solutions for solving the cost criteria

Regarding the point-to-point error metric, several closed-form solutions exist with varying accuracy, stability and speed. The four best known methods are based on singular value decomposition (SVD) [Arun 87], orthonormal matrices (ON) [Horn 88], unit quaternions (UQ) [Horn 87] and dual quaternions (DQ) [Walker 91]. As it would lead us too far to describe all of the methods in detail, we refer the reader to the respective publications. A summary of the main properties is summarized in table 3.1. The SVD method is the most commonly used one, thanks to its good

Method	Accuracy	Stability	Speed (small $\|\mathcal{P}\|$)	Speed (large $\|\mathcal{P}\|$)
SVD	best	best	good	good
OM	fair	poor	best	poor
UQ	good	good	good	fair
DQ	poor	fair	poor	best

Table 3.1: Summary of the four best known closed-form solutions to solve the point-to-point cost function in ICP, ranging from best, good, fair to poor.

performance in terms of accuracy and stability combined with acceptable speed for both smaller and larger point clouds.

The point-to-plane cost function of eq. 3.3 is essentially a least-squares optimization problem and solving it requires the determination of only the values of the six parameters $\alpha, \beta, \gamma, t_x, t_y, t_z$. However, since α, β and γ are Euler angles and thus arguments of non-linear trigonometric functions in the rotation matrix $\mathbf{R}$, efficient linear least-squares techniques can not be applied to obtain the solution. The point-to-plane error metric is therefore often solved using nonlinear least squares methods such as gradient descent, Gauss-Newton or Levenberg-Marquardt [Fitzgibbon 01]. Unfortunately, these solutions are a lot slower than their least squares counterparts.

In [Low 04] a method is therefore derived to approximate the non-linear optimization problem with a linear least squares one in case the relative orientation between the two input point clouds is small. Since we consider point clouds acquired in two consecutive scans, this assumption is guaranteed. The transformation matrix $\mathbf{T}$ can in that case be approximated as follows:

$$\mathbf{T} = \left[\begin{array}{c:c} \mathbf{R} & \mathbf{t} \\ \hdashline \mathbf{0}_3^{\mathsf{T}} & 1 \end{array}\right] \approx \begin{bmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.4}$$

In this equation α, β and γ denote the rotation in radians about the x-, y- and z-axis respectively whereas t_x, t_y, t_z are the elements of the translation vector. Equations 3.3 and 3.4 can be combined as a linear expression of the six parameters α, β, γ, t_x, t_y, t_z. Minimizing the point-to-plane cost function of eq. 3.3 then corresponds with solving a simple matrix expression:

$$\mathbf{A}\mathbf{x} - \mathbf{b}, \tag{3.5}$$

in which

$$\mathbf{x} = (\alpha, \beta, \gamma, t_x, t_y, t_z)^{\mathsf{T}}, \tag{3.6}$$

$$\mathbf{A}_{\{6\times6\}} = \sum_{i=1}^{N} \left[\begin{array}{c:c} (\mathbf{n}_i^t \times \mathbf{p}_i^s)(\mathbf{n}_i^t \times \mathbf{p}_i^s)^\mathsf{T} & (\mathbf{n}_i^t \times \mathbf{p}_i^s)(\mathbf{n}_i^t)^\mathsf{T} \\ \hdashline \mathbf{n}_i^t(\mathbf{n}_i^t \times \mathbf{p}_i^s)^\mathsf{T} & \mathbf{n}_i^t(\mathbf{n}_i^t)^\mathsf{T} \end{array} \right], \tag{3.7}$$

$$\mathbf{b}_{\{6\times1\}} = \sum_{i=1}^{N} \left[\begin{array}{c} (\mathbf{n}_i^t \times \mathbf{p}_i^s)((\mathbf{p}_i^s - \mathbf{p}_i^t)^\mathsf{T}\mathbf{n}_i^t) \\ \hdashline \mathbf{n}_i^t((\mathbf{p}_i^s - \mathbf{p}_i^t)^\mathsf{T}\mathbf{n}_i^t) \end{array} \right]. \tag{3.8}$$

For more details on the derivation of 3.7 and 3.8 we refer the reader to [Low 04]. The beauty of this solution is that it lends itself perfectly to be executed in parallel. The 6×6 linear system can be computed and summed in parallel on a GPU whereas its solution can be derived either on GPU or CPU using a SVD or a LU-decomposition. We propose to use a LU decomposition as it is computationally less complex to compute. The approach allows the transformation to be computed using *all* points in a very quick manner compared to indirect methods that often have to sub-sample the points in order to guarantee acceptable processing times. The method was implemented in the Quasar programming language [Goossens 14] and is part of our real-time camera tracker, presented in [Vlaminck 16c] and [Vlaminck 17a].

3.3.3 Correspondence estimation

Another, time-consuming, part of the ICP algorithm is the estimation of corresponding point pairs between the two clouds. In ICP, usually the set of *closest* points between the two clouds - using the Euclidean distance - are considered as the corresponding point pairs. These points will never represent the exact same physical point, cfr. the *correspondence ambiguity*, but when the orientation between the two point clouds is small and given the stability of the point-to-plane cost function, the solution will most likely converge to a good estimate of the transformation.

In a naive way, the correspondence estimation has a complexity of $\mathcal{O}(n^2)$ - n representing the number of points in one single scan - as for all points in $\mathcal{P}^s$ the distance to all points in $\mathcal{P}^t$ has to be verified. Obviously, one can do a lot better when the point cloud has some structure in it, which will always be the case when it is derived from a projective device such as a ToF camera. In those situations, the 3D point cloud is organized in a 2D image plane and hence we can exploit the adjacency of the 3D points in the 2D domain. The technique *projective data association* (PDA) exploits this knowledge. In case the point cloud does not originate from a projective device, there are still techniques to come to a faster correspondence estimation than the $\mathcal{O}(n^2)$ brute force way. The most important ones are briefly discussed in the following sections.

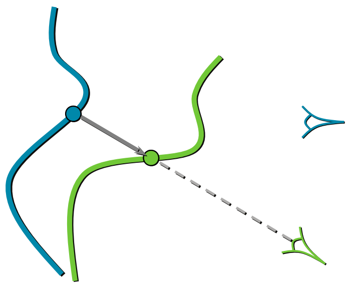


Figure 3.4: In projective data association (PDA), the points of one scan are projected in the camera view of the other (previous) scan.

Projective data association

In projective data association (PDA), the corresponding point of a point $\mathbf{p}_i^s \in \mathcal{P}^s$ is the closest point $\mathbf{p}_i^t \in \mathcal{P}^t$ to the line of projection of $\mathbf{p}_i^s$ into the camera used to obtain the points of the second cloud $\mathcal{P}^t$. Point correspondences are thus computed by projecting one point cloud onto the other with respect to the other's camera, which is depicted in Figure 3.4. A limitation of this approach however is that it introduces a limit in terms of the maximum rotation and translation between two consecutive frames that still allows reasonable data association. Furthermore, it tends to fail in areas that are 'poor' in geometry, for instance in case of long monotone corridors. Projective data association was described and evaluated in [Rusinkiewicz 01] and gained more attention after its use in the Kinectfusion system presented in [Newcombe 11].

Optical flow

In case there is a color camera integrated in the ToF device, cfr. the Kinect, one can also exploit color information to find point correspondences. This way, the limitations of PDA can be overcome and the data association is found by generating a corresponding map based on the flow field found by the Horn-Schunck or Lucas-Kanade algorithm. This strategy was proposed in [Vlaminck 16c]. It can be seen as a *'hybrid'* method, using both color and depth information in a intuitive manner.

Cylinder projection

In case of spinning lidar scanners, the points are not originated from a projective device. However, they still have structure as the point cloud is the result of a set of collinear lasers that are all spinning at the same rate. As one entire sweep of a lidar scanner comprises 360°, we can project the points on a cylinder and derive a

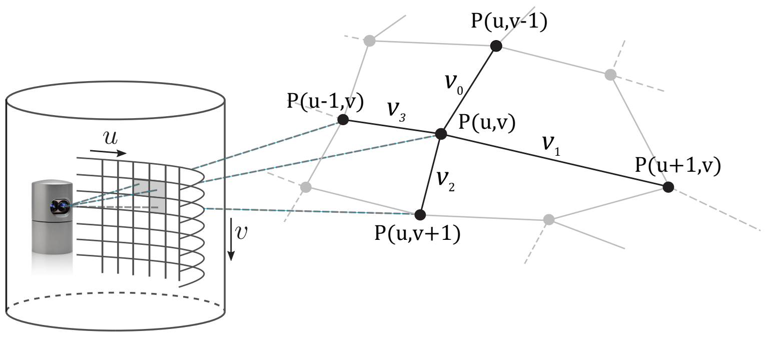


Figure 3.5: 3D points generated by a spinning lidar device are projected onto a cylinder which is subsequently quantised into cells. From these cells a range image is derived. Using this range, the adjacency in the 2D domain can be exploited to quickly find neighbouring points in 3D.

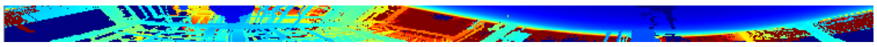


Figure 3.6: A 360° range image obtained by projecting the point cloud of figure 7.8 onto a cylinder and quantise it into cells. The blue color means that points are closeby whereas the red color denotes that points are located further away.

2-dimensional representation from it, which serves as a range image. This approach was proposed in [Vlaminck 16b]. Figure 3.6 shows an example of a 360° range image corresponding to the point cloud of Figure 7.8.

Using this 2D range image, the adjacency in the *pixel* domain can be exploited, e.g. to quickly find neighbouring points in 3D. Later on we will also exploit this knowledge to perform a surface analysis in the point cloud. One disadvantage of this projection is that the cylinder needs to be quantised in cells and the number of cells or the 'resolution' will have its effect on the *sparsity* or *completeness* of the range image. In case a low resolution is chosen, most of the cells will have at least one point projected in it. However, many cells will also contain more than one projected point and then the question arises which point should be kept. On the other hand, when a high resolution is chosen, many cells will have no points at all and as a result many occupied cells will not have any neighbours either.

K-d tree

A k-dimensional tree, also known as a k-d tree, represents a subdivision hierarchy that is generated by splitting a volume along one axis at a time, thereby changing the axis in a cyclic fashion at each subdivision step. A three-dimensional volume is

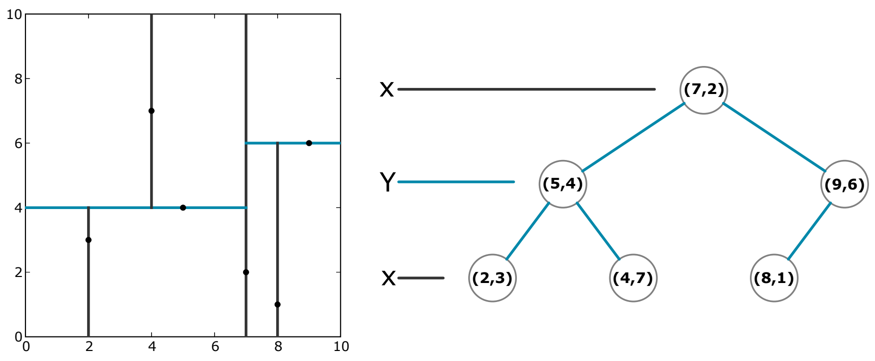


Figure 3.7: The concept of the k-d tree data structure (here in 2D). It is generated by splitting a volume along an alternating axis and one at a time. For example in 2D, first a subdivision is made based on the x-coordinate, then based on the y-coordinate and then the process restarts with the x-coordinate. Mostly, the point with the median value for the respective coordinate is chosen as parent.

commonly split first along the x-axis using a plane parallel to the yz-plane, then along the y-axis using a splitting plane parallel to the xz-plane, and finally along the z-axis using a splitting plane parallel to the xy-plane. The process continues in the next step by again splitting along the x-axis. In our discussion we will assume that the splitting planes are chosen in the x-y-z order. A k-d tree is a special case of a binary space partitioning (BSP) tree for which splitting planes can have arbitrary normal directions. Figure 3.7 shows a 2-D version of the k-d tree. Often times, the splitting is done based on the median value of the considered splitting dimension, in order to keep the tree more or less balanced and to guarantee nearest neighbour queries in $O(\log n)$ computation time.

3.4 Generalized ICP

In section 3.3.1, some cost criteria were presented that are often used in the ICP algorithm. The point-to-plane objective function has been proven more robust than its point-to-point counterpart. However, this former one does still not take into account that the source point cloud itself is also the result of a discretization of an (unknown) underlying geometric surface. Segal *et al.* [Segal 09] therefore proposed *Generalized ICP* (GICP), which is a probabilistic generalization of the ICP algorithm that allows an objective function incorporating structural information from both the source and the target point cloud. Consider again two point clouds $\mathcal{P}^s$ and $\mathcal{P}^t$ to be sampled from an underlying surface. The point clouds are as it were sampled from infinitely sampled surfaces - $\hat{\mathcal{P}}_s$ and $\hat{\mathcal{P}}_t$ - for which perfect point correspondences exist and let $\hat{\mathbf{p}}_i^s$ and $\hat{\mathbf{p}}_i^t$ be two points generated from respectively $\hat{\mathcal{P}}_s$ and $\hat{\mathcal{P}}_t$. The points $\mathbf{p}_i^s$ and $\mathbf{p}_i^t$ are now assumed to be sampled from normal distributions $\mathcal{N}(\hat{\mathbf{p}}_i^s, \mathbf{C}_i^s)$ and $\mathcal{N}(\hat{\mathbf{p}}_i^t, \mathbf{C}_i^t)$. Consider $\mathbf{T} = (\mathbf{R}, \mathbf{t})$ to be the transformation that aligns $\hat{\mathcal{P}}_s$ and $\hat{\mathcal{P}}_t$ such that $\hat{\mathbf{p}}_i^t = \mathbf{T}\hat{\mathbf{p}}_i^s$ and $\mathbf{d}_i = \mathbf{T}\hat{\mathbf{p}}_i^s - \hat{\mathbf{p}}_i^t$. Because $\hat{\mathbf{p}}_i^t$ and $\hat{\mathbf{p}}_i^s$ are drawn from independent normal distributions, $\mathbf{d}_i$ - which is a linear combination of $\hat{\mathbf{p}}_i^t$ and $\hat{\mathbf{p}}_i^s$ - is also drawn from a normal distribution:

$$\mathbf{d}_i \sim \mathcal{N}(\mathbf{T}\hat{\mathbf{p}}_i^s - \hat{\mathbf{p}}_i^t, \mathbf{C}_i^t + \mathbf{T}\mathbf{C}_i^s\mathbf{T}^\mathsf{T}) \tag{3.9}$$

$$= \mathcal{N}(\mathbf{0}, \mathbf{C}_i^t + \mathbf{T}\mathbf{C}_i^s\mathbf{T}^\mathsf{T}). \tag{3.10}$$

Using maximum likelihood estimation (MLE), the optimal transformation matrix $\hat{\mathbf{T}}$ can be determined by

$$\hat{\mathbf{T}} = \underset{\mathbf{T}}{\operatorname{argmax}} \prod_i p(\mathbf{d}_i) = \underset{\mathbf{T}}{\operatorname{argmax}} \sum_i \log(p(\mathbf{d}_i)), \tag{3.11}$$

which can be simplified to

$$\hat{\mathbf{T}} = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_i \mathbf{d}_i^\mathsf{T}(\mathbf{C}_i^t + \mathbf{T}\mathbf{C}_i^s\mathbf{T}^\mathsf{T})^{-1}\mathbf{d}_i. \tag{3.12}$$

The point-to-point ICP version can be seen as a special case of this by setting $\mathbf{C}_i^t = I$ and $\mathbf{C}_i^s = 0$ leading to

$$\hat{\mathbf{T}} = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_i \mathbf{d}_i^\mathsf{T}\mathbf{d}_i \tag{3.13}$$

$$= \underset{\mathbf{T}}{\operatorname{argmin}} \sum_i ||\mathbf{d}_i||_2^2. \tag{3.14}$$

Similarly, the point-to-plane cost function can be seen as another special case where $\mathbf{C}_i^t = \mathbf{P}_i^{-1}$, $\mathbf{P}_i$ representing the projection onto the span of the surface normal at $\mathbf{p}_i^t$ (cfr. [Segal 09]).

This *Generalized ICP* makes it possible to model the accuracy of the data in each point cloud more accurately. As an alternative for the aforementioned cost functions, Segal *et. al.* propose a different choice for the covariances C_i^s and C_i^t. They assume that the point set is locally planar and consider each point to be distributed with high covariance along its local plane, and very low covariance in the surface normal direction. In line with this reasoning they propose to set the covariance matrices as follows:

$$\mathbf{C}_i^s = \mathbf{R}_{\mathbf{n}_i^s} \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{R}_{\mathbf{n}_i^s}^\mathsf{T}, \tag{3.15}$$

$$\mathbf{C}_i^t = \mathbf{R}_{\mathbf{n}_i^t} \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{R}_{\mathbf{n}_i^t}^\mathsf{T}. \tag{3.16}$$

In these equations, the matrices $\mathbf{R}_{\mathbf{n}_i^s}$ and $\mathbf{R}_{\mathbf{n}_i^t}$ are respectively the rotation matrices that transform the basis vector $\mathbf{e}_1$ onto the normal vectors $\mathbf{n}_i^s$ and $\mathbf{n}_i^t$ corresponding to respectively $\mathbf{p}_i^s$ and $\mathbf{p}_i^t$. Doing so, ϵ is reflecting the uncertainty along the approximated normals, which should - under their assumption - be low. Its value accounts for the inaccurate measurements, or more specifically the noise of the sensor and should in general be small. For example, in case of a Velodyne scanner, the value of ϵ should be chosen somewhere between 0.01 and 0.03, approximating the accuracy of the scanner. The cost function of 3.12 with covariances defined as in eqs. 3.15 and 3.16 is often denoted as *plane-to-plane*.

In [Servos 17], the authors propose an extension of the GICP method by incorporating other channels of information such as color, intensity or any other spectral information. Like the original GICP method, MC-GICP assumes that the environment is locally planar but uses the additional channel of information to define the covariance of a point along the surface plane as well.

3.4.1 Surface-based GICP

One of the limitations of the GICP method of [Segal 09] is that its performance is highly affected by the quality of the normal estimation. However, in case of point clouds with a non-uniform point density, it is difficult to accurately estimate them, especially in sparse regions of the point cloud. Such sparse regions are often present in data originated from 3D sensors, in particular in case of lidar scanners due to the angle of incidence of the laser beams. The GICP algorithm assumes that each sampled point is distributed with high covariance along its local plane and very low covariance in the surface normal direction. It does however not take into account that there is some uncertainty about the direction of the normal vector as well. This uncertainty even increases in case of previously misaligned scans in successive registration.

In our own variant, we therefore compute the covariance of a point $\mathbf{p}_i$ in the directions perpendicular to its surface normal. Thereby, we project the neighbouring points $\mathbf{p}_j$ onto the tangent plane of $\mathbf{p}_i$ and assign a weight based on their distance to it [Vlaminck 18c]. To determine the tangent plane we compute a principal component analysis (PCA) of the covariance matrix $\mathbf{C}_i$ computed by the neighbours of $\mathbf{p}_i$:

$$\mathbf{C}_i = \frac{1}{N}\sum_{j=1}^{N}(\mathbf{p}_j - \bar{\mathbf{p}}_i)\cdot(\mathbf{p}_j - \bar{\mathbf{p}}_i)^T \tag{3.17}$$

$$\mathbf{C}_i \cdot \mathbf{v}_l = \lambda_l \cdot \mathbf{v}_l, \tag{3.18}$$

In this equation, N is the number of neighbours $\mathbf{p}_j$ of $\mathbf{p}_i$ that we take into account, λ_l the l-th eigenvalue of the covariance matrix and $\mathbf{v}_l$ the l-th eigenvector. The point

$\bar{\mathbf{p}}_i$ is the 3-dimensional centroid of the nearest neighbours. The surface normal can then be approximated by the eigenvector $\mathbf{v}_3$ corresponding to the smallest eigenvalue, as this is exactly the direction in which the 'least' variation occurs. The tangent plane of $\mathbf{p}_i$ is defined by its normal vector $\mathbf{v}_3$.

Let $\mathbf{x}_j \in \mathbb{R}^2$ be the orthogonal projections of the points $\mathbf{p}_j$ onto the tangent plane. After this transformation, the new population covariance, $\mathbf{\Sigma}_i \in \mathbb{R}^{2\times 2}$ is the diagonal matrix of the largest eigenvalues of $\mathbf{C}_i$. Now, we compute a weighted covariance using a Gaussian kernel that is defined as $\mathcal{N}(\bar{f}_i, \sigma_i)$, centered at the mean distance $\bar{f}_i$ to the tangent plane and with standard deviation σ_i for the distances. We compute the weights for each point as follows:

$$w_j = e^{-\frac{1}{2\sigma_i}(f_j - \bar{f}_i)^2}. \tag{3.19}$$

Using these weights, the distance kernel weighted centroid and covariance, $\bar{\mathbf{x}}_i$ and $\mathbf{\Lambda}_i$, can be computed as:

$$\bar{\mathbf{x}}_i = \frac{1}{\sum_j w_j} \sum_j w_j \mathbf{x}_j \tag{3.20}$$

$$\mathbf{\Lambda}_i = \frac{1}{\sum_j w_j} \sum_j w_j (\mathbf{x}_j - \bar{\mathbf{x}}_i)(\mathbf{x}_j - \bar{\mathbf{x}}_i)^\top. \tag{3.21}$$

This distribution models the uncertainty of the distance of each point to the tangent plane locally. However, it can be biased if the original sample population was itself already biased. For that reason, we compensate the potential bias by normalizing the population covariance as follows:

$$\mathbf{\Omega}_i = \mathbf{\Sigma}_i^{-\frac{1}{2}} \mathbf{\Lambda}_i \mathbf{\Sigma}_i^{-\frac{1}{2}}. \tag{3.22}$$

To use this information in the GICP framework, $\mathbf{\Omega}_i$ is used along the planar directions. Therefore, the resulting covariance used in S-GICP, cfr. eq. 3.12, is:

$$\mathbf{C}_i^s = \mathbf{R}_{\mathbf{n}_i^s} \left(\begin{array}{c:c} \epsilon & \mathbf{0}_2 \\ \hdashline \mathbf{0}_2^\top & \mathbf{\Omega}_i^s \end{array} \right) \mathbf{R}_{\mathbf{n}_i^s}^\top, \tag{3.23}$$

$$\mathbf{C}_i^t = \mathbf{R}_{\mathbf{n}_i^t} \left(\begin{array}{c:c} \epsilon & \mathbf{0}_2 \\ \hdashline \mathbf{0}_2^\top & \mathbf{\Omega}_i^t \end{array} \right) \mathbf{R}_{\mathbf{n}_i^t}^\top. \tag{3.24}$$

In these equations, $\mathbf{C}_i^t$ and $\mathbf{C}_i^s$ represent the covariance matrices for point i in the target and source point cloud respectively. When the goal is to perform registration on consecutive point clouds as part of a SLAM algorithm, we suggest to project all points on the estimated surface to cope with the point cloud artefacts depicted in

Figure 3.1. Ideally, the surface is re-estimated every time new points are added to the aggregated point cloud model.

When we consider eqs. 3.20 and 3.21 for points lying on a perfect plane, cfr. the assumption made in [Segal 09], we see that the weights w_j are reduced to 1 as both the average distance to the tangent plane $\bar{f}_i$ as well as the distance of a random neighbour point to the tangent plane f_j are 0. The covariance $\mathbf{\Lambda}_i$ will then approximate $\mathbf{\Sigma}_i$ and $\mathbf{\Omega}_i$ would degenerate to the identity matrix: $\mathbf{\Omega}_i = {\mathbf{\Sigma}_i^{-\frac{1}{2}}}^{2} \mathbf{\Sigma}_i {\mathbf{\Sigma}_i^{-\frac{1}{2}}}^{2} = \mathbf{I}_2 \in \mathbb{R}^{2\times 2}$. In summary, when the local neighbourhood of a point is planar, our algorithm degrades to the original GICP algorithm.

3.4.2 Optimizing the GICP cost function

Unfortunately, there are no closed-form solutions to solve the general cost function of eq. 3.12 in case the covariance is variable. For that reason, one has to seek refuge in non-linear minimization approaches such as conjugate gradients, Gauss-Newton or gradient descent. One popular technique is the Levenberg-Marquardt (LM) algorithm, which is a combination of Gauss-Newton and gradient descent and hence benefits from the best of both worlds. Let us collect the parameters into a vector $\mathbf{a} = [\alpha, \beta, \gamma, t_x, t_y, t_z]$ as before. We can then formulate the error metric as:

$$E(\mathbf{a}) = \sum_{i=1}^{N} d_i(\mathbf{a}) \tag{3.25}$$

where d is again the distance and i denotes the index of the corresponding point pair. LM is an iterative method that updates in each iteration the current estimate $\mathbf{a}_k$ with a term $\mathbf{x}$ so that the sum $\mathbf{a}_{k+1} + \mathbf{x}$ reduces the error $E(\mathbf{a})$. Expressed as a Taylor expansion, we can write:

$$E(\mathbf{a}+\mathbf{x}) = E(\mathbf{a}) + (\nabla E(\mathbf{a}) \cdot \mathbf{x}) + \frac{1}{2!}((\nabla^2 E(\mathbf{a}) \cdot \mathbf{x}) \cdot x) + \ldots \tag{3.26}$$

where the three dots (...) refer to the higher order terms. Let us now define the vector of residuals $\mathbf{e}(\mathbf{a}) = \{d_i(\mathbf{a})\}_i$ so that $E(\mathbf{a}) = ||\mathbf{e}(\mathbf{a})||^2$, then we can express the terms of expression 3.26 as:

$$E(\mathbf{a}) = \mathbf{e}^{\mathsf{T}}\mathbf{e} \tag{3.27}$$

$$\nabla E(\mathbf{a}) = 2(\nabla \mathbf{e})^{\mathsf{T}}\mathbf{e} \tag{3.28}$$

$$\nabla^2 E(\mathbf{a}) = 2(\nabla^2 \mathbf{e})\mathbf{e} + 2(\nabla \mathbf{e})^{\mathsf{T}}\nabla \mathbf{e}. \tag{3.29}$$

We denote the $N \times 6$ Jacobian matrix $\nabla \mathbf{e}$ by $\mathbf{J}$, with ij^{th} entry $J_{ij} = \frac{\partial d_i}{\partial a_j}$. Neglecting $(\nabla^2 \mathbf{e})\mathbf{e}$ as in the Gauss-Newton approximation, yields:

$$E(\mathbf{a}+\mathbf{x}) = \mathbf{e}^{\mathsf{T}}\mathbf{e} + \mathbf{x}^{\mathsf{T}}\mathbf{J}^{\mathsf{T}}\mathbf{e} + \mathbf{x}^{\mathsf{T}}\mathbf{J}^{\mathsf{T}}\mathbf{J}\mathbf{x}. \tag{3.30}$$

The goal now exists in finding a good update $\mathbf{x}$ such that the error $E(\mathbf{a}+\mathbf{x})$ is minimized. Using the approximation derived in eq. 3.30, differentiating with respect to $\mathbf{x}$ and equating with zero yields:

$$\nabla_{\mathbf{x}} E(\mathbf{a}+\mathbf{x}) = \mathbf{J}^{\mathsf{T}}\mathbf{e} + \mathbf{J}^{\mathsf{T}}\mathbf{J}\mathbf{x} = 0. \tag{3.31}$$

Solving this equation for $\mathbf{x}$ yields the Gauss-Newton update:

$$\mathbf{x} = -(\mathbf{J}^{\mathsf{T}}\mathbf{J})^{-1}\mathbf{J}^{\mathsf{T}}\mathbf{e}. \tag{3.32}$$

Obviously, there is no guarantee that the step will actually reduce the error at $E(\mathbf{a}_{k+1})$, but in general when the Gauss-Newton approximations are good, convergence is fast and reliable. In gradient descent approaches, the update is replaced by:

$$\mathbf{x} = -\lambda^{-1}\mathbf{J}^{\mathsf{T}}\mathbf{e} \tag{3.33}$$

where λ controls the distance travelled along the gradient direction. A small λ implies a large step in the downwards direction whereas a large λ yields a small step. In contrast to Gauss-Newton, gradient descent guarantees that the error $E(\mathbf{a}_{k+1})$ is reduced, given that λ is sufficiently large. The Levenberg-Marquardt algorithm, finally, combines both update steps such that good performance in achieved in all regions:

$$\mathbf{x} = -(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^{\mathsf{T}}\mathbf{e}. \tag{3.34}$$

A large λ now corresponds to small, safe, gradient-descent steps, whereas a small λ allows fast convergence near the minimum. The elegance of a good LM implementation is in tuning λ after each iteration to ensure rapid progress even where the Gauss-Newton approximations are poor.

3.5 Normal distribution transform (NDT)

In the related work section, we mentioned the normal distribution transform (NDT) as an alternative for ICP. As the method is regularly used in scan matching for SLAM applications, we explain a little more detail about it in this section.

The normal distribution transform (NDT) was first proposed by [Biber 03] as a method for 2D scan registration. Later on, Magnusson extended the method to use it in 3D [Magnusson 07]. The first step of the NDT algorithm consists of subdividing the 3D space into a regular grid of cells or cubes. In each cell the surface is represented by a probability density function (PDF), based on the distribution of points within the cell, as is depicted in Figure 3.8.

Each PDF can be seen as an approximation of the local surface, thereby serving as a piecewise smooth representation of the point cloud. As such, the point cloud

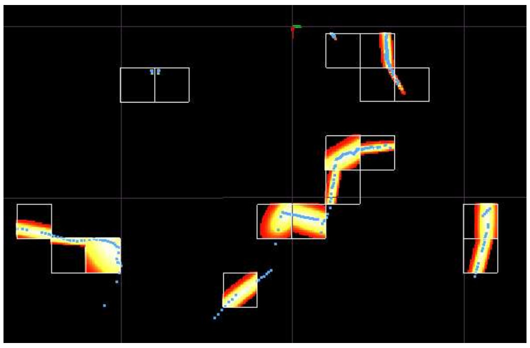


Figure 3.8: In the normal distribution transform (NDT) method [Magnusson 09a], the 3D space is subdivided into a regular grid of cells or cubes. In each cell the surface is represented by a probability density function (PDF), based on the distribution of points within it. The blue dots represent the sampled points by a laser scanner. Brighter areas denote a higher probability.

is represented in a much more compact form compared to ICP-based methods, which leads to faster scan matching. Also, it can be used as an alternative way of representing a map compared to maintaining the whole point cloud as this latter will quickly become intractable. Another important advantage of NDT over ICP is that it has a more robust data association. Instead of matching a 3D point with another 3D point, it matches a 3D point with a PDF, which is more robust.

The main limitation of NDT is that the performance of it is largely dependent on the size of the cells. However, in case of point clouds with a non-uniform point density, it is hard to pick a good cell size. In case a large cell size is picked, the accuracy will drop dramatically. On the other hand, when a small cell size is picked, many cells will contain little to no points and the method will quickly become too slow. The idea of a 3D multi-layered NDT, proposed by [Ulas 13], could mitigate this drawback.

3.6 Evaluation

In this section we will evaluate the four main scan matching techniques that were described in this chapter. For now, we apply simple versions of the ICP-based algorithms, i.e. we do not apply any outlier filtering or correspondence weighting. The algorithms involving the point-to-plane or plane-to-plane cost function need a surface normal for each point in the cloud. To estimate these, we adopt the technique of computing a principal component analysis on the covariance matrix of the set of *neighbouring* points. The neighbours themselves are determined by taking

	sequence	length (m)
1	Corridor	11.02
2	Corridor tilt	11.95
3	Lab	21.73
4	Lab tilt	22.59
5	Lab extended	52.98
6	UFO foyer	65.92
7	UFO entrance	113.72
8	Schoonmeersen	120.26

Table 3.2: The eight different indoor sequences that were captured along with their trajectory length. The first seven sequences were recorded using our Liborg platform. Two of them were recorded with a tilted scanner. The eighth sequence was recorded holding the scanner in our hand while walking around.

the k-nearest ($k = 20$) points using a k-d tree. In the next chapter we elaborate further on the topic of normal estimation in a point cloud to come to more accurate pose estimates.

We will use data captured in two different types of environments. The first group of sequences were recorded in 'man-made' indoor environments, i.e. inside buildings of Ghent University using a Velodyne VLP-16. These scenes naturally consist of large planar surfaces. For the second group of sequences, we use the KITTI dataset presented by [Geiger 12]. Those sequences were captured outdoor using a Velodyne HDL-64 mounted on a car. The car was driving in all kinds of outdoor environments, including highways, the countryside and in urban areas.

3.6.1 Indoor environments

In total we recorded eight different sequences in 'man-made' environments. They are listed in Table 3.2 along with their trajectory length in meter. The first seven sequences were captured using our Liborg platform (depicted in Figure 1.3). The eighth environment was too complex for Liborg as there were staircases present in the building. Therefore, we held the lidar scanner (VLP-16) in our hand while walking around.

To demonstrate the performance for different sensor set-ups, we have put the lidar scanner under two different tilt angles. Six sequences were recorded with a horizontally mounted scanner, whereas the two others were recorded with the scanner put under a tilt angle of approximately 45°. The latter set-up has the advantage that 'overhanging' objects, such as the ceiling, are also captured in detail and that the level of detail is more or less equal for the entire scene. The former set-up has the advantage that the FOV is larger, which is generally beneficial for the scan matching as a larger part of the scene is being captured in one sweep.

At the end of each sequence we created a loop by riding or walking back to the initial starting point. In order to be sure that we made a perfect loop, we added the first captured point cloud at the end of each sequence. Eventually, we compute the difference in position of the start and estimated end point. The distance between both ends exactly represents the error as it should ideally be zero.

Figure 3.9 depicts a top-down view of the 3D reconstructions for the different sequences, including the trajectories. The reconstructions were obtained by our final mapping algorithm, which will be described in the following chapters. The results on all of the sequences for four different scan matching techniques are listed in Table 3.3. The cross symbol '$\times$' in the Table denotes that the tracking was lost or that no meaningful trajectory could be estimated.

The method ICP uses Low's closed-form solution for the point-to-plane cost function and S-GICP is our own surface-based GICP algorithm. Prior to the scan matching we performed an additional pre-processing step by spatially filtering the point clouds, keeping only one point per cubic 10 centimeter. Point clouds captured in indoor environments contain a lot of 'redundancy' as many points are sampled on the same planar surface. Therefore, it is not necessary to use all the points to come to the best transformation estimation. The points are filtered using a regularly spaced voxel grid based on an octree. For each voxel, the centroid point of all points within the voxel is kept.

From Table 3.3, it is immediately clear that the NDT method performs very poorly. For the other methods, we notice that for almost all sequences, the point-to-plane ICP method is inferior to the plane-to-plane GICP method, except for the 'Corridor' sequence. Our own S-GICP method, however, gives the best performance.

The sequence 'UFO entrance' was captured in an environment where a lot of people were walking around. In order to eliminate the influence of these moving people in the FOV of the scanner, we devised a technique to filter them out. We rely on the fact that the vast majority of the points are sampled on static objects in the scene. Since we are using all the points, the estimation will definitely converge towards the correct transformation, except for a small distortion due to the moving objects. After applying the transformation on the *source* point cloud, we test all corresponding points whether or not they agree with this estimated transformation, i.e. whether or not their Euclidean distance after transformation is too large. If so, we can remove them from the point cloud and reprocess the entire sequence using the filtered point clouds. If we apply this procedure for the 'UFO entrance' sequence, we end up with the results listed in Table 3.3. Figure 3.10 shows an

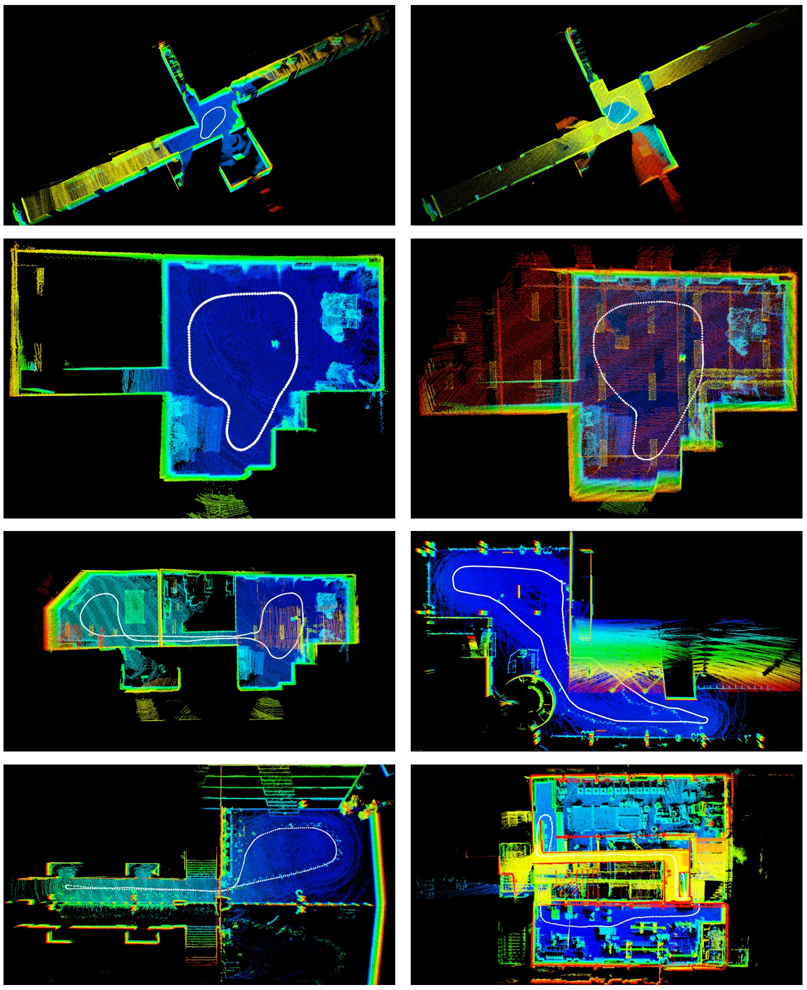


Figure 3.9: Bird-eye view of the eight 3D reconstructions including the trajectories that were followed. Seven of them where recorded using the Liborg platform. The last sequence ‘Schoonmeersen’ was recorded while the scanner was being held in order to overcome the staircases present in the environment.

	error (cm)			
Sequence	NDT	ICP	GICP	S-GICP
Corridor	197.1	**4.0**	8.1	7.5
Corridor tilt	96.2	12.2	7.3	**6.2**
Lab	199.3	19.3	7.2	**5.1**
Lab tilt	189.7	21.8	14.2	**13.0**
Lab extended	180.8	75.6	54.1	**51.7**
UFO foyer	547.0	103.6	88.3	**85.3**
UFO entrance	563.0	208.7	64.4	**31.3**
Schoonmeersen	$\times$	$\times$	243.9	**212.7**

Table 3.3: The error, i.e. the Euclidean distance between the start and estimated end position, in centimeter on the indoor sequences recorded by Liborg for four different scan matching techniques. The error is obtained by calculating the Euclidean distance between the start and end position after creating a loop. The cross symbol '$\times$' denotes that the tracking was lost or that no meaningful trajectory could be estimated.

example of this sequence with the moving objects being present and the result after filtering them out with our 'transformation'-based filtering technique.

The sequence 'Schoonmeersen' is giving poor results, which is partially due to the fact that the scanner was hand-held. Especially in the situation of climbing stairs, the movements were more abrupt compared to the smooth trajectories of the 'Liborg'-sequences. For both the NDT and the point-to-plane ICP method the tracking is lost or is drifting too far away for this sequence. The GICP and S-GICP methods are producing better results but even those methods are drifting more than 2 m on a total trajectory length of 120 m.

For our S-GICP method the relative error - defined by the error divided by the trajectory length - lies between 0.2% and 1.8% with an average of 0.8%. For the standard GICP method, the relative error lies between 0.3% and 2.0% with an average of 0.9%. It is important to note that these results are the outcome of a non-optimized algorithm that does not integrate any outlier removal or correspondence weighting. The goal of these experiments is to demonstrate the performance of the several cost functions. In the next chapters we will optimize our algorithm, which will improve the results drastically.

Apart from its higher accuracy, another main benefit of our S-GICP method over the standard GICP method is that it converges from a larger set of initial configurations. In other words, when two consecutive point clouds are heavily shifted from each other, i.e. when the robot is taking an abrupt turn, the S-GICP method is more likely to converge to the correct transformation. Figure 3.11 shows

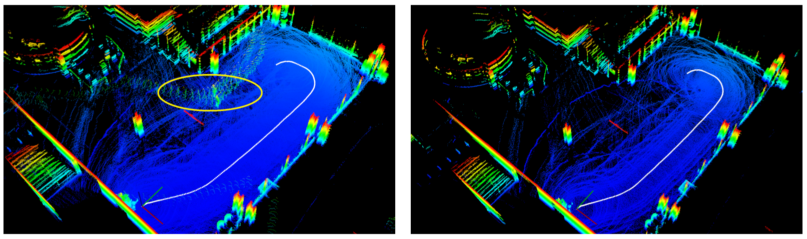


Figure 3.10: Moving objects (yellow ellipse in the left image) can perturb the transformation estimation. However, as the vast majority of the points are lying on static objects, the estimation will certainly converge towards the correct transformation. We can use this estimation to determine which corresponding points do not 'agree' with the transformation and to 'remove' these points, cfr. the right image. Eventually we can process the entire sequence again using the filtered point clouds, which increases the accuracy of the reconstruction.

an example in which the source point cloud (gray) and the target point cloud (blue) have a large difference in orientation. The results after applying all of the four methods are depicted. Once can see that only our S-GICP technique succeeds in correctly aligning the two point clouds.

As mentioned before, the NDT is also able to convergence from a larger set of initial configurations. In this particular example it manages to escape from the 'local minimum' in which the standard ICP and GICP method got stuck from the beginning. However, NDT still did not succeed to let the two point clouds entirely overlap.

3.6.2 Outdoor environments

To evaluate the performance of the different methods for outdoor environments, we use the KITTI vision benchmark [Geiger 12], as it is one of the main benchmarks regarding (lidar) odometry. The dataset was collected using a car that was driving in and around the German city Karlsruhe thereby recording data from different modalities, including a Velodyne HDL-64e lidar scanner and a stereo camera rig. The lidar scanner was put horizontallys and the data points cover a range up to 100 meter. We ran most of the algorithms on the eleven test sequences that are provided. In order to evaluate the performances quantitatively, we use the *relative pose error* (RPE) between the estimated and ground truth poses, as defined in eqs. 2.13 and 2.14. In Table 3.4, the translational errors are shown for the same 4 scan matching techniques as before. The rotational error is shown in Table 3.5. Both errors are computed over all $\Delta \in \{100\text{ m}, \ldots, 800\text{ m}\}$. The cross symbol '$\times$' once again means that no meaningful trajectory could be estimated due to too many errors.

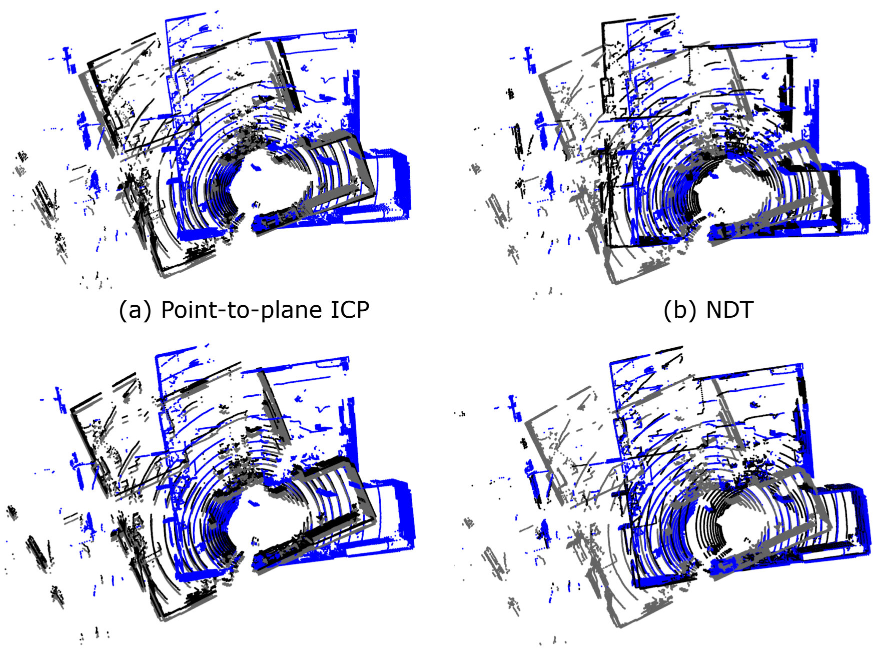


Figure 3.11: Top-down view of two point clouds acquired by a Velodyne HDL-32e in a class room. The source point cloud (gray) has to be aligned with the target point cloud (blue). Both the point-to-plane ICP, NDT and original GICP methods get stuck in a local minimum whereas our own S-GICP succeeds in aligning the two point clouds properly. One can see that the walls are perfectly overlapping.

Sequence	NDT	ICP	GICP	S-GICP
KITTI 00	2.38	1.39	1.70	1.77
KITTI 01	$\times$	3.85	29.85	27.2
KITTI 02	11.37	1.58	10.25	9.81
KITTI 03	4.17	1.07	2.98	2.28
KITTI 04	25.3	0.92	14.69	14.04
KITTI 05	3.69	1.16	1.40	1.41
KITTI 06	11.54	1.08	1.38	1.42
KITTI 07	3.34	0.51	0.93	1.04
KITTI 08	7.50	1.68	1.93	1.87
KITTI 09	13.46	1.13	3.60	3.15
KITTI 10	5.35	1.94	2.84	2.58
AVG	$\times$	1.48	6.50	6.05

Table 3.4: The translational error E_t in $\%$ on the KITTI sequences for 5 different scan matching algorithms. The cross symbol '$\times$' denotes that the tracking was lost or that no meaningful trajectory could be estimated.

Sequence	NDT	ICP	GICP	S-GICP
KITTI 00	1.39	0.94	0.74	0.74
KITTI 01	$\times$	1.02	3.92	3.75
KITTI 02	3.18	0.81	2.93	2.81
KITTI 03	1.76	0.78	0.88	0.82
KITTI 04	1.32	0.35	0.67	0.58
KITTI 05	2.21	0.81	0.59	0.58
KITTI 06	3.00	0.68	0.57	0.59
KITTI 07	2.27	0.58	0.42	0.46
KITTI 08	3.13	0.97	0.75	0.74
KITTI 09	3.55	0.86	1.11	1.00
KITTI 10	2.60	0.93	0.74	0.71
AVG	$\times$	0.79	1.21	1.16

Table 3.5: The rotational error E_r in deg/m $\times 10^2$ for the KITTI sequences using 5 different scan matching algorithms. The cross symbol '$\times$' denotes that the tracking was lost or that no meaningful trajectory could be estimated.

	NDT	GICP	**ICP**	**S-GICP**
HDL-64	4.4	1.0	0.29	1.92
VLP-16 - 10 CM^3	0.252	0.032	0.030	0.038

Table 3.6: The computation time in seconds per sweep for the indoor (recorded with a VLP-16) and outdoor (recorded with an HDL-64) sequences using 4 different scan matching algorithms. The point clouds of the indoor sequences were filtered beforehand using a regularly spaced voxel grid with a voxel size of 10 cm^3.

The main conclusions that can be drawn from this experiment is that once more, the NDT method is leading to poor results. Furthermore, we notice that the point-to-plane method is outperforming the plane-to-plane GICP and S-GICP methods as opposed to the results from the indoor dataset. This is due to the fact that in outdoor scenes, fewer large planar surfaces are present. Instead, a lot of points are sampled from ‘scattered’ objects such as trees, bushes or other kinds of vegetation. For these points it is difficult to accurately estimate surface normals, especially in regions that are sparsely sampled. Those kinds of scenes are especially present in the sequences that were recorded in rural areas or on highways that are solely surrounded by vegetation.

One of the most challenging cases is the KITTI 01 sequence as this one is entirely recorded on a highway, solely surrounded by farmland or vegetation. Moreover, a lot of points are sampled on moving cars, disturbing the estimation process even further. In the next chapter, we will devise a technique to cope with these ‘unstable’ regions. Our own S-GICP is slightly performing better than the standard GICP method, tough the difference for these kinds of environments is minimal.

3.6.3 Computational complexity

Table 3.6 gives an overview of the computation times for both the indoor and outdoor (KITTI) sequences. The experiments were conducted on a computer with an Intel Core i7-7820X CPU @ 3.60 GHz and 128GB RAM inside. The algorithms were implemented and running on a CPU using multi-threading for the computation of nearest neighbours based on a k-d tree, and for the computation of the normal vectors in case of the point-to-plane and plane-to-plane cost functions. Recall that for the indoor sequences we filtered the point clouds beforehand, using a regularly spaced voxel grid with a voxel size of 10 cm^3. For each voxel we kept the centroid of all points lying in the voxel. As expected, the closed-form solution of Low’s point-to-plane method is clearly the fastest method taking on average 290 ms for the KITTI dataset and 30 ms for the indoor sequences. The standard GICP method is still faster than our S-GICP variant. However, for the filtered point clouds, the execution times for the three ICP variants is more or less equal. This is thanks to the fact that the transformation estimation is computed using fewer points and only marginally contributes to the total time compared to the correspondence estimation step, which gains the upper hand. The NDT method is the most time-consuming method and still needs more than 250 ms to compute for the filtered point clouds.

3.7 Conclusion

In this chapter, we explained the basic concepts of point cloud registration and its use as a scan matching technique in SLAM solutions. We described and evaluated a few methods and cost functions including the normal distributions transform (NDT), point-to-plane ICP, plane-to-plane GICP and our own S-GICP. The experiments demonstrated that the best performing cost function is depending on the type of dataset.

In case of 'man-made', indoor environments, the plane-to-plane GICP method is outperforming the NDT and point-to-plane ICP method. Our own S-GICP technique is even doing better, making it the most robust method for these kinds of environments. However, the (S)-GICP algorithm has two main drawbacks. First, it is performing rather poor in environments that do not contain large planar structures, such as many outdoor scenes. Second, there is no closed-form solution and hence one has to use indirect methods, such as conjugate gradients, Gauss-Newton or gradient descent, which comes with an additional computational cost. The latter shortcoming can still be alleviated by down-sampling the point clouds, making S-GICP still the most suitable method. In case of outdoor environments, the best option is to use the point-to-plane cost function. It leads to the most accurate results and we can use the fast closed-form approximation from [Low 04] to derive the transformation.

The methods evaluated in this chapter are still simple versions and there is still much room for improvement. In the next chapter, we will present ways to focus the registration on the most stable regions of the point cloud. This is especially beneficial for outdoor scenes that are dominated by 'scattered' objects such as vegetation. Moreover, we will explain how we can weight the corresponding point pairs in order to filter out outliers. This will on its turn contribute further to the accuracy of the registration. In chapter 5, we will describe how we can maintain a map of the environment in order to relate newly acquired point clouds to this map. Together with a surface reconstruction technique, this will further improve the accuracy of the pose estimation and hence the quality of the reconstructions.

4 Geometric features for enhanced ICP

In the previous chapter we showed that incorporating structural information from the underlying surface is beneficial to enhance the point cloud alignment process. We proposed to integrate surface information in the cost function itself, giving rise to our S-GICP technique. This chapter will investigate how we can further improve the registration by adapting the other steps in the ICP algorithm, including point selection, correspondence weighting and outlier rejection.

One of the main ideas is to derive low-level geometric features from the initial raw point cloud data that can be exploited to focus on the most stable regions of the point clouds. These regions are characterized by solid objects or planar areas compared to regions with scattered objects such as trees or bushes. For the latter, it is very difficult to estimate surface normals in an accurate way. However, as most of the cost functions presented in the previous chapter are based on surface normals, an improvement of the normal estimation will have a positive influence on the quality of the registration. For that reason we propose an optimal neighbourhood selection algorithm upon which geometric features, including the surface normals, are computed.

By focusing on stable regions in the point cloud, fewer points are used to estimate the transformation upon and as a result the computational demand can be reduced. Finally, the low-level shape features will also be used to assign weights to the corresponding point pairs, which will mitigate the influence of wrong correspondences.

4.1 Introduction and related work

The ICP algorithm comes with quite a few limitations, most of them related to the point cloud artefacts described in the previous chapter. In essence, the standard ICP algorithm only works well in ideal situations, which are also depending on the specifications of the sensor itself. In this chapter, we are still focussing on data originated from spinning lidar systems such as the Velodyne or Ouster scanners. As mentioned before, one of the main challenges regarding this radial scanning is that it produces point clouds that have a non-uniform and sparse point density.

In the work of [Bosse 10] the authors tried to tackle this problem by applying the ICP algorithm to regions with strong shape characteristics, e.g. planar and cylindrical regions. To that end, they discretize the workspace into a 3D grid and examine the points that fall into each voxel to determine how planar or cylindrical it is. Voxels that contain too little points are simply discarded. In contrast to standard ICP solutions, they make voxel correspondences instead of point correspondences, where the shape parameters are incorporated to ensure that the local structure is considered. Matches between planar features are then constrained by the offset between the centroids in the direction of their surface normals, whereas matches between cylindrical features are constrained by the offsets perpendicular to the cylinder axis. To further account for the non-uniform point density, they additionally form a pyramid of grids with increasing voxel cell sizes and various offsets.

Another work worth mentioning is the *'Normal Iterative Closest Point'* (NICP) method of [Serafin 15a]. The main idea of the technique is to utilize an extended measurement vector composed by the point coordinates and its surface normal. Similar to the work of [Bosse 10], they use this augmented vector to search for correspondences. However, they extend the approach of [Bosse 10] by also incorporating the augmented vector in the computation of the alignment, i.e. in the error metric they aim to minimize.

In this PhD, we will integrate ICP within a 'hybrid' solution in the sense that we combine it with feature extraction and surface reconstruction. In this chapter we will focus on the former, i.e. the derivation of low-level shape features that describe the local geometry of the point cloud. We exploit these properties to identify for each point an ideal neighbourhood. The latter will enhance the estimation of surface normals, which will on its turn improve the transformation estimation.

The shape features are further used to guide the ICP process in a way that the alignment concentrates itself near the most stable regions, i.e. planar regions instead of irregular or scattered points. More specifically, we will propose a 'geometric stable' point selection algorithm based on the shape features as well as a robust iterative weighting scheme for point correspondences. The advantage of our proposed solution is that it benefits from large planar surfaces while at the same time it remains generic, making it more widely applicable.

Our approach has some common ground with the aforementioned ones, but there are a few major differences. The main difference with the work of [Bosse 10] and [Serafin 15b], is that they estimate correspondences by taking the nearest neighbour to each voxel or point in the 9D or 6D descriptor space instead of the 3D Euclidean space. This kind of nearest-neighbour search is however computationally demanding and jeopardizes real-time execution. In addition, when using these nearest neighbours in 9D or 6D space, the ICP algorithm turns out to be less stable and less predictable.

A second point of difference is that we do not limit the computation of the shape features to the boundaries of an artificially constructed regular-sized voxel grid as in [Bosse 10]. Instead, we carefully design a neighbourhood for each point, which is irregularly shaped. In the next section we will describe how to cluster the points into dominant planes and solid objects. This clustering will allow us to refine the neighbourhood for each point, which will eventually lead to more accurately estimated surface normals.

4.2 3D Point clustering

The clustering of points into dominant planes and solid objects is a computationally intensive task. To ease this computational burden, we rely on the organized structure of the point cloud. This structure results from the ordered acquisition by the different lasers allowing us to index the point cloud in the 2D domain after projecting it onto a cylinder as explained in chapter 3. The latter can easily be done in parallel on a GPU, by letting each CUDA thread operate on a separate point in the point cloud making it an extremely fast operation.

The second step consists of computing an *initial* estimate of the surface normals for each point, which will lead to a single normal map $\mathbf{N}$. Using the 2D projection of the point cloud $\mathcal{P}$, the normals can quickly be computed by considering two vectors that are tangential to the local surface at point $\mathbf{p}$. These vectors can be computed using the left, right, upper and lower neighbouring point in $\mathcal{P}$. The normal is then obtained by taking the cross product of the two vectors. Due to the anisotropic character, the distance between neighbouring points in the horizontal and vertical direction can be very different. The estimate will thus be rather inaccurate, but still sufficient to perform the clustering. Afterwards, the normal vectors will be updated using a more robust algorithm, which is explained in the next section.

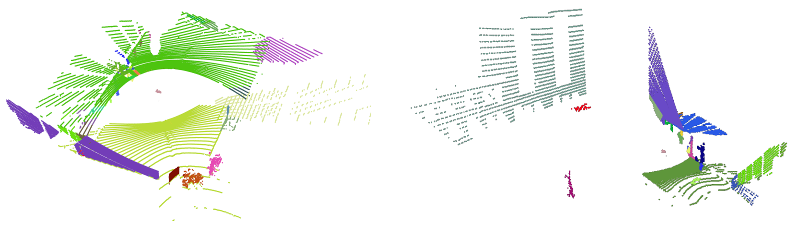


Figure 4.1: Two examples of point clouds in which the points are clustered into dominant planes and solid objects.

The third step is to conduct the actual clustering (or segmentation) of the point cloud. To ease this job, we exploit the 'Manhattan world' assumption, which states that many real world scenes consist of large planes of which many are either perpendicular or parallel to each other. For indoor scenes these planes represent the floor, ceiling and walls, but also for outdoor environments we often have the ground plane and some (large) planar objects, such as the facades of houses, present in the captured data. Motivated by this Manhattan world assumption, we first identify the large dominant planes in the scene.

To extract the dominant planes, we exploit the organized structure of the point clouds by scanning it in its 2D domain, thereby comparing neighbouring 'pixels' in a 4-connected way. Each pixel is assigned a label according to its properties compared to its neighbours. More specifically, we use two different comparison functions, the first one testing for the Euclidean distance of the corresponding 3D points, $||\mathbf{p}_i - \mathbf{p}_{i-1}||_2 < \epsilon$, the second one testing for the deviation in normal vector $\mathbf{n}_i \cdot \mathbf{n}_{i-1} < \cos\theta$. The value of the parameter ϵ is chosen dynamically, based on the distance of the points to the origin of the point cloud. Only when these two conditions are met, the two neighbouring pixels are assigned the same label. To obtain the label images we first compute a binary image (i.e. mask), which we then feed to our GPU-based connected component labeling (CCL) algorithm.

Once the dominant planes in the scene are segmented, the remaining objects are easily distinguishable. It is easy to segment these objects by clustering the points based on their relative distance. We therefore conduct a second loop of CCL, but this time only using the condition on the Euclidean distance. Figure 4.1 depicts two examples of point clouds in which the points are clustered into dominant planes and solid objects.

4.3 Normal estimation

In the previous section, we explained how to obtain an initial guess on the normal vectors by taking the cross product of two vectors that are tangential to the local surface. This estimate is rather poor and since the registration is highly affected by the normals, we propose a better way to compute them. Our method is based on a principal component analysis (PCA) of a points neighbourhood.

Recall that in the experiments of the previous chapter, we used a fixed number of neighbours. However, in case of an inhomogeneous point density, a fixed number of neighbours is not appropriate. In sparsely sampled areas, a high number of points will cause the 'neighbouring' points to be far away from the point under consideration. Some of the neighbours could be lying on entirely different surfaces. A 'low' number of points on the other hand might be too inaccurate due to imprecise measurements and noise.

Also, in case of discontinuities, a fixed number of neighbour points can cause surface normals near the intersection of two planes to point in the wrong direction, as indicated by the green line in drawing (a) of Figure 4.2. In some cases this would not cause a problem for registration, for example when the incidence angle of the laser beams make the intersection of the planes visible in the two point clouds that need to be aligned. However, when the intersection is not visible in one point cloud, i.e. only one of the planes is visible, the neighbours will all be selected on the plane causing the normals to point in the 'correct' direction, cfr. image (b) in Figure 4.2. This means that for the same *physical* point, e.g. the yellow corner point, the normal vectors in the two point clouds could be estimated differently. This could on its turn jeopardize the correct alignment as these correct correspondences would be penalized.

An alternative way to estimate surface normals is to use only points within a specified distance, rather than a fixed number of points, e.g. by selecting neighbours when they are within 5 cm from the point under consideration. This approach often leads to even poorer estimates, especially for point clouds with non-uniform point densities such as the ones acquired with spinning lidar scanners. In those situations, it can happen that all the neighbours within the search radius are selected along the same scan line, as is depicted by the light blue dots in Figure 4.3. The green points from the two 'neighbouring' scan lines however provide a lot more useful information about the surface so they should be considered as well.

In order to overcome the limitations of the two aforementioned approaches, we propose to estimate the 'underlying topology' of the points. The latter is defined as the estimation of the neighbourhood relationship among the points in a point cloud and it will allow a better estimate on the surface normals. Our technique

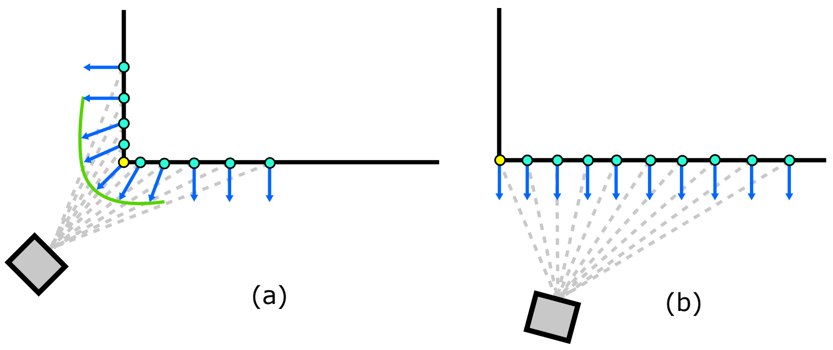


Figure 4.2: The estimation of normal vectors using the k-nearest neighbour criterion. In the left image, around the intersection of the two planes (here drawn in 2D), normal vectors are pointing in the wrong direction (cfr. the green line). In the right image they do not suffer from this artefact because points on the vertical plane are not sampled (due to occlusion) and are hence not used to compute the normal vector of the blue corner point. This results in a differently estimated surface normal for the yellow point. In case of registration of the two point clouds, the (correct) matching of the corner point might be penalized by its discrepancy in normal vector, which should be avoided.

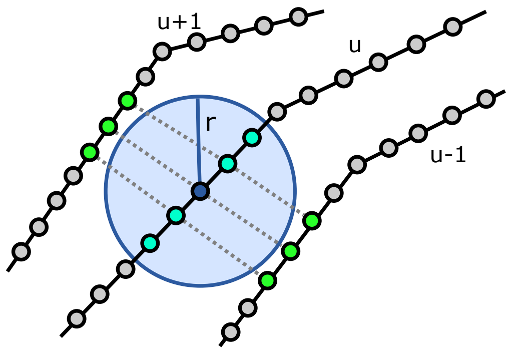


Figure 4.3: Normal estimation based on a radius has the risk that only points along the scan line are taken into account. Instead, it is better to take the topological neighbourhood into account.

comprises two main steps. First, we conduct a clustering of the points based on their geometrical properties, i.e. guided by the Euclidean distance between the points as well as an initial guess of their normal vectors. Second, based on the clustering we construct an optimal neighbourhood for each point that we use to re-estimate its surface normal. Doing so, we solve for the discontinuities in the point cloud, such as the one depicted in Figure 4.2. For the yellow corner point, this means that only points of either one of the planes will be taken into account to estimate its true surface normal. Note that in reality it is very unlikely that exactly the intersection of the two planes is sampled. A point like the blue one can thus be assumed to always belong to either one of the planes.

Similar to section 3.4.1, the derivation of the surface normal of a point $\mathbf{p}_i$ is conducted using a PCA on the covariance matrix $\mathbf{C}_i$ of the neighbours $j \in \{1, \ldots, N\}$ of that point as given by equation 4.2:

$$\mathbf{C}_i = \frac{1}{N} \sum_{j=1}^{N} (\mathbf{p}_j - \bar{\mathbf{p}}_i) \cdot (\mathbf{p}_j - \bar{\mathbf{p}}_i)^T \tag{4.1}$$

$$\mathbf{C}_i \cdot \mathbf{v}_l = \lambda_l \cdot \mathbf{v}_l, \quad l \in 1, 2, 3 \tag{4.2}$$

In this equation, N is the number of neighbours of $\mathbf{p}_i$ that we take into account, λ_l the l-th eigenvalue of the covariance matrix and $\mathbf{v}_l$ the l-th eigenvector. The point $\bar{\mathbf{p}}_i$ is the 3-dimensional centroid of the nearest neighbours. The surface normal can then be approximated by the eigenvector $\mathbf{v}_3$ corresponding to the smallest eigenvalue, as this is exactly the direction in which the 'least' variation occurs. One could opt to add an additional weight to each neighbour in order to give less influence to points located further away from the point under consideration.

4.4 Optimal neighbourhood selection

As mentioned in the previous section, an easy neighbourhood function is often chosen, such as the k-nearest points or using a fixed radius. However, these overly simple functions cause the estimation of normal vectors to be inaccurate as points can be separated by an object border or can be part of entirely different surfaces. When used in point cloud registration, it jeopardizes the correct alignment of the two point clouds and hence the accurate estimation of the sensor pose. To deal with this, we exploit the clustering described in the previous section to select only points that are lying in the same cluster. We will now further enhance the neighbourhood selection by selecting only those points that minimize the *'Eigen Entropy'* and hence the 'disorder' in the set of neighbours.

The main question is thus how to find for each 3D point the optimal neighbourhood size or the most suitable local point set that describes the underlying geometry.

This problem can be considered as an interdependence problem as geometrical features largely depend on the choice of the neighbourhood, whereas a good neighbourhood definition should rely on the local geometry, and thus on geometrical features.

Let us first define a *neighbourhood* function n that maps a point to a set of neighbouring points: $n^r(\mathbf{p}_i) : (x, y, z) \mapsto (x, y, z)^N$, in which r denotes the search radius for neighbouring points and N denotes the number of points belonging to the neighbourhood. Recall that we first clustered the points and that the 'radius search' only selects neighbouring points that are part of the same cluster. Using this neighbourhood we determine the principal components of the covariance matrix of the points belonging to it, as explained in the previous section. Thus, given a set of neighbours of a point $\mathbf{p}_i$ determined by a radius r, we first compute the eigenvalues λ_1, λ_2 and λ_3 corresponding to the eigenvectors $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$. Now, we consider the three values

$$\psi_1 = \frac{\lambda_1 - \lambda_2}{\lambda_1}, \psi_2 = \frac{\lambda_2 - \lambda_3}{\lambda_1}, \psi_3 = \frac{\lambda_3}{\lambda_1}, \tag{4.3}$$

that respectively represent how elongated, planar or scattered the underlying surface is. We further define the *dimensionality* label as

$$l = \underset{i \in [1,3]}{\operatorname{argmax}}(\psi_i). \tag{4.4}$$

When $\lambda_1 \gg \lambda_2, \lambda_3$, then ψ_1 will be larger than the two other features. This corresponds to lines between planar surfaces or thin structures such as pipelines. On the other hand, when $\lambda_1, \lambda_2 \gg \lambda_3$, then ψ_2 will be larger, corresponding to planar surfaces. Finally, when $\lambda_1 \approx \lambda_2 \approx \lambda_3$, then ψ_3 will be larger, which is the case for *scattered* points, for example points that are part of bushes or tree leaves. Thus, points lying on the intersection of two planes are assigned the label '1', points lying on planar surfaces are assigned the label '2' and finally points belonging to volumes or scatter are assigned the label '3'. As the three values ψ_1, ψ_2 and ψ_3 represent a partition of unity Ψ, we can compute its 'entropy', which is given by:

$$e_{\Psi^r} = -\psi_1 \ln(\psi_1) - \psi_2 \ln(\psi_2) - \psi_3 \ln(\psi_3). \tag{4.5}$$

This entropy, as a function of r, gives a notion of how certain we are that a point is really belonging to one of the three classes. The optimal neighbourhood radius r^* is then defined as the minimum of the entropy function e:

$$r^* = \underset{r \in [r_{min}, r_{max}]}{\operatorname{argmin}} \; e_{\Psi^r}, \tag{4.6}$$

in which r_{min} and r_{max} are thresholds on respectively the minimum and maximum radius. In other words, by minimizing the above function, we favour the minimal disorder of the neighbourhood set. The radius r^* leads to an initial guess of the optimal neighbourhood $\mathcal{N}_i$ of a point $\mathbf{p}_i$. As points within the optimal radius r^* can still have different dimensionality labels, we additionally define a similarity measure S denoting the homogeneity within the neighbourhood of each point:

$$S(n^r(\mathbf{p}_i)) = \frac{1}{N} \sum_{\mathbf{p}_k \in n^r(\mathbf{p}_i)} \mathbf{1}_{l(\mathbf{p}_i)=l(\mathbf{p}_k)}. \tag{4.7}$$

In this equation $\mathbf{1}$ represents the indicator function and N is the cardinality of $n^r(\mathbf{p}_i)$. If less than half of the points in the neighbourhood set $n^r(\mathbf{p}_i)$ have the same dimensionality label, i.e. S is smaller than 0.5, we recompute the optimal radius r^* using a different criterion:

$$r^* = \underset{r \in [r_{min}, r_{max}]}{\operatorname{argmax}} \; S(n^r(\mathbf{p}_i)). \tag{4.8}$$

In the other case when S is higher than 0.5, we remove the points that have another dimensionality label from the neighbourhood of $\mathbf{p}_i$. Hence, we define the optimal neighbourhood $\mathcal{N}_i$ of a point $\mathbf{p}_i$ as:

$$\mathcal{N}_i = \{\mathbf{p}_k : \mathbf{p}_k \in n^{r^*}(\mathbf{p}_i), l(\mathbf{p}_i) = l(\mathbf{p}_k)\}. \tag{4.9}$$

Using this optimal neighbourhood set $\mathcal{N}_i$ we recompute for all points $\mathbf{p}_i$ the feature vector $\mathbf{x}_i$: the eigen values $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \lambda_3\}$, the eigen vectors $\mathbf{v} = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$, the dimensionality values $\Psi = \{\psi_1, \psi_2, \psi_3\}$, the dimensionality label l and the values r^* and E^*. On top of that, we define the *omnivariance* as $O = \sqrt[3]{\prod_{i \in [1,3]} \sigma_i}$, the anisotropy as $\frac{\lambda_1 - \lambda_3}{\lambda_1}$ and the local surface variation as $c_{\mathbf{p}} = \frac{\lambda_3}{\sum_i \lambda_i}$.

Eventually, we can aggregate the (sampled) point $\mathbf{p}_i \in \mathcal{P}$, together with its feature vector $\mathbf{x}_i$, and the set of neighbours $\mathbf{N}_i$ into a *surfel* data structure. The word *surfel* is a concatenation of 'surface' and 'element' and can be seen as a building block (element) to represent a surface patch. We denote the surfel of a point $\mathbf{p}_i$ by $\mathcal{S}_i = \{\mathbf{p}_i, \mathbf{x}_i, \mathbf{N}_i\}$. In the following sections, we will explain how this representation can be used to optimize the ICP process to obtain both, faster convergence as well as more accurate pose estimates.

4.5 Optimizing the ICP algorithm

In section 3.3.1, we thoroughly described the fundamentals of the ICP algorithm. We also presented a few cost functions to derive the transformation between two scans or point clouds, including the point-to-plane and plane-to-plane cost function.

In order to apply these cost functions, we need an accurate estimate of the surface normals $\mathbf{n}_i$ for all points $\mathbf{p}_i$. As explained in the previous section, the local surface normal of a point can be approximated by the eigenvector $\mathbf{v}_3$ corresponding to the smallest eigenvalue λ_3 of the covariance matrix constructed using its neighbours, cfr. eq. 4.2. By applying our optimal neighbourhood selection (cfr. eq. 4.9), the principal component analysis is conducted using the 'ideal' neighbourhood set, which tremendously improves the estimation of surface normals.

The rest of the features will not be used directly in the cost function. Nevertheless, they are still valuable in further enhancing the ICP process. In the next sections, we will propose two different strategies to incorporate the features. The first strategy is to exploit the features in order to select stable points, i.e. points that are lying on solid surfaces rather than points that are sampled in unstructured areas such as bushes or grass. A second way of incorporating the features is to use them for correspondence weighting and rejection. When the features are very similar, the corresponding pair will be assigned a higher weight than for the case in which they are rather asimilar. In case the features are simply too different the correspondence will be considered as wrong and we will reject it.

4.5.1 Point selection

Instead of using all the points in the point cloud to estimate the transformation, it is more appropriate to select only a subset of points. Apart from faster computation times, the selection of good candidate points can improve the pose estimation tremendously. Obviously, we should pick *reliable* points that are part of stable areas and for which accurate surface normals can be estimated. These areas are generally characterized by solid surfaces, often times planar in case of man-made environments, such as the floor, the ceiling, the walls, cupboards, desks, etc. Also in outdoor (urban) environments, objects with high planarity are often present, e.g. facades of houses or parked cars along the road.

Points that are part of under-sampled or unstructured areas on the other hand can better be discarded as they might jeopardize accurate pose estimation. These *scattered* areas are often induced by natural phenomena or vegetation, such as tree leaves, bushes or grassland. For points sampled in such areas it is very hard to compute reliable surface normals that are consistent among consecutive scans. One way to deal with this problem is to simply discard these points from the estimation process.

However, we should make sure that - at all times - the selected points provide enough 'observability' of the translation and orientation components along all of the three axis x, y and z. This observability denotes the extent to which the sampled points are able to explain the translation and orientation of the sensor. For example, ground points provide observability of the translation along the y-axis. Facades on the other hand give observability of the translation along the x- and z-axis.

We propose two improvements over the original ICP method. First, we *normalize* the rotational and translational observability, to make sure that the transformation is not biased towards one particular direction. Second, we incorporate the features to add more importance to planar areas.

4.5.1.1 Normalizing rotational and translational observability

Imagine the scenario in which a car is driving along a road and where the majority of the points are captured on the road surface. Suppose that the scanner is mounted vertically and that the normal vectors of the ground points are oriented in the direction of the y-axis. In that case, the ground points only provide observability of the translation in the direction of this y-axis. The points sampled on the ground plane will always result in the same pattern making it impossible to estimate the translation in the directions parallel to the x- and z-axis. We should thus make sure that we sample an equal amount of points on structures that are perpendicular to the ground plane as well, otherwise the estimation will be biased towards the identity transformation. Furthermore, we should also favour points that are located further away from the scanner, as they are stronger, more stable, cues to compute the rotational parameters upon.

In order to fulfil these two constraints, we start by defining the following six *'observability'* values for each point $\mathbf{p}_i$ in the point cloud: $|\mathbf{n}_i \cdot \mathbf{x}|$, $|\mathbf{n}_i \cdot \mathbf{y}|$, $|\mathbf{n}_i \cdot \mathbf{z}|$, $(\mathbf{p}_i \times \mathbf{n}_i) \cdot \mathbf{x}, (\mathbf{p}_i \times \mathbf{n}_i) \cdot \mathbf{y}, (\mathbf{p}_i \times \mathbf{n}_i) \cdot \mathbf{z}$. In all these definitions, $\mathbf{x} = (1, 0, 0)$, $\mathbf{y} = (0, 1, 0), \mathbf{z} = (0, 0, 1)$ and $\mathbf{n}_i$ represents the normal vector of the point $\mathbf{p}_i$. The former three values give the contribution of a point $\mathbf{p}_i$ to the observability of the three unknown components of the translation vector. The latter three values on the other hand give the contribution of a point $\mathbf{p}_i$ to the observability of the different unknown angles (pitch, yaw and roll) of the vehicle. Note that points located further away from the sensor center contribute more.

The idea is to sample points equally distributed among those that score high for each of the six observability values. To this end we sort the points based on all of the six values, generating six ordered lists. For every sorted list we eventually select the N highest-valued points. Note that this implies that fewer than $6N$ points can be selected in total, as the same point can be part of multiple lists. The exact value of N is determined through experimental evaluation and is discussed in section 4.6.

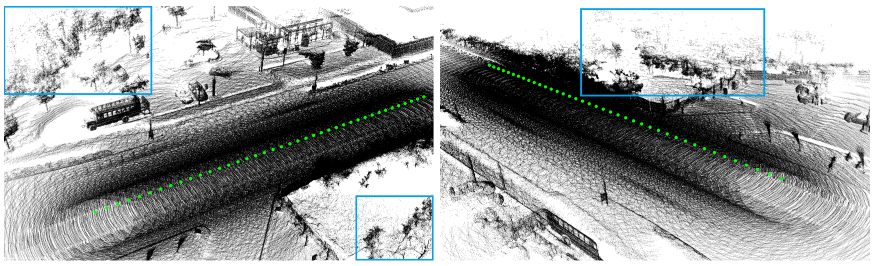


Figure 4.4: The start of the KITTI04 sequence, in which a car is driving along a straight line in a street with almost no buildings. In the beginning, there is only a bit of vegetation present aside the road, causing difficulties in aligning consecutive scans.

4.5.1.2 Feature-based selection

Apart from maximizing the rotational and translational observability, it is also beneficial to take the features into account to determine which points to select, in order to select those that are lying in *stable* regions. One way of doing this, is to select points that have a dimensionality label equalling ‘2’, denoting that they are part of ‘planar’ regions. However, not all the points with label ‘2’ are lying on strong solid objects. To make the point selection more robust, we can opt to conduct an additional selection based on the entropy values e_Ψ. When the uncertainty about the dimensionality label of a point is high, this will reflect in a high entropy and should better be discarded.

Figure 4.4 depicts a scene from the KITTI04 sequence, where a car is driving along a straight line in a street with almost no buildings aside it. Instead, some trees and bushes are present along the street (indicated by the blue rectangles) making it difficult to accurately estimate surface normals for these regions. This on its turn makes it difficult to robustly match them in consecutive point clouds, which can cause the ICP algorithm to get stuck in a local minimum.

One solution is to remove all the points lying on those bushes and trees, based on the ‘dimensionality’ or ‘planarity’ feature defined in the previous section. A problem that pops up in that case is that we end up with points mainly lying on the ground plane. As explained in the previous section, the ground plane only provides observability of the translation along the y-axis. To get observability of the translation in the x-axis and z-axis we should keep at least some points sampled on objects that are perpendicular to the ground plane. Ideally, these objects are solid such as facades of houses, but sometimes there are not many of these objects present in the scene. If the number of points lying on the ground plane is too dominant compared to the ones lying on objects perpendicular to the ground plane, we will run into problems. In that case, ICP will converge to the unit transformation as the

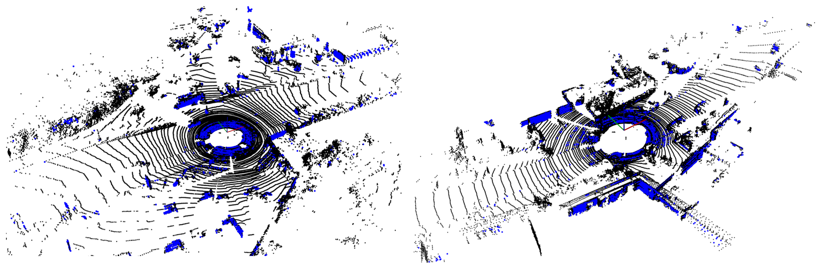


Figure 4.5: Two examples demonstrating our geometrical stable sampling technique. The blue points represent the ones that have the highest planarity and cover the highest 'observability' in each of the main directions.

way the points are sampled on the ground is the same among consecutive scans (under the assumption that the vehicle is driving on a flat road).

We thus need an equal part of the points being sampled on the ground plane as well as on other objects perpendicular to the ground plane. One way to deal with this, is to extend the strategy described in the previous section. There, we sorted the points based on their contribution to the observability comprising the translational and rotational components. We can now scale the values with the *planarity* feature defined in section 4.4, in order to give a higher weight to those points that are sampled on robust planar regions. We propose to update the values as follows: $\psi_2|\mathbf{n}_i \cdot \mathbf{x}|$, $\psi_2|\mathbf{n}_i \cdot \mathbf{y}|$, $\psi_2|\mathbf{n}_i \cdot \mathbf{z}|$, $\psi_2(\mathbf{p}_i \times \mathbf{n}_i) \cdot \mathbf{x}$, $\psi_2(\mathbf{p}_i \times \mathbf{n}_i) \cdot \mathbf{y}$, $\psi_2(\mathbf{p}_i \times \mathbf{n}_i) \cdot \mathbf{z}$, in which ψ_2 denotes the planarity of point $\mathbf{p}_i$. Like in the previous section, we will then select from each list the N points with the highest value. In Figure 4.5, our geometrical stable sampling technique is depicted in two different scenes. The blue points represent the points that have the highest planarity and cover the highest 'observability' in each of the main directions.

4.5.2 Correspondence weighting and rejection

Selecting only a subset of points from each point cloud is one way of increasing the robustness of the alignment. Another way is to cope with wrong corresponding pairs. The 'standard' ICP algorithm treats all corresponding pairs equal and does not cope with these outliers. A common approach is to only use a subset of pairs, estimate the transformation using this subset and finally check if the other correspondences *agree* on it. This technique is often referred to as *random sample consensus* (RANSAC), as it usually involves a random sub-sampling. The transformation with the largest *consensus set* is considered the correct one.

The main drawback of this RANSAC method is its non-deterministic behaviour and the fact that it is computationally demanding. In this PhD we therefore opt to weight the corresponding pairs in order to eliminate the influence of outliers. As the corresponding points are re-determined in each iteration of the ICP algorithm, the weights are also recomputed, hence the name *iteratively re-weighted least squares* (IRLS) ICP.

4.5.2.1 Iteratively reweighted least squares (IRLS) ICP

A graceful way of dealing with outlier correspondences is to incorporate robust M-estimators in what is known as *robust regression*. In that case, the cost function of 3.1 is adapted to:

$$E(\mathbf{T}; \mathcal{P}^s, \mathcal{P}^t) = \sum_{i=1}^{N} w_i d(\mathbf{R}\mathbf{p}_i^s + \mathbf{t}, \mathbf{p}_i^t) \tag{4.10}$$

where d is again a distance function, either the point-to-point, point-to-plane or plane-to-plane distance, presented in eqs. 3.2, 3.3 and 3.12 and w_i a weight assigned to the i-th corresponding pair. Three common robust weight functions are Huber's function, Cauchy's function and Beaton Tukey's bi-weight function, respectively defined as follows:

$$w_{Hu}(r) = \begin{cases} 1, & \text{if} |r| \leq c \\ \frac{c}{|r|}, & \text{if} |r| > c \end{cases} \tag{4.11}$$

$$w_{Ca}(r) = \frac{1}{1 + \left(\frac{r}{c}\right)^2} \tag{4.12}$$

$$w_{Tu}(r) = \begin{cases} (1 - \frac{r^2}{c^2})^2, & \text{if} |r| \leq c \\ 0, & \text{if} |r| > c \end{cases} \tag{4.13}$$

in which c represents a tuning constant. In the standard ICP formulation, the value r represents the Euclidean distance between the two corresponding points. However, in case of lidar scanning, the distance between corresponding points from two rotated point clouds, will be larger in case they are located further away from the origin. Obviously, the corresponding pair should not be penalized when it represents the same physical point in space. For that reason, we will not weight the correspondence points based on their Euclidean distance, but rather use their features derived in the previous section. Note that the feature distance between the same correspondences in consecutive iterations will not change, as the features are fixed and independent of the orientation of the point cloud. However, the correspondences themselves will change in every iteration and therefore the weights are re-calculated.

Each of the three robust M-estimators, plotted in Figure 4.6, have different properties regarding handling outliers. Tukey's bi-weight function has the highest

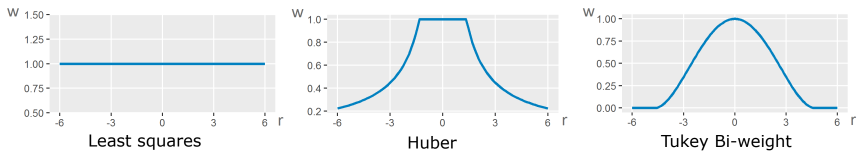


Figure 4.6: Two examples of common robust M-estimators compared to ordinary least squares (left): Huber's weight function (middle) and Tukey's bi-weight function (right).

Algorithm 1 The IRLS-ICP algorithm

```
Require: $\mathcal{P}_s$, $\mathcal{P}_t$, weight function $w$, distance functions $d_1$ and $d_2$
1: $\mathbf{R}^{(0)} = I$, $\mathbf{t}^{(0)} = \mathbf{0}_3$
2: $k = 0$, $\mathbf{p}_i^{s,0} = \mathbf{R}^{(0)}\mathbf{p}_i^s + \mathbf{t}^{(0)}$
3: repeat
4:     $k = k + 1$
5:     determine closest points
6:     $w_i = w(d_1(\mathbf{p}_i^{s,k-1}, \mathbf{p}_i^{t,k-1})), \forall i \in \{1 \dots N\}$
7:     $[\mathbf{R}, \mathbf{t}] = \underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i d_2(\mathbf{R}\mathbf{p}_i^{s,k-1} + \mathbf{t}, \mathbf{p}_i^{t,k-1})$
8:     $\mathbf{p}_i^{s,k} = \mathbf{R}\mathbf{p}_i^{s,k-1} + \mathbf{t}$
9:     $\mathbf{R}^k = \mathbf{R}\mathbf{R}^{k-1}$
10:    $\mathbf{t}^k = \mathbf{R}\mathbf{t}^{k-1} + \mathbf{t}$
11: until convergence
```

protection against outliers, excluding them entirely from the minimization process. Cauchy's function does not disregard any correspondences but still protects strongly against outliers. Huber's function finally is the least strict. The IRLS-ICP method is outlined in Algorithm 1. Note that we use two different distance functions d_1 and d_2, the former one to compute the weights, the second one to estimate the transformation.

A special case of this IRLS-ICP algorithm integrates *probabilistic data association*. It was first proposed by [Agamennoni 16] as a technique to deal with cases where one point cloud is denser than the other, but it can also be applied for point clouds with similar point densities. In their method, each point in the source point cloud (the sparse one) is associated with a set of points in the larger target point cloud. Each association is then weighted so that the weights form a probability distribution. The authors propose to use a t-distribution, which serves as a heavy-tailed distribution with parameter v to control the weight of the tails. For finite v it assigns a non-negligible probability to the tails, thereby implicitly taking outliers into account without the need to pre-filter them or to treat them as a special case. The weights are determined through expectation-maximization (EM): the E-step effectively estimates the values for the weights, while the M-step updates the parameters in order to decrease the expected negative log-likelihood.

4.5.2.2 Feature-based weighting

As mentioned in the previous section, we will incorporate the shape features to assign a higher weight to corresponding points $\mathbf{p}_i^s$ and $\mathbf{p}_i^t$ whose local neighbourhoods are more similar. Thus, based on the feature representations of $\mathbf{p}_i^s$ and $\mathbf{p}_i^t$, we define a similarity measure expressing how likely it is that the two matched points are truly corresponding to the same physical point. The more similar the local neighbourhoods of the two corresponding points are, the higher the weight we assign to it. This will not only lead to more accurate pose estimates but will also help ICP to converge a lot faster to the minimum of the cost function.

There are many ways of combining the features to compute a weight for each corresponding pair. One way is to select a subset of the feature vector $\mathbf{x_p}$ and to use all the features in this vector jointly. However, different features constitute different quantities, units and range. To account for this issue, a normalization, i.e. min-max scaling, across all feature vectors is carried out. On the other hand, some features are more expressive than others and hence they should get a higher influence in the total distance computation. In addition, some features contain outlier values, causing the min-max scaling to map the majority of the values close to zero. To deal with this, we define a coefficient vector $\boldsymbol{\beta}$ that assigns a weight for every feature in the vector $\mathbf{x_p}$.

We eventually apply the Beaton Tukey's bi-weight robust M-estimator as it provides the strongest protection against poor corresponding pairs. In fact, those outlier correspondences are entirely excluded in the minimization. The weight function is thus given by eq. 4.13, where $r = ||\boldsymbol{\beta}^\mathsf{T}(\mathbf{x}_{\mathbf{p}_i^s} - \mathbf{x}_{\mathbf{p}_i^t})||_2$ is the *'feature distance'* between two corresponding points $\mathbf{p}_i^s$ and $\mathbf{p}_i^t$ and $c = 13\sigma$ the tuning constant with $\sigma = \text{median}(\{d_i\}, i = 1 \dots N)$. The choice for a rather high tuning constant is motivated by the fact that true outlier matches are largely deviating from the median. We thus want to keep as much points as possible to estimate the transformation while still providing protection against severe outliers. The use of the median in the estimation of σ provides less vulnerability to outliers than if the scaling were based on the actual upper distances.

4.6 Evaluation

In order to evaluate the suggested improvements, we will again use the KITTI dataset [Geiger 12]. This time, we will conduct a comprehensive analysis for the different sequences in that dataset. Recall that the data was captured using a

Sequence	Distance (m)	Environment type
0	3714	Urban
1	4268	Highway
2	5075	Urban + Country
3	563	Country
4	397	Country
5	2223	Urban
6	1239	Urban
7	695	Urban
8	3225	Urban + Country
9	1717	Urban + Country
10	919	Urban + Country

Table 4.1: The trajectory length and type of environment for the KITTI sequences.

Velodyne HDL64-E lidar scanner mounted horizontally on top of a car. The data was recorded in a variety of environments, including urban areas, countryside areas and highways. As a consequence, the speed of the car largely varies for the different sequences. Table 4.1 lists for all of the test sequences in the dataset in which kind of environment the data was captured as well as the trajectory length.

Just like in chapter 3, we will use the relative pose error (RPE), defined in eq. 2.9, to evaluate the results. The error is measured and averaged for different intervals $\Delta \in \{100\text{ m}, \dots, 800\text{ m}\}$. We compare the 'standard' point-to-plane ICP variant with our improvements presented in the previous sections including the geometrically stable point selection and feature-based weighting. Thereby we take into account the overall accuracy, stability, robustness and maximum initial alignment. For all of the experiments, we adopt the point-to-plane cost function given by eq. 3.3 as this cost function is best suited for outdoor environments, cfr. chapter 3.

Regarding the standard point-to-plane ICP method, we use a k-nearest ($k = 20$) neighbour selection for the estimation of the surface normals. In order to select the optimal radius in our method, cfr. eq. 4.8, we first determine the average distance d of a point to its closest neighbour, and we quantise the radius interval $[r_{min}, r_{max}]$ as follows: $r \in \{\frac{1}{2}d, d, 2d, \dots, 2^5 d\}$. The normals are then estimated following the entropy-based neighbourhood selection method described in section 4.4. The experiments were conducted on a computer with an Intel Core i7-7820X CPU @ 3.60 GHz, 128GB RAM and an nVidia Geforce GTX 1080 Ti inside.

N	avg. E_t (%)	avg. E_r (deg/m $\times 10^2$)	avg. time (ms)	avg. no. it.
1000	1.27	0.73	60	6.17
1500	1.22	0.69	71	6.33
2000	1.18	0.68	79	6.52
2500	1.19	0.69	91	6.79
3000	1.20	0.68	95	6.79

Table 4.2: The average translational and rotational error and the computation time on the KITTI sequences for our GSS algorithm for varying number of samples N per 'list'.

4.6.1 Geometrically stable sampling

Regarding our geometrically stable sampling (GSS) strategy, we first determine the optimal number of samples per 'observability' category or list. Sometimes, the same point will be selected for two or more observability categories. In that case, we only keep the point once. Table 4.2 summarizes the RPE and execution times for a varying number of samples ranging from 1000 to 3000. We see that the error is decreasing for a higher number of samples up to $N = 2000$, after which the error is slightly increasing again. The computation time is logically increasing for a higher number of samples going from 60 ms for $N = 1000$ to 95 ms for $N = 3000$. However we also notice that the average number of iterations is slightly increasing for higher N. Expressing the computation time per iteration, we see that it is increasing from 9.7 ms for $N = 1000$ to 14 ms for $N = 3000$.

For $N = 2500$, we further investigate the differences between the sequences as this value leads to the largest improvement for the majority of the sequences. The RPE, both the translational error E_t (%) and rotational error E_r (deg/m $\times 10^2$) are summarized in Table 4.3. The first thing that stands out in this experiment is that there are major differences in accuracy between the different sequences. The main reason for this is that some of the sequences have areas along their trajectory that lack structural information from which no geometrical features can be derived. Sometimes, the car is driving in rural areas or on highways where there is only vegetation along the road. This is the case for the 'KITTI01' sequence as can be seen in Figure 4.7.

Apart from the lack of large structures, the high speed of the car on the highway (more than 70 km/h) is also causing problems. In those cases, the point clouds acquired in consecutive sweeps differ quite a lot from each other, making correct data association very hard. On top of that, there are other cars driving at approximately the same speed of the vehicle, causing another risk to accurately estimate the ego-motion. In our method, the points on the cars will highly contribute in the

	E_t (%)		E_r (deg/m $\times 10^2$)	
Sequence	ICP	GSS-ICP	ICP	GSS-ICP
KITTI 00	1.39	1.10	0.94	0.80
KITTI 01	3.85	3.48	1.02	0.94
KITTI 02	1.58	1.24	0.81	0.66
KITTI 03	1.07	0.89	0.78	0.74
KITTI 04	0.92	0.26	0.35	0.23
KITTI 05	1.16	0.83	0.81	0.71
KITTI 06	1.08	0.82	0.68	0.63
KITTI 07	0.51	0.51	0.58	0.58
KITTI 08	1.68	1.27	0.97	0.80
KITTI 09	1.13	1.10	0.86	0.75
KITTI 10	1.94	1.62	0.93	0.75
AVG	**1.48**	**1.19**	**0.79**	**0.69**

Table 4.3: The translational (E_t) and rotational error (E_r) on the KITTI sequences for our ICP algorithm with geometrically stable sampling (GSS) using $N = 2500$ samples per list, compared to the standard point-to-plane ICP algorithm. For the majority of the sequences, GSS-ICP substantially reduces the error.

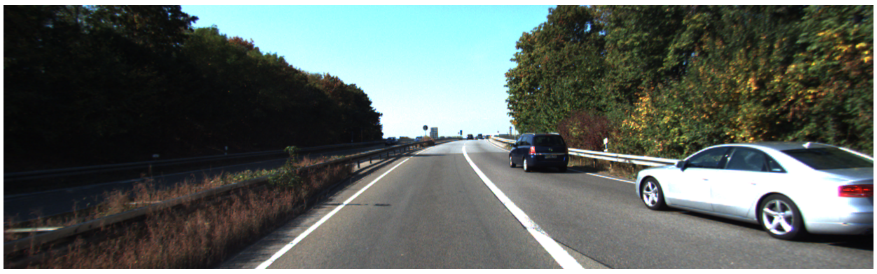


Figure 4.7: Sample image from 'KITTI01'. The fact that no large buildings are next to the highway, in combination with the high speed of the car and the presence of other moving cars, makes it challenging to correctly estimate the ego-motion of the vehicle.

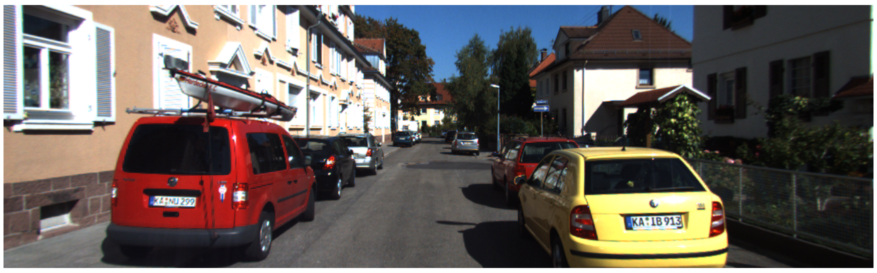


Figure 4.8: Sample image from 'KITTI07'. Thanks to the presence of large buildings and parked cars along the street, the ego-motion estimation for this sequence is highly accurate.

pose estimation process, as they are solid objects with a high planarity. The only solution to this problem is to entirely exclude all moving objects, which is far from trivial when solely relying on the point cloud data. For this particular 'KITTI01' sequence, it is thus hard to achieve a good result. However, our improved GSS-ICP algorithm still manages to reduce the translational error from 3.85% to 3.48% and the rotational error from 0.0102 deg/m to 0.0094 deg/m.

The best results are logically obtained for sequences recorded in urban environments as these scenes generally consist of large planar surfaces. Figure 4.8 depicts an image from the 'KITTI07' sequence that was recorded in a city center. As the entire scene consists of large buildings and parked cars, there are almost no geometrically 'unstable' areas. As a result, even the standard ICP algorithm succeeds well in estimating the ego-motion. However, our method still benefits from the fact that fewer points are needed to estimate the ego-motion, which results in faster computation times. Finally, figure 4.9 depicts an image from the 'KITTI03' sequence, recorded in a countryside area. The environment consists of a combination of vegetation and buildings. For these environments, our GSS-ICP method has the largest benefit compared to the standard ICP method.

From Table 4.3, we can conclude that for almost all the sequences, our GSS-ICP method is outperforming the standard ICP algorithm. In general, we obtain a reduction in average translational error from 1.48% to 1.19% and an average rotational error from 0.0079 deg/m to 0.0069 deg/m.

4.6.2 Feature-based weighting

The results from the previous section can be further improved by extending our method with feature-based weighting. To that end, we define

$$\mathbf{x} \doteq [\psi_1, \psi_2, \psi_3, O, a, c, e_\Psi]$$

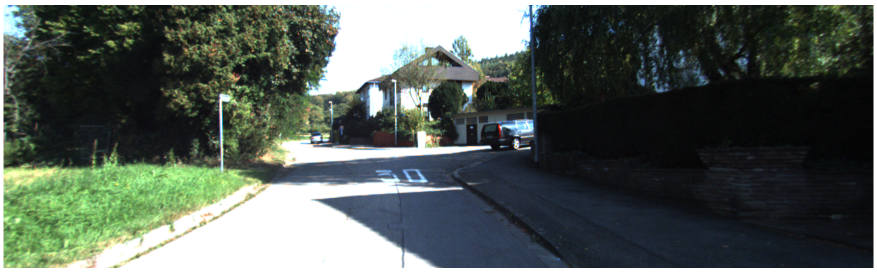


Figure 4.9: Sample image from 'KITTI03'. The presence of abundance vegetation along the road in countryside areas, makes it difficult to accurately estimate the ego-motion. However, our approach, which favours planar areas, can largely increase the accuracy compared to traditional techniques as it focuses on the more stable regions, such as the (few) buildings that are still present.

Feature	Formulae
Linearity	$\psi_1 = \frac{\sigma_1 - \sigma_2}{\sigma_1}$
Planarity	$\psi_2 = \frac{\sigma_2 - \sigma_3}{\sigma_1}$
Sphericity	$\psi_3 = \frac{\sigma_3}{\sigma_1}$
Omnivariance	$O = \sqrt[3]{\prod_{i=1}^{3} \sigma_i}$
Anisotropy	$a = \frac{\sigma_1 - \sigma_3}{\sigma_1}$
Local surface variation	$c = \frac{\sigma_3}{\sum_i \sigma_i}$
Entropy	$e_\Psi = -\sum_{i=0}^{3} \psi_i \ln(\psi_i)$

Table 4.4: Summary of the 3D shape features used in the weighting process.

as the feature vector as this combination was leading to the best results in the experiments. The definitions of these values are summarized in table 4.4. The results of our improved method, which we denote as W-GSS-ICP, are listed in table 4.5. We see that for the majority of the KITTI sequences, the feature-based weighting is beneficial for the overall accuracy. The W-GSS-ICP further reduces the translational error from 1.48 $\%$ (ICP) and 1.19 $\%$ (GSS-ICP) to 1.09 $\%$ and the rotational error from 0.0079 deg/m and 0.0069 deg/m to 0.0067 deg/m.

To obtain a better idea of the performance gain, we have plotted the ground truth and estimated trajectories for three different sequences. These trajectories are the result of projecting the 6D pose on the ground plane. Figure 4.10 shows the results for the 'KITTI04' sequence. We notice that the standard ICP method has problems in the beginning to estimate the right translation due to the abundance of vegetation and the lack of salient geometrical features. Later on, the estimation is more accurate, but the small error is taken along the rest of the trajectory, resulting

	E_t (%)		E_r (deg/m $\times 10^2$)	
Sequence	ICP	W-GSS-ICP	ICP	W-GSS-ICP
KITTI 00	1.39	1.03	0.94	0.75
KITTI 01	3.85	3.11	1.02	0.85
KITTI 02	1.58	1.15	0.81	0.66
KITTI 03	1.07	0.90	0.99	0.74
KITTI 04	0.92	0.14	0.35	0.17
KITTI 05	1.16	0.68	0.81	0.69
KITTI 06	1.08	0.61	0.68	0.52
KITTI 07	0.51	0.51	0.58	0.60
KITTI 08	1.68	1.31	0.97	0.83
KITTI 09	1.13	1.06	0.86	0.73
KITTI 10	1.94	1.51	0.93	0.78
AVG	**1.48**	**1.09**	**0.79**	**0.67**

Table 4.5: The translational (E_t) and rotational error (E_r) on the KITTI sequences for our GSS-ICP method extended with feature-based weighting (hence W-GSS-ICP), compared to the standard point-to-plane ICP algorithm. For the majority of the sequences, W-GSS-ICP further reduces the error compared to GSS-ICP.

in an error of approximately 25 meters in the direction of the z-axis at the end of the sequence.

A similar phenomenon can be noticed in the results on the 'KITTI06' sequence, depicted in Figure 4.11. For the standard ICP algorithm, the translation in the direction of the z-axis is wrongly estimated in the beginning, causing the entire trajectory to be shifted from the ground truth. Finally, in Figure 4.12, the trajectory for the 'KITTI01' sequence is depicted. The first thing that can be noticed from this trajectory is that the error of standard ICP is mainly in the direction perpendicular to the ground plane. The reason for this is that the majority of the points lie on the ground plane. Thus, while minimizing the point-to-plane error metric, the correct estimation of the surface normals for points belonging to the ground plane plays a crucial role. If a fixed (small) number of nearest neighbours are taken, it can happen that due to bumpy parts of the floor or inaccurate measurements, the normal vector is slightly deviating. However, the slightest deviation in normal vector can quickly degenerate the minimization of the point-to-plane distance, leading to wrongly estimated poses with a bias mainly in the y-direction, perpendicular to the ground plane. We thus benefit from our ideal neighbourhood function, as the normal vectors are computed upon a lot more points making them more robust against small inaccuracies of the sensor. We see that for our W-GSS-ICP method, the error made in the direction of the y-axis is reduced.

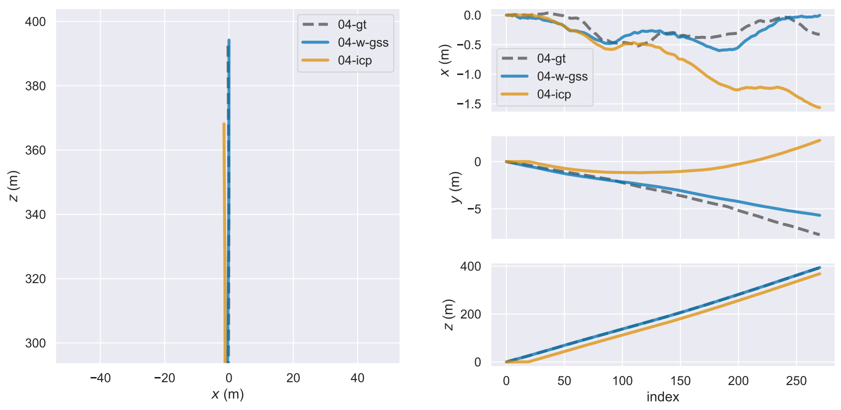


Figure 4.10: Left: the resulting trajectories on the 'KITTI04' sequence using our W-GSS-ICP method (blue line) and the standard point-to-plane ICP method (orange line) compared to the ground truth (dash line). Right: Plot of each coordinate separately. W-GSS-ICP greatly improves the translation estimate in both the x and y direction.

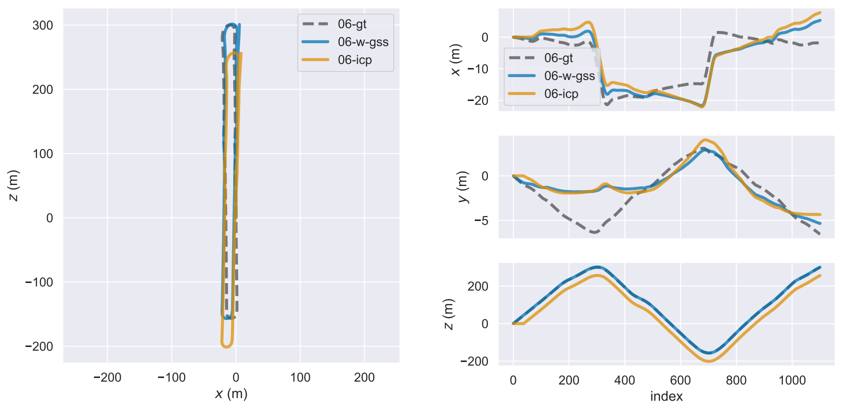


Figure 4.11: Left: the resulting trajectories on the 'KITTI06' sequence using our W-GSS-ICP method (blue line) and the standard point-to-plane ICP method (orange line) compared to the ground truth (dash line). Right: Plot of each coordinate separately.

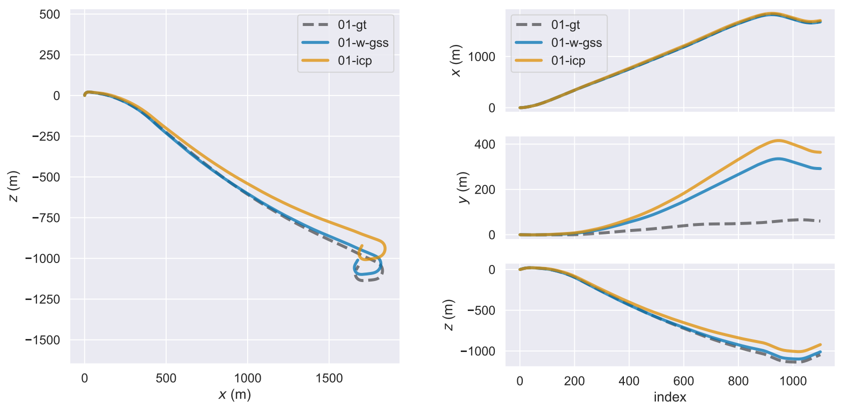


Figure 4.12: Left: the resulting trajectories on the ‘KITTI01’ sequence using our W-GSS-ICP method (blue line) and the standard point-to-plane ICP method (orange line) compared to the ground truth (dash line). Right: Plot of each coordinate separately. W-GSS-ICP greatly improves the translation estimate in the z direction and slightly improves the translation estimate in the y direction.

One of the main factors that influences the effectiveness of ICP-based methods is the speed at which the sensor is moving. The KITTI sequences are captured with a Velodyne HDL-64E, which is spinning at 10Hz, hence producing a point cloud every 100 ms. When the speed of the vehicle is too large, the data association process in the ICP method will obviously fail as it relies on ‘closest’ points between two shifted point clouds. In Figure 4.13 we plotted the average translational error in function of the speed. We can see that our W-GSS-ICP method is resulting in a lower translational error for all speeds. However, from 65km/h onwards, both standard ICP and our W-GSS-ICP quickly degenerate, hence imposing a maximum on the speed that is still resulting in acceptable accuracies. We notice a small drop in the translational error for ICP near 50 km/h, but we are uncertain about the reason.

4.7 Conclusion

In this chapter, we proposed improvements over the standard ICP algorithm by incorporating low-level shape-features in order to obtain more accurate and more robust pose estimates. Our improvements are threefold. First, we designed an optimal neighbourhood selection method upon which surface normals are more accurately estimated. The determination of the optimal neighbourhood is a combination of an efficient point clustering and entropy-based radius selection. Second,

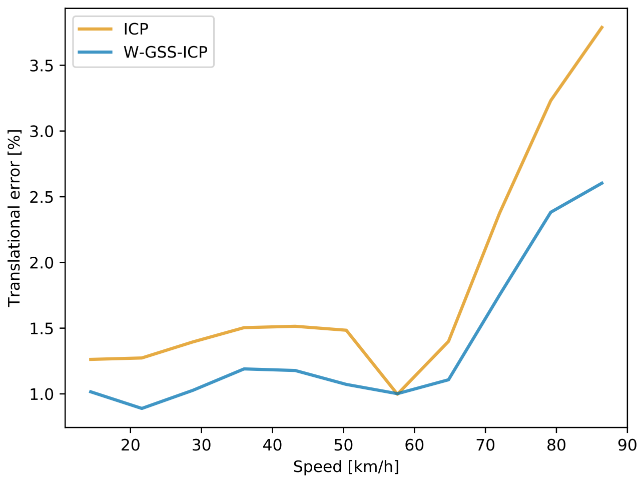


Figure 4.13: The translational error (%) on the KITTI dataset for increasing speeds. Our own W-GSS-ICP method clearly performs better than the standard (point-to-plane) ICP method. Both methods however have a high increase of translational error at a speed of 65 km/h, hence posing an upper limit on the speed that is still resulting in acceptable accuracies.

we proposed a geometrically stable point selection algorithm, taking advantage of the planarity in the scene. Furthermore, we cope with wrong directional biases due to the non-uniform point density, by proportioning the observability in all directions. Third, we developed a feature-based weighting scheme to penalize bad point correspondences.

The experiments on the KITTI dataset demonstrate that we are able to reduce the translational error from 1.48% to 1.09% and the rotational error from 0.0079 deg/m to 0.0067 deg/m compared to the original point-to-plane ICP algorithm. The careful selection of the points makes the convergence of the ICP algorithm more stable. Because of this, the number of iterations until convergence for the KITTI dataset averages around 6 to 7 and as a result the transformation can be computed in less than 100 ms. Given the fact that lidar scanners, such as the Velodyne or Ouster series, are generally acquiring 10 point clouds per second, our registration techniques runs in real-time.

Regarding the speed of the vehicle, the performance of both the standard point-to-plane ICP and our own W-GSS-ICP algorithm remains high until 65 km/h is reached. From that speed onwards, the performance decreases drastically for both methods, but our methods still yields the best results for all speeds.

5 Efficient scan-to-map matching

The previous two chapters were dealing with the alignment of point clouds, a process resulting in a transformation matrix that allows fusing them into one coherent coordinate system. Doing so, we create *'aggregated point clouds'*, which can - after conversion to a simplified 3D map - be used to perform obstacle avoidance, navigation and path planning. The goal of this chapter is twofold. First, we explain how these aggregated point clouds can be used to improve the scan matching process described in the previous chapters. In other words, we elaborate on how we can associate new sensor measurements to the reconstructed scene in order to improve pose estimation.

The *'fusion'* process will be based on a surface reconstruction: we approximate the underlying surface and re-project the points on it in order to reduce noise and small registration errors. The reconstruction also allows to fill small gaps in regions that are not or too sparsely sampled. This process will eventually help solving the correspondence ambiguity in the data association process. As we will have a continuous surface available, we can correctly match the exact same physical point among different scans. Second, we will investigate how we can efficiently store these aggregated point clouds as a 3D map for online use. Besides obstacle avoidance and navigation, this online use also comprises updating the maps on the fly facilitating the challenge of 'livelong mapping'.

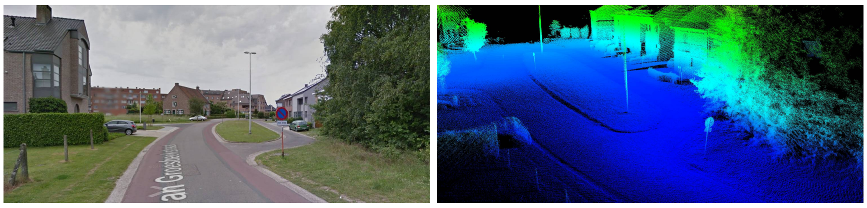


Figure 5.1: Google street view image of a road in the Belgian city Hasselt (left) and the associated point cloud in full resolution (multiple points per square centimeter). The Velodyne HDL-32e scanner can produce up to 700k points per seconds, leading to approximately 8.4MB per second or 0.5GB per minute of acquired data.

5.1 Introduction

As mentioned in the previous chapters, one of the main uses of 3D maps is navigation. Navigation devices today are however still dominated by static 2D maps or meter-resolution 3D alternatives. These low-resolution maps may be sufficient for human navigation, but autonomous cars need maps that tell them where the curb is within a few centimeters. In addition, autonomous cars should be able to distinguish between traffic signs, street lights, bus stop signs and so on. Even more, they should have the ability to recognize which type of traffic sign they encounter. All of this implies that the autonomous vehicles of the future need high-resolution maps for them to fully operate.

On the other hand, maps need to be compact in order to fit in the memory of computers and to be efficiently distributable among other autonomous systems. Moreover, it should be possible to quickly update them in a way that changes, both abrupt (e.g. sudden obstacle on the road) or long term (e.g. damages on the road surface) can be easily integrated. Being both compact and quickly up-datable are two prerequisites to develop fully dynamic and ever up to date maps. Whether the map is built from scratch or not, the key challenge in the future will be on how to maintain it and how to incorporate a tremendous amount of data in an efficient way.

Today, many approaches adopt *out-of-core* implementations in which a part of the map is released from the RAM memory and saved to disk. Figure 5.1 depicts a Google street view image of a residential area together with a full-resolution 3D reconstruction using data from a Velodyne HDL-32e. This scanner is able to capture 700k points per second leading to approximately 0.5GB per minute of acquired data. It is thus clear that we need data structures that make it possible to quickly access the right data from this huge point cloud. In all this, we should keep in mind that different applications impose different requirements on the level of detail or the speed of retrieval.

Often times, there is a relationship between the importance of objects and their spatial properties. The majority of interesting objects are found in specific locations. In case of autonomous cars, these are mostly found along the road, such as curbs, traffic signs, bus stops, etc. We could thus model these particular areas with finer granularity than the road area itself. On the other hand, it should still be possible to detect unpredictable obstacles on the road or imperfections of the road surface (e.g. bumpy parts or holes). This imposes a requirement on the minimum resolution at which the road area should be modelled in the map. Furthermore, for autonomous cars, buildings along the road are less critical and do not require high detail, but they could still serve as visual cues for integration with other GIS systems. For these elements it is thus sufficient to have their rough bounding box in the map. In other cases such as urban planning, one might still be interested in the fine details of buildings. A particular example is the deployment of optical fiber along the facades of houses in order to provide internet to families. For this, the internet service provider needs the precise layout of the street scene at centimeter accuracy in order to figure out where to attach the wires.

Another key element in the future of map building will be how to combine data from different modalities into one coherent space. It is clear that there is a need for *spatial partitioning algorithms* as all of the existing sensors have different properties in terms of resolution and point density. The good news is that we can learn from algorithms in the computer graphics literature that already deal with the efficient partitioning of 3D spaces in rendering engines. In those use cases, they are used to perform operations like collision detection or frustum culling, which are useful to determine the region of interest or identify obstacles.

In this chapter we will describe how we can efficiently build (or organize), maintain and update 3D maps of large-scale environments. We will focus on two main aspects. First, we study surface reconstruction algorithms, which reduce the noise and measurement errors as well as errors made by the pose estimation. This process will further enhance the quality of the map, i.e. the overall accuracy, but it will also provide a means of lowering the spatial redundancy in the scene, which on its turn benefits the compactness of the map. Second, we investigate efficient data structures and spatial partitioning, which are necessities in order to keep the computation time limited. We will discuss how octrees can serve as a compact representation of the scene and how they can be used to compress the 3D map for potentially streaming the data to other systems or servers.

The chapter is organized as follows. In the next two sections, we will give a brief literature overview on the two aspects of this chapter being 1) map representation and 2) surface reconstruction. In section 5.4 we will describe how we exploit the two aspects in order to increase the accuracy as well as the speed of 3D mapping, followed by a thorough evaluation.

5.2 Map representations

In the early mapping literature, one often made the distinction between *topological* and *metric* map models. The former one considers places and the relations between them. The map can then be seen as a graph in which the nodes correspond to places and the edges to the paths connecting them. This kind of representation is well-suited to find out which other places are reachable from a given place and is generally used when the goal is to perform navigation. However, metric map models are way better suited for interactions with the scene or to analyse the environment. In this work, the main focus is not on navigation but rather on comprehensive analysis, including use cases as damage assessment, inspection, creation of digital twins, etc. In this section we will, therefore, focus on the metric map representations. We discuss the five most prominent ones: feature-based, dense, spatial-partitioning-based, boundary-based and object-based.

5.2.1 Feature-based sparse representations

Most of the early SLAM systems represented the scene as a set of *sparse* 3D features, derived from discriminative features in the scene. As such, they are based on the extraction and matching of salient features in sensor data, either images or depth measurements. Examples of appearance-based features are SIFT (scale-invariant feature transform) [Lowe 04], SURF (speeded-up robust features) [Bay 08], BRIEF (binary robust independent elementary features) [Calonder 10] or ORB (Oriented FAST and Rotated BRIEF) [Rublee 11]. These appearance-based features have been widely applied in the field of robotics but are also used in the context of structure from motion. In the latter domain, the sparse 3D point cloud is usually densified in a subsequent step by adopting a multi-view stereo (MVS) algorithm such as GIPUMA [Galliani 15] or COLMAP [Schönberger 16]. The complexity of these methods, however, prevent real-time operation. Figure 5.2 depicts an example of ORB-SLAM [MurArtal 15], a feature-based visual SLAM system that is running in real-time, but results in a sparse map of the environment.

With the advent of novel, affordable, depth sensing modalities such as ToF cameras and laser scanners, some researchers also focused on the use of 3D features to conduct SLAM. Examples are SIFT-3D [Scovanner 07], Harris3D [Sipiran 10a], Spin images [Johnson 97], NARF [Steder 10], and SHOT [Salti 14]. The main drawback of this approach however is that it is very challenging to extract repeatable 3D features from 3D (laser) scans due to their sparse and non-uniform point density. Apart from that, they are also extremely costly to compute. For those reasons, feature-based systems are not often used in combination with 3D scan data.

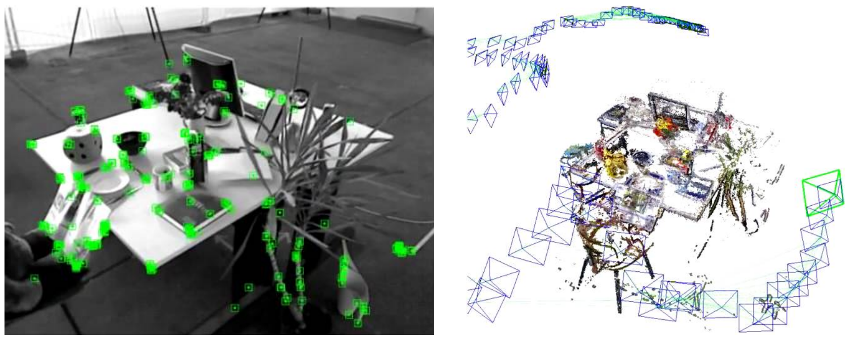


Figure 5.2: Example of ORB-SLAM [MurArtal 15], a feature-based SLAM system. Left: the ORB features extracted from the image. Right: The reconstructed sparse 3D point cloud together with the camera poses.

5.2.2 Low-level dense representations

In contrast to feature-based representations, dense representations provide high-resolution 3D models of the scene, usually in the form of a large unstructured set of points or point cloud. These methods lately received more attention in combination with direct methods, which estimate the trajectory of the robot or camera directly from the 'raw' data points as opposed to feature-based approaches. In the context of visual SLAM, this means that the intensity values of all the pixels in the image are used, like in LSD-SLAM [Engel 14] or DVO-SLAM [Kerl 13a]. Regarding SLAM using 3D sensors, all methods based on the ICP algorithm that do not incorporate an underlying data structure fall in this category as well.

Sometimes, the raw data points are *augmented* with surface information, hence creating so-called *surfel maps*, which encode the geometry as a set of disks, as shown in the right-most image of Figure 5.3. The main drawback of dense representations is their high memory footprint. For large-scale environments a dense representation quickly becomes intractable and is, therefore, not suitable for long-term use in autonomous systems. Moreover, due to their unstructured nature it is almost impossible to perform scan-to-map matching. As such, these systems usually only integrate scan-to-scan matching.

5.2.3 Spatial partitioning representations

Space partitioning is the process of dividing the 3D space, often Euclidean, in non-overlapping regions. The best-known spatial partitioning representation is the *spatial-occupancy* map, which decomposes the 3D space into identical cubes (voxels) arranged in a regular 3D grid. As searching and updating this kind of

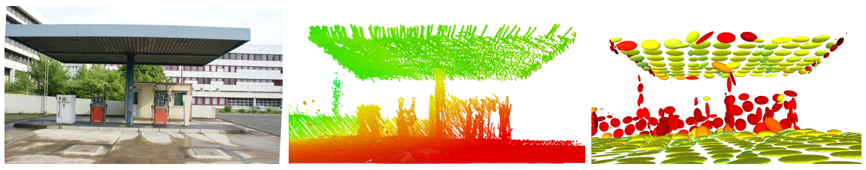


Figure 5.3: Example of dense SLAM. Left: image of an outdoor scene, middle: point-cloud based representation and right: surfel representation.

representation is rather inefficient, the 3D grid is often organized in the form of a hierarchical tree data structure, which has a $O(\log n)$ search complexity. There exist several hierarchical partitioning schemes, such as the *octree*, the *binary-space partitioning* or BSP-tree and the kd-tree. A major limitation of the latter two is that they are complicated and time-consuming to update them. It is very likely that the tree has to be rebuild when newly acquired points of the scene have to be inserted as the tree would otherwise become unbalanced. In the latter case the search complexity would degrade back to $O(n)$. For that reason, octrees are more often used in robotics as they are more flexible when the map is extended or in case many moving objects are present in the scene.

In [Hornung 13], the authors presented OctoMap, a probabilistic 3D map representation based on octrees. This framework was used by [Beno 16] to map the environment and subsequently help in navigating autonomous robots. As mentioned in the introduction, sometimes an application requires different levels of detail for different regions in the environment. The authors of [Wurm 11] propose to represent the map as a hierarchy of octrees in which each map node is represented as a separate octree, allowing to subdivide the scene on the object level. Each sub-map is maintained independently and the parameters, such as the resolution, can be adjusted for each of them separately. Each sub-map can also be manipulated independently, such that individual objects can be moved while others remain static. Finally, in environments without hanging obstacles, 2.5D elevation maps have also been used [Brand 14].

5.2.4 Boundary representations

Boundary representations define the scene in terms of their surface boundary. These representations are hence better suited to conduct obstacle detection, avoidance and path planning in the field of robotics. Moreover, they can help making the pose estimation more accurate and more robust as they allow to conduct scan-to-map matching in a more efficient way. First, the matching can be computed using a larger part of the scene which limits the risk of poor data association. Second, as they aggregate data from multiple scans, they can cope with measurement noise from the sensor.

As we will discuss later in this chapter, the surface can either be represented in an explicit or implicit way. Among the explicit methods are the curve-based representations using NURBS or B-splines and the surface mesh models, which are connected sets of polygons, usually triangles. Implicit surfaces are defined by the zero crossings of a function defined on $\mathbb{R}^3$. Examples of functions include *radial-basis functions* (RBF) [Carr 01], *signed-distance functions* (SDF) [Curless 96] and *truncated signed-distance functions* (TSDF) [Zach 07]. The latter group gained a lot of popularity after the seminal paper *'KinectFusion'* of [Newcombe 11]. Another group of a boundary representations are plane-based maps, which have been used by [Pathak 10; Kaess 15].

5.2.5 High-level object-based representations

Today, most of the operational SLAM systems either use a dense representation in the form of a point cloud or a spatial partitioning data structure such as the octree. However, we envision that the future of SLAM will embrace the use of object-level maps or maps containing solid shapes. One of the first SLAM systems that exploited this idea is the work of [Civera 11], in which a 'hybrid' solution was proposed. Their monocular EKF-based SLAM system is extended with an object recognition thread, which informs about the presence of an object in the sequence by searching for SURF correspondences and checking their geometric compatibility. When an object is recognized it is inserted in the SLAM map where it helps estimating future poses while its position is being refined by the SLAM algorithm at the same time.

In [Dame 13], the authors propose something similar, combining dense SLAM with 3D object pose and shape recovery. More specifically, they augment their SLAM system with the 6D pose and additional degrees of freedom for the objects of known class in the scene, combining image and depth data for the pose and shape recovery. In [SalasMoreno 13], the authors propose their *'SLAM++'* system comprising a localization algorithm based on complete objects thereby exploiting the assumption that many scenes consist of repeated, domain-specific objects and structures.

The main drawback of those high-level object-based approaches is that it comes with the overhead of creating an object database. In case of [SalasMoreno 13], the authors first reconstruct high quality 3D models of the objects in the scene, in order to exploit them afterwards in their SLAM system.

5.3 Surface reconstruction

The boundary representations from the previous section are in some way related to the topic of surface reconstruction as many of them basically represent a kind of surface model. In this section we will give a brief overview of the different ways to reconstruct the surface from a set of sampled points. The latter can be seen as an inverse problem: given a set of measurements, find the geometric primitives that explain these measurements.

The reason to reconstruct the surface can vary. Some applications require a detailed representation of the environment that is easy to interpret and analyse. Other applications require a compact representation that can quickly be rendered, for example on virtual reality devices. The main goal of the surface reconstruction in this work, however, is to improve the registration in view of more accurate 3D reconstructions.

5.3.1 Explicit vs. implicit surfaces

Surface reconstruction algorithms generally fall into two categories: explicit and implicit reconstruction. Explicit surfaces can be seen as a description of the precise location of the surface. This description can either be in a parametric form or can be expressed by means of a triangulation. A parametric surface uses deformations of a primitive model that cover an arbitrary portion of the points. Example primitives being used are B-splines, NURBS, planes, spheres and ellipsoids. Triangulated surfaces use the points as vertices and compute the connectivity (triangulation) between the point samples. The main limitation of these methods is their high sensitivity to noisy data. Moreover, scans are not always perfectly aligned, creating additional artefacts that are incorporated in the triangulation. These methods, therefore, require to create new vertices in overlap regions, leading to a lot of additional computational burden.

Implicit surfaces on the other hand divide the space into two areas, inside and outside of the object. They also come in parametric and non-parametric representations. The former are usually defined as the zero set of an indicator function. Examples are *radial-basis functions* (RBF), Poisson, Fourier or Wavelet surface reconstruction. Non-parametric representations gained a lot of attention after the *'KinectFusion'* system was presented by [Newcombe 11]. The authors use an implicit surface, represented by a *truncated signed distance function* (TSDF) which is coded in a voxel map. A TSDF is thus a volumetric representation of the scene where each location stores the distance to the closest surface.

In [Newcombe 11], the voxel data structure is implemented as a 3D array

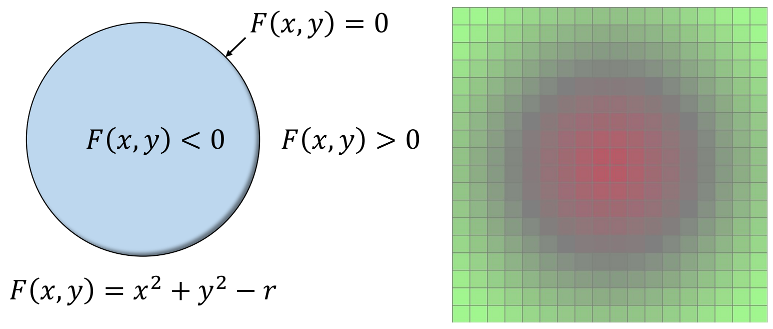


Figure 5.4: Example of an implicit surface, i.e. a signed distance function (SDF), in parametric form (left) and defined by a voxel grid (right).

stored in GPU memory with each cell containing a distance value and a weight proportional to the uncertainty of the surface measurement. During initialization, two main parameters are configured, being the size of the TSDF in voxels and the dimension in meters. These two values affect the size of the area that can be reconstructed and the resolution, which on its turn affects the performance and accuracy. KinectFusion can deliver acceptable results but the fact that the TSDF is defined by a fixed voxel grid makes it only suitable for small 'work' spaces.

Figure 5.4 demonstrates the idea of implicit surfaces, both in parametric and non-parametric form. Implicit surfaces need to be post-processed in order to 'visualize' them. Marching cubes [Lorensen 87] is a method that is widely used to generate a triangular mesh from an implicit surface.

5.3.2 Interpolated vs. approximated surfaces

Another way of subdividing surface reconstruction is by distinguishing approximation methods from interpolation methods. Interpolation methods are used for scanning data that suffer from non-uniformity or undesired holes (missing data). An interpolated surface is one for which at least a few of the input points lie on the actual surface. Almost all the explicit surfaces are considered interpolated surfaces, specifically triangulated meshes as all of the scanning points rest on the surface. Approximation methods on the other hand define a smooth function that does not pass necessarily through the point samples. They are better able to handle noise in the data. On the other hand, they are often limited by their incapability of representing drastic changes on the surface, such as edges and corners.

In this work, we adopt an approximation approach, which is motivated by the fact that we want to account for the measurement noise of the 3D sensor and for the imprecisions in prior pose estimation steps. Interpolation methods could enlarge these artefacts and deteriorate the reconstructed surface. By approximating a continuous function, we avoid that flat areas grow perpendicular to the plane, e.g. points sampled on a wall are prevented from being ‘spread’ and forming a ‘thick’ wall in the point cloud. The latter will improve further localization as in that case the ICP energy function has a well-defined minimum. In order to cope with smoothing along edges and corners, we rely on the ideal neighbourhood selection method described in the previous chapter. There, we proposed to select the set of neighbours that are part of the same object or geometric primitive and for which the Eigen entropy is minimized.

5.4 Proposed approach

The previous chapters were dealing with the alignment of consecutive scans in order to find the transformation matrix defining the position and orientation of the sensor. We have demonstrated that due to inaccurate measurements, the estimate of this pose matrix can also be inaccurate. For that reason, it is beneficial to conduct an additional scan-to-map alignment in order to further refine it. The result from the scan-to-scan matching can herein be used as a prior estimate.

Our final approach consists of two main components. First, we adopt an octree-based map to accelerate spatial queries, such as nearest neighbour searches for the data association step. Second, on top of the octree backbone we conduct a moving least squares (MLS) surface reconstruction to deal with noise as a result of inaccurate measurements and prior pose estimates. The MLS technique is an approximation method that smooths the data, handling well the noise without over-smoothing as is often the case for Poisson reconstruction. The MLS approximation can be used to re-sample the point cloud afterwards, ensuring a uniform point density, which improves the robustness and accuracy of the scan-to-map alignment. Finally, it can also be used to fill small gaps in the surface.

5.4.1 Octree-based map

One of the main limitations of the TSDF used in KinectFusion is its fixed-size voxel grid. However, the final extent of the scene to be mapped is not always known in advance. To overcome this, we propose to use an octree, which can be more easily resized. As shown in Figure 5.5, an octree is a hierarchical data structure that represents the environment as a cube, which is iteratively subdivided by 8 smaller disjoint sub-cubes or *octants*. The cube at the root level thereby serves as an axis-aligned bounding box of the environment. If a sub-cube is not occupied, i.e. there are no objects or points present, the division of that cube stops at that

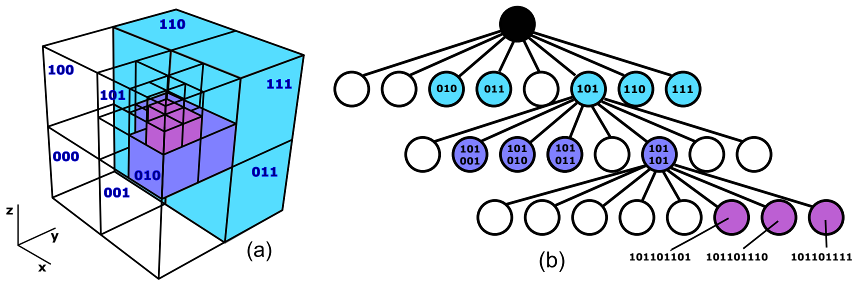


Figure 5.5: The octree data structure (b) and its representation in 3D (a). In a *linear* octree representation, each child octant is represented by a *location code* based on its position in the parent octant. The location code of any child node in the tree is computed recursively by concatenating the octant numbers of all the nodes from the root down to the node under consideration. Only leaf nodes are saved and stored contiguously in memory. This hierarchical space partitioning is the backbone of our 'map', making it possible to quickly associate new sensor measurements to the map under construction.

resolution. Each node in the tree can hence have at most eight children but can also have no children at all.

During initialisation it is still necessary to define a bounding box of the scene. However, once the sensor is leaving its original bounding volume, a new root node is created and the 'old' bounding box is fitted in one of the eight child nodes of the new root node. Still, it is crucial that the octree is expanded into the direction where new measurements are acquired. Therefore, we choose the center of the root node in such a way that the new 'outlying' measurements fall into it.

5.4.1.1 Common octree representations

Octrees can be represented in several ways. As it would lead us too far too describe them all in detail, we briefly discuss the three common approaches below.

Full octree

In a full octree every internal node has eight children and all leaf nodes have exactly the same tree depth d. A full octree, therefore, has $N = 8^d$ leaf nodes and the total number of tree nodes is given by $\sum_{i=0}^{d} 8^d$. The main benefit of this approach is that the address of any node can be calculated based on its location in the tree, making the use of explicit pointers unnecessary, which saves memory. However, if many nodes contain no objects, the memory savings of small nodes quickly get lost by the huge amount of allocated nodes. Full octrees are, therefore, only useful for scenes that are non-sparse and static with evenly distributed geometry. In case of mapping large-scale 3D environments, many nodes will be empty and for that reason full octrees are not the best choice.

Pointer-based octree

One of the most common approaches are the pointer-based implementations, also referred to as explicit octrees. There are two common representations. In the first *'exhaustive tree'* representation each node has eight pointers to each of the children and a reference to the data. The second representation, called *sibling-child* representation, stores for each node a pointer to its first child and to the next child of its parent and a reference to the data. The second approach is a lot more memory efficient as it only needs 2 pointers instead of 8, but it requires on average more pointer dereferences to access a given node. Sometimes an additional pointer is stored to access the parent node, in order to accelerate bottom-up traversals.

Linear octree

Linear octrees are a variation on pointer-less octrees in which only leaf nodes are saved and stored contiguously in memory. In linear octrees, each node is represented by an *interleaved* base-8 code, called a *location code* or *Morton code* [Morton 66], based on its relative position in the octant (cfr. Figure 5.5): bottom-left-front (000), bottom-right-front (001), bottom-left-back (010), bottom-right-back (011), top-left-front (100), top-right-front (101), top-left-back (110), top-right-back (111). The location code of a child node in the tree is then computed recursively by concatenating the octant numbers of all the nodes from the root down to the node under consideration. The key k_n of the parent of node n can simply be obtained by removing the 3 last bits: $k_n \gg 3$. With three bits per octree level, a 32 bit location code can represent a tree with a maximal depth of 10, a 64 bit location code a maximal depth of 21. Thus, rather than connecting the leaf nodes via interior nodes, the leaf nodes are sorted by location code and then laid out sequentially in memory. When the leaf nodes are sorted according to their location codes, the order is the same as the pre-order (depth-first) traversal of the tree, often referred to as the *Morton*-order or Z-order indexing, cfr. Figure 5.6. Doing so, the data locality is preserved as well, which leads to superior performance in memory access.

Linear hashed octree

The octree nodes can further be stored in a hash map - an associative container data structure - which allows to address them based on their Morton code. A hash table

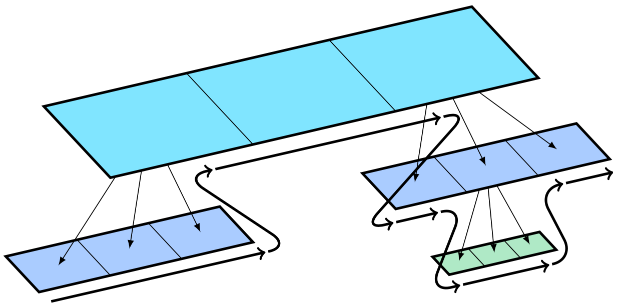


Figure 5.6: We store the leaf nodes in memory according to their *Morton* codes, which corresponds to the pre-order (depth-first) traversal of the tree, also referred to as the *Z-order*. The main benefit of this approach is the spatial coherence, leading to superior memory performance.

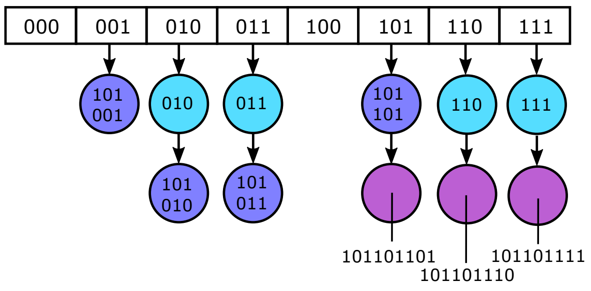


Figure 5.7: Example hashing of the linear octree of Figure 5.5.

potentially provides $O(1)$ access to its elements as long as there are no collisions in the hash table. In practice, this condition is difficult to fulfil and collisions do happen when the hash function generates the same index for different keys. Figure 5.7 shows an example for which the three least significant bits are used as index. Using linear hashed octrees, moving down or up the tree is a simple two-step operation which consists of deriving the code of the next node and looking-up the node in the hash-map. However, creating and deleting nodes at the top of hashed octrees can be very costly, as the location code of all nodes below the new root node gets 3 bits longer and must be updated. In some cases, the hash map must therefore be updated as well.

5.4.1.2 Our octree implementation

In our implementation we make use of the 8-base location code as depicted in Figure 5.5. However, we do not use a hash map as the overhead in case of expanding the octree is too expensive. Usually, the location codes are used for linear octrees in order to get rid of the pointers and to obtain a more compact representation. In true linear octrees, only the leaf nodes are stored.

As we will describe in section 5.4.4, we will use the inner nodes to store information, in order to have a multi-resolution representation of the map. For that reason, we integrate the location codes in a pointer-based representation to get the best of both worlds. The location codes facilitate more efficient queries for point location, region location and neighbour finding. In our implementation we use three integers for the x, y and z-axis of our axis-aligned bounding box (AABB). The location code or key k_n of node n can then also be generated from the depth d and the position (x, y, z) of its *center*. The key k_n is computed by interleaving the bits of x, y and z: if $x = x_1x_2 \dots x_M$, $y = y_1yx_2 \dots y_M$ and $z = z_1z_2 \dots z_M$, then $k_n = z_1y_1x_1z_2y_2x_2 \dots z_dy_dx_d$.

The pointer implementation on the other hand mitigates the creation and deletion of octree nodes. Especially the creation of nodes needs to be efficient as our goal is to build the 3D map incrementally. Furthermore, we support on-demand allocation of nodes, as the octree in 3D mapping use cases will generally be sparsely populated. The latter will save quite some memory compared to a pre-allocation in the form of a full octree.

The octree serves as the backbone of our map representation. The next question is what to store in the octree. Obviously, we gain little to no memory if we store all the acquired points. Therefore we will, in each octant, reconstruct the surface and only store the parameters of this surface and a centroid point in it. The next section will describe how we approximate this underlying surface using the moving least squares technique.

5.4.2 Moving Least squares surface

Moving least squares (MLS) is a method for approximating a continuous function from a set of unorganized point samples. Thereby, a weighted least squares measure is calculated for the region around the point at which the reconstructed value is needed. The term ‘moving’ comes from the fact that we change the weights ‘as we move’ on the surface. MLS thus leads to a ‘point set surface’ as the method creates a point function for each sample in the dataset.

In [Kolluri 08], the authors defined the implicit moving least squares (IMLS) surface as the set of zeros of the function $I(\mathbf{p}_i)$ given by eq. 5.1. The function serves as an approximate distance of any point $\mathbf{p}_i \in \mathbb{R}^3$ to the implicit surface defined by the 'map' point cloud $\mathcal{M}$:

$$I_{\mathcal{M}}(\mathbf{p}_i) = \frac{\sum_{\mathbf{p}_j} \theta(||\mathbf{p}_j - \mathbf{p}_i||)((\mathbf{p}_i - \mathbf{p}_j) \cdot \mathbf{n}_i)}{\sum_{\mathbf{p}_j} \theta(||\mathbf{p}_j - \mathbf{p}_i||)} \tag{5.1}$$

In this equation, $\mathbf{p}_j \in \mathcal{M}$ are the 'closest neighbours' of $\mathbf{p}_i$, $\mathbf{n}_i$ the normal vector of the point $\mathbf{p}_i$ and $\theta(x) = e^{-(\frac{x}{\sigma_r})^2}$ an exponential weighting function that is based on the Euclidean distance between $\mathbf{p}_j$ and $\mathbf{p}_i$ and the average separation σ_r of the 3D points. Thus, neighbouring points $\mathbf{p}_j \in \mathcal{M}$ located further away from $\mathbf{p}_i$ get a lower weight than points located closer.

As can be seen, the IMLS representation defined in eq. 5.1 is computed directly on the map point cloud $\mathcal{M}$ and can be used to conduct scan-to-map matching. When a point cloud $\mathcal{P}_k$, acquired in a new scan k, needs to be aligned with the map $\mathcal{M}$, the transformation $\mathbf{T} = (\mathbf{R}, \mathbf{t})$ can be estimated by minimizing the sum of squared IMLS distances:

$$\underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} \sum_{\mathbf{p}_i \in \mathcal{P}_k} |I_{\mathcal{M}}(\mathbf{R}\mathbf{p}_i + \mathbf{t})|^2. \tag{5.2}$$

This approach, however, has two major limitations. First, due to the exponential weights, this non-linear least-squares optimization problem can not be approximated by a linear least-squares one. Second, because the IMLS is computed directly from the raw point cloud, all of the points should be kept in memory. As such, we can end up with a too dense map as many points might be sampled in a small portion of the 3D space. This concentrated map impacts the performance of the system: the time and memory efficiency degrades while there is minimal improvement in accuracy.

The above means that this approach is not scalable for large environments or for long-term operation. For that reason, we propose to use the MLS surface in its explicit form and re-sample the original point cloud in order to keep fewer points. More specifically, we approximate the underlying surface around a point $\mathbf{p}_i$ by fitting a second degree polynomial p_i through the distances of its neighbouring points to its tangent plane H_i. The bivariate case of this quadratic function is given by $p_i(x, y) = ax^2 + by^2 + cxy + dx + ey + f$ and has 6 parameters. To obtain an estimate on these parameters we minimize the following objective function:

$$\tilde{p}_i = \underset{p_i}{\operatorname{argmin}} \sum_{\mathbf{p}_j} (p_i(\mathbf{x}_j) - f_j)^2 \theta(||\mathbf{p}_j - \mathbf{p}_i||), \tag{5.3}$$

in which $\mathbf{p}_j$ are once again the neighbours of $\mathbf{p}_i$, $\mathbf{x}_j$ the orthogonal projections of $\mathbf{p}_j$ onto the tangent plane $H_i \triangleq (\mathbf{n}_i, d_i)$ and $f_j \triangleq \langle \mathbf{p}_j, \mathbf{n}_i \rangle - d_i$ the distance of $\mathbf{p}_j$ to H_i. The normal vector $\mathbf{n}_i$ defining the tangent plane is computed as described in chapter 4. Finally, $\theta(x) = e^{-(\frac{x}{\sigma_r})^2}$ is the same weighting function as used in eq. 5.1.

5.4.3 Resampling

As it is infeasable to solve the minimization problem of eq. 5.2, we propose to resample the point cloud by projecting the original points onto the estimated surface. For each point $\mathbf{p}_i \in \mathcal{M}$ the polynomial in eq. 5.3 is estimated and the point is projected on its local surface as follows:

$$\mathbf{p}_i' = \mathbf{p}_i - \tilde{p}_i(\mathbf{x}_i)\mathbf{n}_i, \tag{5.4}$$

in which $\tilde{p}_i$ is the estimated polynomial and $\mathbf{x}_i$ the projection on the tangent plane of $\mathbf{p}_i$. This projection step will account for the noise resulting from inaccuracies in both the measurements and the pose estimates.

Right now, we are still dealing with the same number of points originally acquired by the sensor. As some regions are overly dense we subsample them in order to obtain a more uniform point density in the point cloud. To that end, we use the octree-based voxel grid and compute for each leaf node one point to store. More specifically, we compute the centroid point using all the points in the voxel, subsequently estimate the underlying (point set) surface according to eq. 5.3 and finally project the centroid point back on this surface. Eventually, we store the 6 parameters from the estimated polynomial together with the projected centroid in the node of the octree. The other points are removed in order to save memory.

Sometimes, the map contains small gaps because a region was not sampled at all, for example due to occlusion (so called *'lidar shadows'*). In order to make the map more complete and hence increase the robustness of the mapping, we conduct an additional *voxel dilation*. We, therefore, dilate empty voxels with a point located at their center position. Subsequently, we project these center points on the surface estimated at the closest point (in a neighbouring voxel) in the map point cloud.

The surface is reconstructed for each voxel or octant individually and is triggered each time new points are added to it. The beauty of this approach is that it lends itself perfectly for parallel computing, for example by means of distributed computing using several clusters or by using multiple GPUs. Each cluster is then responsible for computing the MLS surface for one voxel.

The surface reconstruction is thus applied several times and as such it is continuously refined. At some point, however, the gain of the refinement will be too minimal compared to the computational and memory resources that need to be spend. In addition, as the sensor is moving in the 3D space some voxels will not be updated anymore and therefore they could be removed from the RAM memory and stored on disk.

In order to perform scan-to-model matching we can now, instead of minimizing the IMLS cost function of eq. 5.2, use the same point-to-plane or plane-to-plane

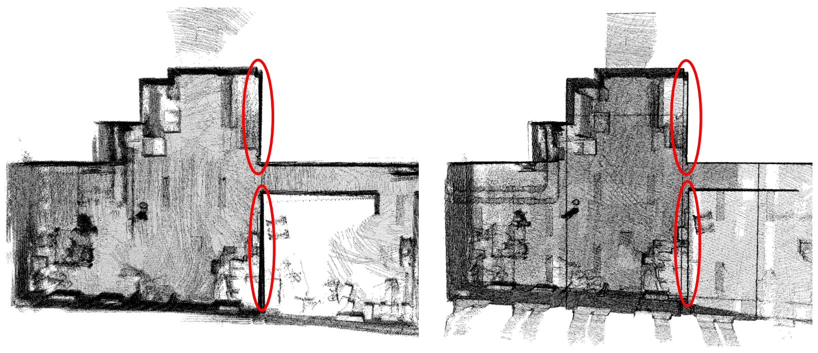


Figure 5.8: Top-down view of a point cloud before (left) and after (right) the projection on its underlying surface. Before projection, we can see that the measurement noise and imprecisions in the pose estimation cause flat areas such as the walls, to grow ‘perpendicular’ to the plane, hence creating a ‘thicker’ wall, as indicated by the red ellipses. After projection, the points belonging to the wall are lying on a thin surface.

cost function defined in eqs. 3.3 and 3.12, using the new ‘projected’ points in the map point cloud $\mathcal{M}$. Note that for the new points, the point normal is also updated based on the computed polynomial.

Figure 5.8 shows a top-down view of a point cloud captured in a lab. The left image shows the point cloud resulting from scan-to-scan matching, without applying a projection on the underlying surface. The right image shows the same point cloud resulting from scan-to-map matching with projection. We clearly see that the walls in the left point cloud are spread out over a few centimetres (indicated by the red ellipses) whereas the walls in the right point cloud are more ‘thin’. By performing re-sampling, small errors can thus be corrected and the ‘double walls’ artefacts can be smoothed.

5.4.4 Multi-resolution ICP

The surface reconstruction and resampling explained in the previous sections have three major benefits. First, as the map point cloud gets a more uniform point density, the pose estimation is not biased towards the regions that are more densely sampled. This in turn leads to a higher robustness of the system. Second, the noise due to inaccurate measurements and previous errors from the pose estimation is reduced, leading to more accurate 3D models. Third, the scan-to-map alignment can be conducted using fewer points leading to faster computation times. As most of the time of the ICP algorithm is spent on the computation of corresponding pairs, the largest speed-up can be obtained by lowering the number of points used to estimate the transformation. To facilitate this down-sampling, we exploit the octree-based voxel grid to select a point cloud $\mathcal{M}$ with a uniform point density, consisting of the projected centroids up to some level in the tree.

The main idea is to perform registration using different resolutions, going from the lower to the higher one, hence adopting a *coarse-to-fine* procedure. For every iteration, the result of the previous one serves as an initial guess and doing so we iteratively refine the transformation. In every iteration we conduct the main steps of the ICP algorithm, but in contrast to standard approaches, we change the resolution each time. In other words, based on the octree representation of the map, we *'filter'* the point cloud spatially and keep as it were a summarized version of it.

Each occupied octree cell at all levels in the tree contains the projected centroid as well as the parameters of the polynomial representing the surface in that point. It is important to note that the projected centroid point is computed on the fly and changes dynamically when more points are acquired within the same voxel. In order to limit memory usage, we use a threshold on the number of points that are being kept in one voxel. When this threshold is reached, the final projected centroid point is computed and the other points in the voxel are removed in order to free up memory space.

Apart from the target point cloud $\mathcal{M}$, the source point cloud $\mathcal{P}_s$ to be aligned can also be down-sampled to the same spatial resolution, using a similar octree-based voxel grid. As the point density of a single point cloud is low, the surface reconstruction can be omitted and the original (instead of projected) centroid point can be kept in each voxel.

Our hierarchical octree-based ICP method has a few major advantages. First, it helps finding nearest neighbours in a quick way. Second, since the transformation computed at a low resolution can serve as an initial guess for the transformation estimated at higher resolutions, it is not necessary to perform multiple iterations at the same level. This way we can lower the total number of iterations. These two advantages thus lead to a significant speed-up.

The environments that benefit the most from this approach are indoor scenes with large planar surfaces such as conference rooms or university campuses. For these environments, the octree cells at some point in the hierarchy will contain only points that are lying on a planar surface and can hence easily be summarized by its parametric form. Figure 5.9 depicts an example of a reconstructed 3D map at different resolution levels, going from a voxel size of 64 centimeters to 8 centimeters.

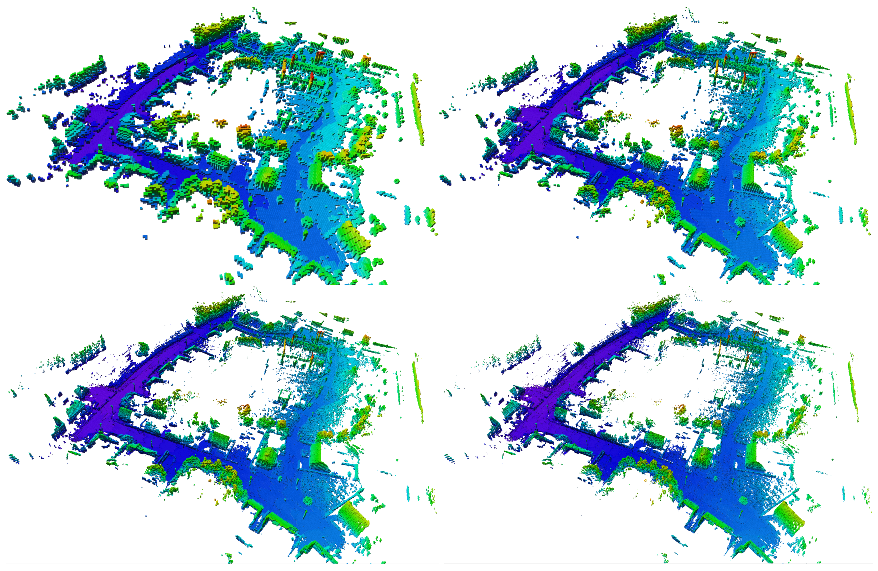


Figure 5.9: The same map depicted using different leaf resolutions. From left to right and top to bottom, the voxel size is respectively 64, 32, 16 and 8 centimeters.

Convergence and time complexity

The algorithm starts by using a low resolution representation of the map $\mathcal{M}$ and the source point cloud $\mathcal{P}_s$, obtained by considering all *nodes* down to the level with a certain voxel size. It can start for example with a voxel size of 64 centimeter and end up with a voxel size of 1 centimeter. In every iteration the closest points between the two point clouds are determined. Thanks to the octree representation of the map, finding the (approximate) closest points between $\mathcal{P}_s$ and $\mathcal{M}$ is an extremely fast operation. For each point in $\mathcal{P}_s$ we just have to traverse the tree until the voxel at depth d has been reached. The approximated closest point is then the centroid point projected on the reconstructed surface. This operation takes $\mathcal{O}(d)$ time as it is not depending on the number of points that have been acquired or that are currently stored in the map.

Important to note is that we only consider *mutual* closest point pairs, i.e. we consider two points matching as both the 'source' point is the closest one to the 'target' point in the map and vice versa. It is thus not possible for two points in the source point cloud to be matched with the same point in the map. The total time complexity for finding corresponding points at resolution r is therefore given by $\mathcal{O}(N_{s,r})$, $N_{s,r}$ representing the number of points in the source point cloud at resolution r. The computation time for the scan-to-map alignment is thus only dependent on the resolution level and the size of the bounding box, i.e. the

octree level	leaf size (cm)	avg. residual (cm)	t (ms)	acc. t (ms)
initial		3.1907		
11	32	0.6309	5	5
12	16	0.3027	11	16
13	8	0.2799	16	32
14	4	0.2725	32	64
15	2	0.2719	51	115
16	1	0.2716	77	192
non-MR		**0.2730**		**7212**

Table 5.1: Results of the octree-based alignment process. The term residual denotes the final cost of the ICP procedure after convergence, i.e. the point-to-plane distance of all corresponding point pairs. The time t denotes the time that is needed to compute the transformation using the respective resolution and 'acc. t' denotes the accumulated time. Our multi-resolution algorithm reduces the total processing time from 7212 ms to 64 ms (to reach the same accuracy).

dimensions of the map, and not on the total number of acquired points.

It can still happen that some of the approximate closest points are bad corresponding pairs and are hence acting as *outliers*. To cope with this, we can adopt the same strategy as proposed in chapter 4 using the low-level shape features to assign a weight for each corresponding pair based on the distance in feature space. By using the Beaton Tukey robust M-estimator from eq. 4.13, outlier correspondences are entirely rejected. The downside of this weighting is that the shape features have to be computed, which is a time consuming operation and not worth to spend the resources in some cases. Using our multi-resolution technique, it is in any case no longer needed to continue to iterate until convergence for each resolution. In fact, only one iteration seems to be sufficient for the intermediate resolutions. Only at the highest resolution the iterations can continue until convergence, but even there one iteration suffices most of the time.

To demonstrate this convergence behaviour, we conducted an experiment using data that was captured with a Velodyne HDL-32e lidar scanner at a campus of Ghent University. The experiment was executed on a laptop computer with an Intel Core i7-4712HQ, 2.30Ghz CPU inside and 16GB RAM. The 4 cores of the CPU were used simultaneously through OpenMP for several parts of the algorithm, including the correspondence estimation. In total, 14 sequences were evaluated, all of them containing 30 seconds to 3 minutes of lidar data. Table 5.1 summarizes the processing times as well as the average residuals for the different resolution levels. The term residual denotes the final 'cost' of the ICP procedure after convergence, i.e. the Euclidean distance of all corresponding point pairs. The results are obtained using the standard point-to-plane cost function without any weighting of correspondences.

As can be seen, the process converges quite quickly to the optimal residual. Between the iterations at resolution level 12 and 13, the total gain is less than one millimeter. For some applications, e.g. map building for navigation in autonomous robots, the level of accuracy achieved at resolution level 13 is sufficient and can be achieved in ± 31 ms. Even if we pursue the highest possible accuracy we can conclude that our multi-resolution approach is beneficial in terms of processing speed. Processing at resolution levels higher than 16 no longer leads to a decrease of the residual. In fact, when the processing stops at resolution level 14, more or less the same residual as for a non-multi-resolution approach is achieved. The difference in processing time is, however, tremendous, reducing from more than 7 s to only 64 ms.

5.5 Escaping from local minima

One of the main drawbacks of the standard ICP method is that it is still prone to get stuck in local minima. This problem gets even more severe when the difference between the two point clouds is large. Recall that in case of scan matching we consider point clouds acquired from consecutive sweeps or frames. However, in case there is an abrupt movement of the sensor, or in case the (angular) velocity of the moving sensor is too high, the probability that ICP got stuck in a local, wrong, minimum is high. For that reason, we propose to incorporate the ICP algorithm into a Branch-and-Bound (BnB) registration scheme that explores a part of the 3D motion space, similar to the work of [Yang 13].

More specifically, based on the structure of the underlying geometry we define upper and lower bounds for the ICP residual, which alleviates the choice for a good initialization, hence leading to efficient transformation computations. The algorithm can be summarized as follows. Starting from an initialization, which is based on the transformation estimation of the previous point cloud, conduct ICP to find the minimal residual. Use this residual as an upper bound and subsequently search the motion space for a better solution, until a good solution is found. A graphical representation of the algorithm is depicted in Figure 5.10. Now, two important questions remain unanswered: 1) how do we parametrize and branch the domain of 3D motions and 2) how to efficiently find an upper and lower bound. The answers are given in the following sections.

The BnB algorithm is obviously very time-consuming as the ICP algorithm is run several times. As such, it is not a part of our real-time SLAM system. We used the technique as a correction method when we noticed that for a part of a lidar

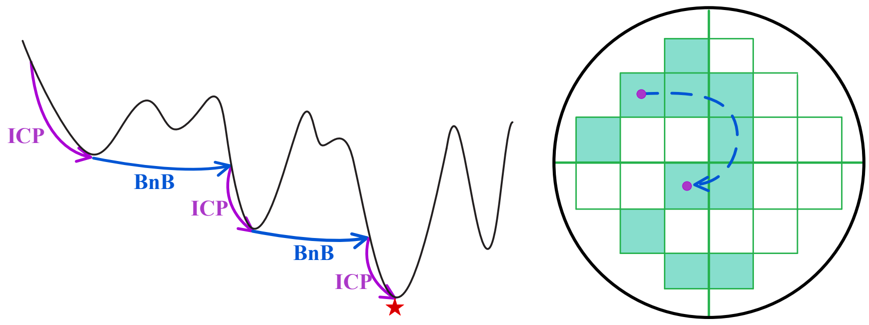


Figure 5.10: Graphical representation of the Branch-and-Bound registration scheme integrating the ICP algorithm, as presented in [Yang 13].

sequence our ICP-based scan matching was making a severe error due to an abrupt movement. In that case, the BnB helps to escape from the local minimum and to prevent that a severe error is made.

5.5.1 Domain parametrization

Like in the previous section, we use the angle-axis representation to encode rotation and we use $\mathbf{R}_\mathbf{r}$ to denote the rotation matrix with angle-axis representation $\mathbf{r}$. The entire space defined by all 3D rotations can then be represented as a solid radius-π ball in 3D. To ease the search, we use the minimum cube $[-\pi, \pi]^3$ that encloses the π-ball as the rotation domain. Regarding the translation, we use a bounded cube in which we assume that the optimal translation is lying. During the BnB process, we use the octree datastructure to subdivide the initial cubes into smaller sub-cubes C_r and C_t.

5.5.2 Uncertainty radii

The bounding functions are based on the concept of *uncertainty radius*. This uncertainty radius is defined as the region in which a 3D point $\mathbf{p}$ can be transformed if it undergoes a 3D rigid motion with rotation $\mathbf{r} \in C_r$ and/or translation $\mathbf{t} \in C_t$. Using Figure 5.11 we derive an upper bound for the rotation uncertainty radius as follows. Given the sine rule, we find the expression:

$$||\mathbf{R}_\mathbf{r}\mathbf{p} - \mathbf{R}_{\mathbf{r}_0}\mathbf{p}|| = \frac{\sin(\alpha)}{\sin(\frac{1}{2}(\pi - \alpha))}||\mathbf{p}||, \tag{5.5}$$

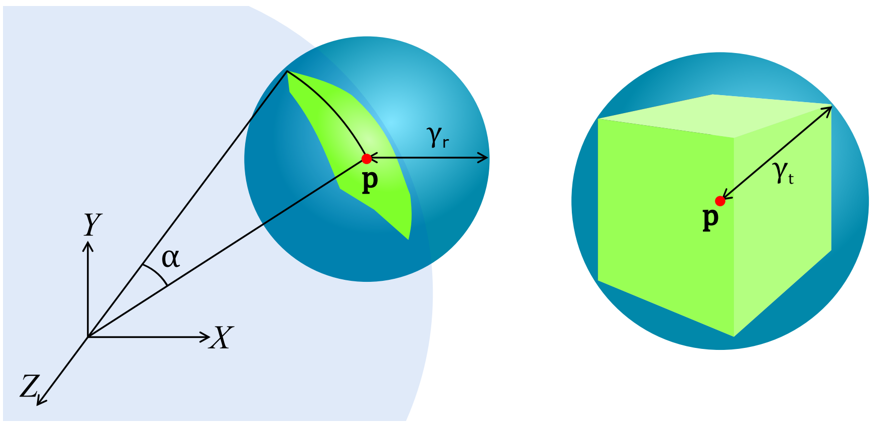


Figure 5.11: Uncertainty radii at a point, as presented in [Yang 13]. Left: rotation uncertainty ball for C_r (in blue) with center $\mathbf{R}_{\mathbf{r}_0}\mathbf{x}$ (red dot) and radius γ_r. Right: translation uncertainty ball for C_t (in blue) with center $\mathbf{x} + \mathbf{t}_0$ (red dot) and radius γ_t. In both diagrams, the uncertainty balls enclose the range of $\mathbf{R}_\mathbf{r}\mathbf{x}$ or $\mathbf{x} + \mathbf{t}$ (in green).

in which $\alpha = \angle(\mathbf{R}_\mathbf{r}\mathbf{p}, \mathbf{R}_{\mathbf{r}_0}\mathbf{p})$. Using the double angles formula in the numerator and the difference formula in the denominator we obtain:

$$||\mathbf{R}_\mathbf{r}\mathbf{p} - \mathbf{R}_{\mathbf{r}_0}\mathbf{p}|| = \frac{2\sin(\frac{\alpha}{2})\cos(\frac{\alpha}{2})}{\sin(\frac{\pi}{2})\cos(\frac{\alpha}{2}) - \cos(\frac{\pi}{2})\sin(\frac{\alpha}{2})}||\mathbf{p}||, \tag{5.6}$$

which simplifies to:

$$||\mathbf{R}_\mathbf{r}\mathbf{p} - \mathbf{R}_{\mathbf{r}_0}\mathbf{p}|| = 2\sin(\frac{\alpha}{2})||\mathbf{p}||. \tag{5.7}$$

Now, we use the lemma of paper [Hartley 09]:

$$\angle(\mathbf{R}_\mathbf{r}\mathbf{p}, \mathbf{R}_{\mathbf{r}_0}\mathbf{p}) \leqslant \angle(\mathbf{R}_\mathbf{r}, \mathbf{R}_{\mathbf{r}_0}) \leqslant ||\mathbf{r} - \mathbf{r}_0||, \tag{5.8}$$

which states that the angular distance between two rotations in the underlying manifold is less than their vector distance in the angle-axis representation.

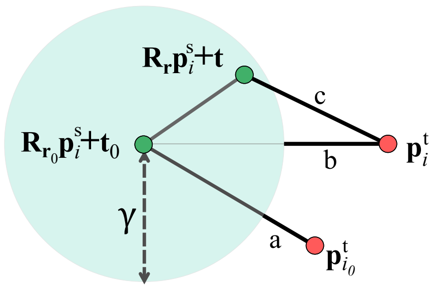
Using this lemma, we can derive the following inequality:

$$||\mathbf{R}_\mathbf{r}\mathbf{p} - \mathbf{R}_{\mathbf{r}_0}\mathbf{p}|| \leqslant 2\sin(\min(\frac{\sqrt{3}\sigma_r}{2}, \frac{\pi}{2}))||\mathbf{p}|| \doteq \gamma_r \tag{5.9}$$

in which we call γ_r the uncertainty radius and σ_r the half side-length of the rotation cube C_r. We can also derive a translation uncertainty radius γ_t for a translation cube C_t with half side-length σ_t centered at $\mathbf{t}_0$:

$$||(\mathbf{x} + \mathbf{t}) - (\mathbf{x} + \mathbf{t}_0)|| = ||\mathbf{t} - \mathbf{t}_0|| \leqslant \sqrt{3}\sigma_t \doteq \gamma_t. \tag{5.10}$$

Figure 5.11 shows the geometrical representation of these uncertainty radii γ_r and γ_t. They are both used to subsequently derive upper and lower bounds for the ICP residual.



$\mathbf{R_r p}_i^s\mathbf{+t}$
c
$\mathbf{R_{r_0} p}_i^s\mathbf{+t}_0$
b
$\mathbf{p}_i^t$
γ
a
$\mathbf{p}_{i_0}^t$

Figure 5.12: Graphical explanation for the derived lower bound, as presented in [Yang 13]. It can be seen that $a \leq b \leq c$ while $a = \underline{e_i}$ and $c = e_i(\mathbf{R_r}, \mathbf{t})$.

5.5.3 Bounding functions

An upper bound for the per-point L_2 residual error $e_i(\mathbf{R}, \mathbf{t}) = ||\mathbf{R}\mathbf{p}_i^s + \mathbf{t} - \mathbf{p}_i^t||$ within a given rotation cube C_r centered at $\mathbf{r}_0$ can be trivially found as:

$$\overline{e_i} \doteq e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}_0) \geqslant \min_{\forall(\mathbf{r},\mathbf{t})\in(C_r \times C_t)} e_i(\mathbf{R_r}, \mathbf{t}). \tag{5.11}$$

A suitable lower bound for this residual error appears to be a harder task. However, using the concepts of the uncertainty radii, we can derive the following. Like before, we consider the point $\mathbf{p}_i^t \in \mathcal{P}^t$ to be the closest to $\mathbf{R_r}\mathbf{p}_i^s + \mathbf{t}$, $\mathbf{p}_i^s \in \mathcal{P}^s$, i.e. the points in $\mathcal{P}^s$ and $\mathcal{P}^t$ are again indexed according to their closest points. Now, let $\mathbf{p}_{i_0}^t$ be the closest point to $\mathbf{R}_{\mathbf{r}_0}\mathbf{p}_i^s + \mathbf{t}_0$. Then, $\forall(\mathbf{r}, \mathbf{t}) \in (C_r \times C_t)$:

$$\begin{aligned}
e_i(\mathbf{R_r}, \mathbf{t}) &= ||\mathbf{R}\mathbf{p}_i^s + \mathbf{t} - \mathbf{p}_i^t|| && (5.12)\\
&= ||(\mathbf{R}_{\mathbf{r}_0}\mathbf{p}_i^s + \mathbf{t}_0 - \mathbf{p}_i^t) + (\mathbf{R_r}\mathbf{p}_i^s - \mathbf{R}_{\mathbf{r}_0}\mathbf{p}_i^s) + (\mathbf{t} - \mathbf{t}_0)|| && (5.13)\\
&\geqslant ||\mathbf{R}_{\mathbf{r}_0}\mathbf{p}_i^s + \mathbf{t}_0 - \mathbf{p}_i^t|| - (||\mathbf{R_r}\mathbf{p}_i^s - \mathbf{R}_{\mathbf{r}_0}\mathbf{p}_i^s|| + ||\mathbf{t} - \mathbf{t}_0||) && (5.14)\\
&\geqslant ||\mathbf{R}_{\mathbf{r}_0}\mathbf{p}_i^s + \mathbf{t}_0 - \mathbf{p}_i^t|| - (\gamma_{r_i} + \gamma_t) && (5.15)\\
&\geqslant ||\mathbf{R}_{\mathbf{r}_0}\mathbf{p}_i^s + \mathbf{t}_0 - \mathbf{p}_{i_0}^t|| - (\gamma_{r_i} + \gamma_t) && (5.16)\\
&= e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}_0) - (\gamma_{r_i} + \gamma_t), && (5.17)
\end{aligned}$$

where eq. 5.14 results from the reverse triangle inequality, eq. 5.15 from the definition of the uncertainty radii and 5.17 from the closest-point mechanism.

Now we can derive the lower bound of the per-point residual for $C_r \times C_t$ as:

$$\underline{e_i} \doteq \max(e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}_0) - (\gamma_{r_i} + \gamma_t), 0) \leqslant \min_{\forall(\mathbf{r},\mathbf{t})\in(C_r \times C_t)} e_i(\mathbf{R_r}, \mathbf{t}). \tag{5.18}$$

Summing up the squared upper and lower bounds of the per-point residuals of eqs. 5.11 and 5.18, we arrive at the following bounds for the L_2 error:

$$\overline{E} \doteq \sum_{i=1}^{N} \overline{e_i}^2 = \sum_{i=1}^{N} e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}_0)^2, \tag{5.19}$$

$$\underline{E} \doteq \sum_{i=1}^{N} \underline{e_i}^2 = \sum_{i=1}^{N} \max(e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}_0) - (\gamma_{r_i} + \gamma_t), 0)^2, \tag{5.20}$$

for a 3D motion domain $C_r \times C_t$ centered at $\mathbf{r}_0, \mathbf{t}_0$ with uncertainty radii γ_{r_i} and γ_t.

5.5.4 Branch and bound algorithm

One single BnB loop would be quite complex and inefficient to search for the motion space. Therefore, it is more appropriate to adopt a nested branch and bound scheme in which the rotation and translation part are separated. The rotation part is integrated in an outer BnB, whereas the translation part is incorporated in an inner BnB. In the outer rotation BnB, for a cube C_r the bounds of the registration error can be set as $\overline{E}_r = \min_{\forall \mathbf{t} \in \mathcal{C}_t} \sum_i e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t})^2$ and $\underline{E}_r = \min_{\forall \mathbf{t} \in \mathcal{C}_t} \sum_i \max(e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}) - \gamma_{r_i}, 0)^2$ where $\mathcal{C}_t$ is the initial translation cube. To solve $\underline{E}_r$ with the inner translation BnB, the bounds for a translation cube C_t can be chosen as $\overline{E}_t = \sum_i \max(e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}_0 - \gamma_{r_i}), 0)^2$ and $\underline{E}_t = \sum_i \max(e_i(\mathbf{R}_{\mathbf{r}_0}, \mathbf{t}_0) - (\gamma_{r_i} + \gamma_t), 0)^2$ in which we set the uncertainty radii γ_{r_i} to zero.

5.6 Evaluation

We evaluated the accuracy of our ‘final’ scan matching algorithm on the same datasets used in chapter 3. We again make the distinction between indoor, man-made environments with strong ‘Manhattan world’ characteristics and outdoor scenes. For the former environments, we use the same eight sequences that were recorded inside buildings on different campuses of Ghent University. For the outdoor sequences, we use the KITTI benchmark as well as our own recorded dataset in the Belgian city Hasselt. The main difference between the two datasets is the fact that the lidar scanner was tilted for the latter one, in view of capturing the entire area in high detail, including the facades and rooftops of the houses.

5.6.1 Indoor environments

The specifications of the eight different sequences are listed in Table 3.2. We are evaluating the same four scan matching techniques described in chapter 3, but this

Sequence	NDT	ICP	GICP	**S-GICP**
Corridor	2.0	0.5	0.3	**0.3**
Corridor tilt	1.5	1.7	0.9	**0.9**
Lab	3.3	2.3	1.1	**0.7**
Lab tilt	4.4	2.5	0.6	**0.4**
Lab extended	×	4.7	1.3	**0.8**
UFO foyer	8.1	8.8	**0.7**	1.8
UFO entrance	5.3	26.0	1.3	**0.9**
Schoonmeersen	×	×	102.3	**52.5**

Table 5.2: The error in centimeter on the indoor sequences acquired by a Velodyne VLP-16 for four different scan matching techniques combined with scan-to-map alignment. The error was computed by calculating the Euclidean distance between the start and end position after creating a loop. The trajectories resulting from our own S-GICP method are the least subject to drift. Note that these results, integrating scan-to-model matching, tremendously improve the results presented in Table 3.3.

time in combination with scan-to-model matching: NDT, ICP, GICP and S-GICP. Recall that all of the sequences were recorded in such a way that a loop was created at the end. We consider the Euclidean distance between the first and final estimated position as the error. This Euclidean distance is determined using the translational components of the pose matrices. Regarding the scan-to-map alignment, we picked a voxel size of 10 cm for the leaf nodes. The results are presented in Table 5.2.

Compared to the results presented in Table 3.3, the performance of our final registration algorithm - combining scan-to-scan and scan-to-map alignment - is tremendously improved. For almost all of the sequences (1-7), the error at the end of the sequence is below 1 centimeter. Especially for the longer sequences 'Lab extended' or 'UFO entrance' the error reduction is significant, dropping from respectively 51.7 and 31.3 cm to 0.8 and 0.9 cm. Given the total length of these sequences, 52.98 m and 113.72 m, the translational error is respectively 0.015 % and 0.008 % which is negligible. The experiments show that our scan-to-map alignment is able to prevent the system from heavily drifting.

Figure 5.13 depicts a zoomed in image of the result we obtain for the sequence 'Lab tilt' using our final scan matching algorithm. One can see that the trajectory results in an almost perfect loop and that little to no artefacts are visible in the point cloud. Table 5.2 also affirms the conclusions from chapter 3 that our own S-GICP cost function is the best performing for indoor, man-made environments. The other techniques NDT, ICP, and GICP also benefit from the integration with scan-to-map alignment, but none of them reach the accuracy of our S-GICP method.

The sequence 'Schoonmeersen' remains challenging. Using the NDT or stan-

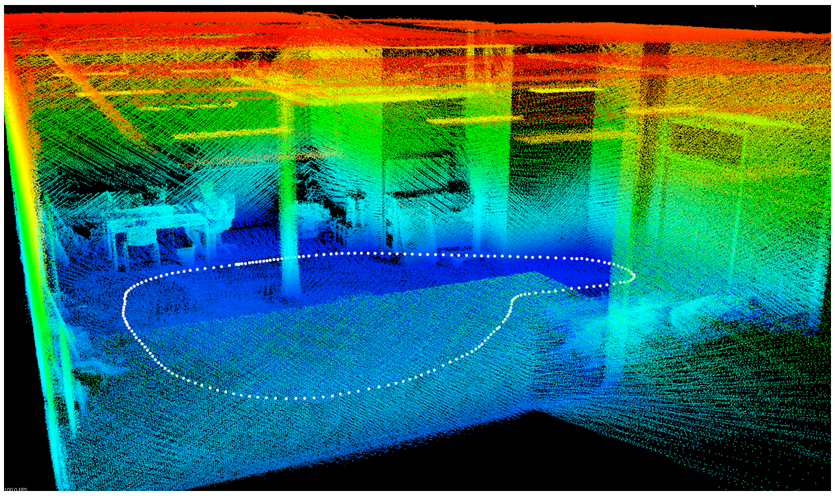


Figure 5.13: Reconstructed point cloud and trajectory from the sequence 'LAB TILT', zoomed in. The sequence was recorded with our Liborg robot equipped with a Velodyne VLP-16 tilted in order to capture the overhanging structures.

dard point-to-plane ICP method, the drift - even with the scan-to-map alignment - remains too large to obtain a meaningful reconstruction. Regarding our own S-GICP method, the error is reduced from 212.7 centimeter to 52.5 centimeter, hence leading to an improvement with a factor of approximately 4. The main reason for this poor result is the abrupt motion of the lidar scanner when ascending or descending a staircase. The errors made in those particular regions of the trajectory contribute for the larger part to the total drift. It is thus a matter of robustness rather than accuracy as for the rest of the sequence the scan matching estimates the poses with sufficient accuracy.

5.6.2 Outdoor environments

In order to test our final registration method on outdoor sequences, we use both the KITTI benchmark as well as our own dataset captured in Hasselt. The latter dataset was recorded with a tilted lidar scanner and is therefore a lot more challenging. As in chapter 4, we will use our W-GSS-ICP method instead of our S-GICP as it has been proven more successful for outdoor datasets. To recap, W-GSS-ICP consists of a geometrical stable point sampling part, to make sure that the majority of the points are taken from stable, planar, areas. In addition, it prevents directional biases due to the non-uniform point sampling by picking points in such a way that the observability in all directions is proportioned. Finally, the point correspondences are weighted based on the mutual 'shape' similarity of the two points.

Sequence	Environment type	**LOAM**	**OUR**
KITTI 00	Urban	0.78	1.03
KITTI 01	Highway	1.43	2.80
KITTI 02	Urban + Country	0.92	1.15
KITTI 03	Country	0.86	**0.68**
KITTI 04	Country	0.71	**0.14**
KITTI 05	Urban	0.57	**0.56**
KITTI 06	Urban	0.65	**0.61**
KITTI 07	Urban	0.63	**0.41**
KITTI 08	Urban + Country	1.12	1.16
KITTI 09	Urban + Country	0.77	1.06
KITTI 10	Urban + Country	0.79	1.05
AVG		**0.84**	**0.97**

Table 5.3: The translational (E_t) in $\%$ on the KITTI sequences for our map-based tracking algorithm compared to the best state-of-the-art method LOAM presented in [Zhang 14].

5.6.2.1 KITTI dataset

Just like in the previous chapters we will use the RPE error defined in eqs. 2.13 and 2.14 to evaluate the performance. The translational errors E_t in $\%$ for the test sequences of the KITTI dataset are presented in 5.3. The translational errors are again computed as the average for different distance intervals ranging from 100 m to 800 m. The results are compared with the state-of-the-art method presented in [Zhang 14], named ‘lidar odometry and mapping’ (LOAM), which is currently still listed as the best performing lidar-based SLAM method on the KITTI benchmark.

The experiments show that we are able to further reduce the error from 1.09$\%$ using our W-GSS-ICP scan matching to 0.97$\%$ when we combine it with scan-to-model matching. We also notice that for some sequences we obtain better results than the LOAM method. The sequences ‘05’, ‘06’ and ‘07’ are recorded in urban environments, where our method greatly benefits from the large planar structures that are present. Also for the sequences ‘03’ and ‘04’, which are classified as ‘Country’, our method is more accurate. For the sequences classified as ‘Urban + Country’ our method performs only slightly worse.

The worst result is obtained for the sequence ‘01’, which was recorded on a highway. As mentioned in chapter 4, the difficulties for this sequence are two-fold. First, the areas next to the highway are dominated by vegetation for which it is difficult to accurately estimate surface normals. Second, a lot of other cars are driving at approximately the same speed causing the pose estimation to be biased

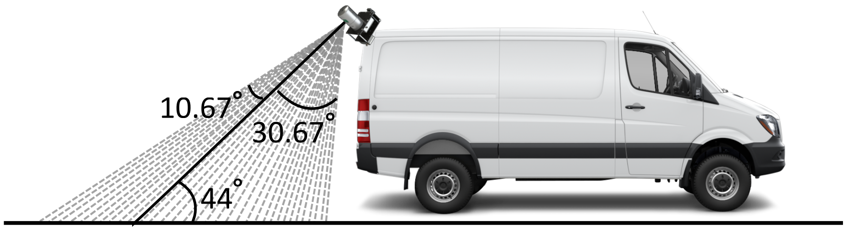


Figure 5.14: Schematic drawing of the acquisition platform, consisting of a velodyne HDL-32e lidar scanner that was mounted on a mobile mapping van of the company Grontmij.

towards the identity transformation. Often times, the cars are also driving close by which causes a lot of points to be sampled on them instead of on the static environment. If we only consider sequences recorded in urban and countryside areas, we obtain the same average error as LOAM being 0.78%.

5.6.2.2 Hasselt survey

The main ‘shortcoming’ of the KITTI dataset is that all sequences are recorded with a horizontally mounted scanner as its main purpose is to test algorithms in view of autonomous driving. For those applications, it makes sense to mount the scanner horizontally as it maximizes the (horizontal) field of view, which facilitates the detection of road users. We, on the other hand, want to test our algorithms on data that is captured with a tilted scanner as well. The latter can be beneficial in those cases where we want to map the entire area in high detail, including the facades and rooftops of the houses. For that reason, we recorded our own data by mounting a Velodyne HDL-32E lidar scanner on a mobile mapping van of the Grontmij company (later acquired by Sweco). As depicted in Figure 5.14, the sensor was tilted, making an angle of approximately 44° with respect to the ground plane.

The Velodyne lidar scanner has 32 collinear lasers, which cover a vertical FOV of 41.3° hence resulting in a vertical resolution of 1.29°. This vertical FOV covers 30.67° below the middlepoint and 10.67° above it. Just like the other Velodyne scanners, the head is continuously spinning at approximately 10 Hz covering a horizontal FOV of 360°. A picture of the starting point - taken from Google Streetview - is depicted in Figure 5.15. The corresponding point cloud - acquired during the first sweep of the scanner - is depicted in Figure 5.16.

The exact longitude and latitude of this starting point was respectively $5.30935°$ and $50.9416°$. The mobile mapping van was driving at an average speed of ap-

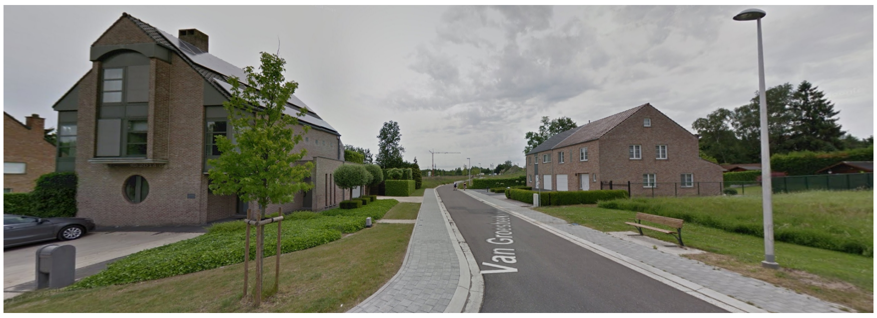


Figure 5.15: The starting point of the sequence recorded in Hasselt, taken from Google Street View.

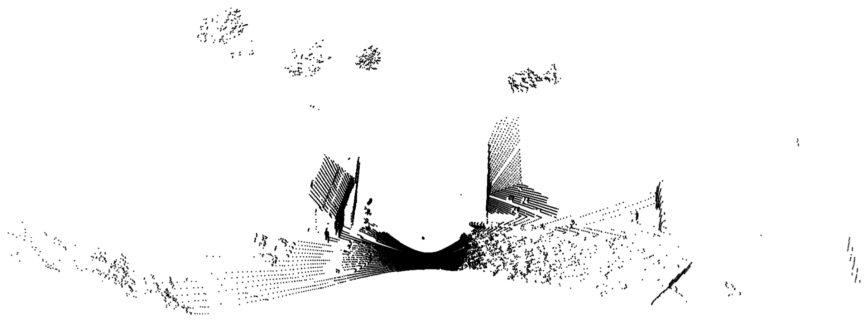


Figure 5.16: The point cloud captured at the starting point, acquired with the set-up depicted in Figure 5.14. Due to the tilted scanner, the 'horizontal' field of view is limited. On the other hand, the facades and rooftops of the houses are richly sampled.

proximately 25 km/h and stopped at the exact same position from where it has started, hence creating a loop. Apart from data captured with the Velodyne scanner, we also recorded accurate positioning information using the POS LV 420 sensor developed by Applanix. This INS system incorporates both an inertial measurement unit (IMU), a distance measurement indicator (DMI) and Trimble's BD960 GNSS receiver. The positioning system is, therefore, highly accurate and we will use it in the experiments as the *ground truth*. Since each point of the Velodyne data is timestamped we can derive its precise geolocation. We used the Belgian Lambert72 geographic information system (GIS) to express this geolocation.

To be able to evaluate our outcome against the ground truth we transformed both the ground truth trajectory $\mathbf{P} = \{\mathbf{P}_1, \ldots, \mathbf{P}_k, \ldots, \mathbf{P}_N\}$ and estimated trajectory $\hat{\mathbf{P}} = \{\hat{\mathbf{P}}_1, \ldots, \hat{\mathbf{P}}_k, \ldots, \hat{\mathbf{P}}_N\}$ into the same reference system. We set the origin of the first laser frame as global origin. As such we define an initial transformation $\mathbf{T}_0$ as follows:

$$\mathbf{T}_0 = \left[\begin{array}{c:c} \mathbf{R}_0 & \mathbf{t}_0 \\ \hdashline \mathbf{0}_3^\top & 1 \end{array}\right] = \mathbf{T}_0^{(las2geo)} = \mathbf{T}_0^{(veh2geo)}\mathbf{T}^{(las2veh)}. \tag{5.21}$$

The matrix $\mathbf{T}^{(las2veh)}$ represents the transformation from the laser frame to the vehicle frame and is obtained by accurate calibration. The vector of transformation matrices $\mathbf{T}^{(veh2geo)} = \{\mathbf{T}_0^{(veh2geo)}, \ldots, \mathbf{T}_k^{(veh2geo)}, \ldots, \mathbf{T}_N^{(veh2geo)}\}$ is derived from the POS LV positioning system at the timestamp of the first acquired point $\mathbf{p}_k^0 \in \mathcal{P}_k$ acquired during sweep k. The first pose of the ground truth trajectory is thus defined as follows:

$$\mathbf{P}_1 = \mathbf{T}_1^{(veh2geo)}\mathbf{T}_0^{-1}. \tag{5.22}$$

The rest of the ground truth trajectory is subsequently given by:

$$\mathbf{P}_k = \left[\begin{array}{c:c} \mathbf{R}_k & \mathbf{t}_k \\ \hdashline \mathbf{0}_3^\top & 1 \end{array}\right] = \mathbf{P}_k^{(veh2geo)}\mathbf{P}_{k-1}. \tag{5.23}$$

Similarly, for the estimated trajectory, we put each sensor frame in the same reference system by transforming it using the same rotation $\mathbf{R}_0$. We thus define the initial estimated pose matrix $\hat{\mathbf{P}}_0$ as:

$$\hat{\mathbf{P}}_0 = \left[\begin{array}{c:c} \mathbf{R}_0 & \mathbf{0}_3 \\ \hdashline \mathbf{0}_3^\top & 1 \end{array}\right]. \tag{5.24}$$

The estimated trajectory is then given by:

$$\hat{\mathbf{P}}_k = \left[\begin{array}{c:c} \hat{\mathbf{R}}_k & \hat{\mathbf{t}}_k \\ \hdashline \mathbf{0}_3^\top & 1 \end{array}\right] = \hat{\mathbf{P}}_k\hat{\mathbf{P}}_{k-1}. \tag{5.25}$$

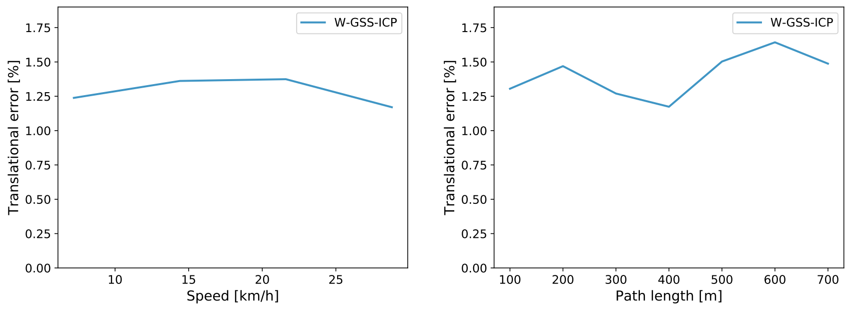


Figure 5.17: The translational errors in function of the speed (left) and trajectory length (right), for the Hasselt survey. The error for speeds ranging from 2 to 30 km/h is quite stable and is situated near 1.3%. The error for varying trajectory lengths is also stable and averages around 1.4%.

The matrix $\hat{\mathbf{P}}_k$ represents the outcome of our pose estimation. Yet, in order to be able to evaluate our system quantitatively, we projected the ground truth and estimated position of the vehicle onto the ground plane and derived the translational error e_t and rotational error e_r from it. Since we have set the first sensor frame as the origin, the ground truth and estimated position of the vehicle at the start of sweep k is given by respectively $\mathbf{t}_k$ and $\hat{\mathbf{t}}_k$. Recall that our original sensor was tilted by $44°$, but the rotation $\mathbf{R}_0$ aligned the ground plane with the xz-plane in the final reference system.

Just like for the KITTI sequences, we used the point-to-plane cost function of eq. 3.3 in combination with geometric stable sampling and feature-based weighting as presented in chapter 4, referred to as the W-GSS-ICP method. In Figure 5.17, two graphs are depicted showing the translational errors (in $\%$) for varying speeds and trajectory lengths. The errors for speeds ranging from 2 to 30 km/h remain very stable and average out around 1.3%. Also for different trajectory lengths of 100 m to 700 m the error is quite stable and averages out around 1.4%. Note that these results are only slightly worse than the ones we obtained for the KITTI dataset. We can thus conclude that even for challenging sequences with a limited ‘horizontal’ FOV, our method is able to generate accurate results.

In Figure 5.18, the trajectories are shown for standard point-to-point and point-to-plane scan matching (left) as well as for our final scan matching algorithm comprising both scan-to-scan and scan-to-map registration (right). The pose estimation of the standard point-to-point ICP method is very poor and a meaningful trajectory can hardly be obtained. The problems related to this failure are due to the challenges and shortcomings described in chapter 3. The standard point-to-plane method performs a lot better, but we see that it still suffers from the inhomogeneous point sampling and the lack of strong shape characteristics in the scene. Our own

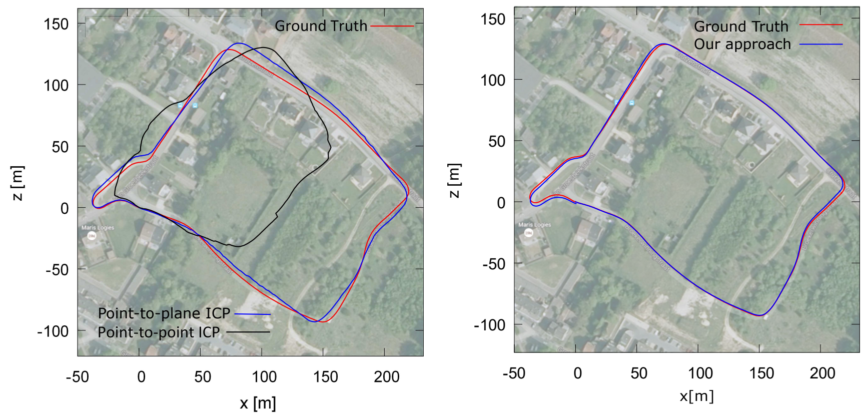


Figure 5.18: The resulting trajectory of the Hasselt survey for the standard point-to-point and point-to-plane cost function (left), and our final registration combining scan-to-scan and scan-to-map registration (right).

method (right in Figure 5.18) on the other hand is performing very well, indicated by the almost entire overlap of the estimated trajectory with the ground truth trajectory. At the end, the difference between the start and estimated end position equals 1.38 meter, which can be considered as the error. Note that this estimated trajectory is the outcome before loop closure. As we will see in chapter 6, we can further improve this result by exploiting the fact that the van was riding in a closed loop and by correcting the error at the end of this loop.

5.7 Conclusion

In this chapter we extended the scan-to-scan matching from chapters 3 and 4 with scan-to-model matching. The latter is facilitated by a surface reconstruction and resampling algorithm that guarantees that the point cloud remains bounded in size. We evaluated our approach on both indoor and outdoor datasets.

Regarding the indoor datasets we used the same Velodyne VLP-16 sequences as in chapter 3. They were recorded in such a way that a loop was created and the error is evaluated by comparing the start- and end position of the sequence. The dataset included both sequences recorded with a horizontally mounted scanner as well as a tilted one. We evaluated the same four scan matching techniques as in chapter 3: NDT, ICP, GICP and S-GICP. We integrated them in a registration algorithm comprising both scan-to-scan and scan-to-model alignment, the latter including our surface reconstruction and resampling technique. The experiments demonstrated that our S-GICP method lead to the best results for almost all of the sequences.

Moreover, the errors for the longer sequences dropped from tens of centimeters (in case of only applying scan-to-scan matching) to less than 1 centimeter (in case of scan-to-scan and scan-to-model matching) for trajectories of more than 100 m, hence resulting in a translational error of as low as 0.01%.

To evaluate the performance on outdoor scenes, we used the KITTI dataset as well as our own 'Hasselt' dataset recorded with a tilted lidar scanner. On the former, we are able to reduce the translational error from 1.09% (scan-to-scan alignment) to 0.97% (scan-to-scan and scan-to-model alignment). Considering only those sequences that are recorded in urban or country side areas we obtain an average translational error of 0.78%, which equals the state-of-the-art method LOAM for those sequences. The advantage of our method over LOAM is that ours also performs well in case of a tilted scanner resulting in a limited (horizontal) field of view. On the 'Hasselt' dataset we achieve an average translational error of 1.4%, which is only slightly higher than the result on the KITTI dataset.

Finally, in this chapter we also proposed multi-resolution ICP to limit the execution time. Experiments on the indoor sequences demonstrated that the multi-resolution ICP converges in less than 100 ms whereas the standard ICP needs several seconds to converge.

6 Globally consistent maps

In this chapter we will explain the concept of globally consistent maps and review the techniques that can be applied in order to obtain them. As mentioned in chapter 2, the scan matching process - referred to as the front-end of the SLAM system - does not produce perfect pose estimates. Imperfections are caused by a plethora of reasons including the inhomogeneous and sparse point density leading to the correspondence ambiguity, numerical rounding errors or approximations in case of closed-form solutions. On top of that, the sensing modalities themselves are inherently imperfect and produce inaccurate measurements.

When all these small errors accumulate over time, which is inherent to dead reckoning, we can eventually end up with large absolute errors between points acquired in the beginning and at the end of the acquisition. This phenomenon is called *drift* and when the robot or vehicle is visiting places more than once, inconsistencies such as the one depicted in Figure 2.2, can occur.

Fortunately, when the robot returns to an earlier visited place, we are able to quantify the total error that has been made along the way and propagate it back in the *pose* graph. This evidently requires algorithms that can recognize those revisited areas as well as algorithms that can *close* the loop and correct the errors properly. However, existing solutions for loop detection and correction in 3D data are computationally demanding.

In this chapter we aimed at speeding up both loop detection and correction, thereby focussing on data produced by scanning lidar devices. We propose two different algorithms that are able to respectively detect and correct loops in 3D lidar data in a highly efficient way while at the same time increase their effectiveness compared to the state-of-the-art.

6.1 Loop detection

Before we can actually ‘close’ loops and correct the accumulated error, we need to recognize them first. There are a lot of loop detection methods - also referred to as place recognition - in the SLAM literature, but none of them are really successful. Many of them use a regular RGB-camera as it provides rich information about the scene. Detecting loops using lidar sensors alone turns out to be a lot more challenging. Below we will give a brief overview of place recognition methods that are based on 3D scan data.

6.1.1 Literature overview

A quick review on the literature reveals that loop detection in 3D data can roughly be categorized into three main classes: local keypoint detection and matching in combination with a Bag-of-Words (BoW) approach, global descriptor matching and loop detection based on geometric primitives or whole objects.

The algorithms in the first category generally detect *salient* keypoints in a point cloud, compute *signatures* for these keypoint positions, build a BoW and finally match them in different scans [Steder 11]. Many 3D keypoint detectors have been proposed in literature, including SIFT 3D [Scovanner 07], the intrinsic shape signatures (ISS) [Zhong 09], Harris 3D [Sipiran 10a], NARF [Steder 10], as well as many descriptors such as spin images [Johnson 97], fast point feature histogram (FPFH) [Rusu 09], SHOT [Salti 14], etc. However, despite this abundance of choice, the detection of distinctive keypoints with high repeatability remains a challenging problem in 3D point cloud analysis, especially when the point clouds have a sparse and inhomogeneous point density as is the case for those acquired with lidar scanners.

One way of dealing with this lack of high repeatability is by using *global* descriptors which usually come in the form of histograms. Examples are the Viewpoint Feature Histograms (VFH) [Rusu 10] or multiview 2D projection (M2DP) [He 16]. The latter is a global descriptor especially designed for loop detection and is based on the projection of the point cloud on multiple non-parallel planes. Another global descriptor is the normal distributions transform (NDT) [Magnusson 09a], which was discussed in chapter 3. As the normal distributions transform offers a compact surface shape representation, it lends itself perfectly to describe the general appearance of a location.

However, global descriptors along with their local counterparts still suffer from respectively the lack of descriptive power or the lack of invariance for scale or rotation. As a result, when used for loop detection, many loops will not be recognized or too many false positives will be present, which on its turn imposes a restriction on fast and reliable loop closure. Another drawback of keypoint-based methods is that it is often hard to find a priori a suitable threshold for matching scores in such a way that the system is not overwhelmed by false positives nor starved for true matches. A final drawback of both local or global feature matching is its high computational cost, especially as most existing approaches employ a pairwise comparison of place descriptors. Finding matches in that case requires linear time per place.

One way to achieve sub-linear matching times is by using a keypoint voting approach as was proposed by [Bosse 13]. In their algorithm, a set of keypoints is chosen from a place π_i's local point cloud and a descriptor is computed for each keypoint. Next, a database containing the descriptors from all other places is queried for the k nearest neighbours to each descriptor in place π_i. Each descriptor match subsequently votes for its associated place π_j and the places with a sufficient number of votes, i.e. above a certain threshold, are considered as candidates for place matches.

More recently, researchers tend towards the application of convolutional neural networks (CNNs) to learn both the feature descriptors as well as the metric for matching them in a unified way [Zeng 17a; Yin 17a; Yin 17b; Dewan 18]. A severe limitation of these methods on the other hand is that they need a tremendous amount of training data. Moreover, they do not generalize well when trained and applied on data with a varying point density or acquired under different conditions. A model that was trained on data originated from traffic environments, might perform poorly on indoor scenes and vice versa.

A third group of methods focus on place recognition based on either complete objects or planes. In [Moral 13], the authors presented a place recognition algorithm based on plane-based maps. Unfortunately, their approach is only suitable for indoor man-made environments and is hardly applicable for large outdoor scenes. [Dubé 16] on the other hand proposed a method based on segments for which they compute several descriptors that are integrated in a learning framework. Before feeding the segment descriptors to the recognition model, they first subject them to a geometric verification test. Note that their method differs from local keypoint matching as it is based on the recognition of whole objects. As it does not rely on the presence of planes, but also exploits other objects, it is a lot more widely applicable than the work of [Moral 13]. However, when used in man-made environments such as buildings, some scans (e.g. parts of corridors) might still lack geometrical features. The last resort to detect loops in that case is to combine the lidar data with visual information, as they did in [Zhu 18].

In this PhD, we improved the global descriptor *multiview 2D-projection* (M2DP) proposed in [He 16]. Compared to other global descriptors, it is a lot more suitable for 3D scan data with a non-uniform point density. In addition, it lends itself perfectly to be executed in parallel, which inspired us to develop a GPU-based implementation of it. Besides the improvements on the M2DP method, we propose a global registration technique based on 4-points congruent sets to determine the transformation between the two loop ends, which also provides an additional verification test. We improved the efficiency of the original algorithm of [Mellado 14] significantly by omitting its randomized component, which leads to faster execution times. Moreover, our improvements make the registration technique more robust for non-uniform 3D scan data.

6.1.2 Proposed approach

Our loop detection pipeline consists of two main steps: 1) descriptor matching through multi-view 2D projection and 2) registration of the point clouds representing the two loop ends. Prior to these two steps, we additionally perform a quick selection of loop candidates by testing for each newly computed position whether or not it is *close* to a place we already visited. Assuming that the local registration is quite accurate, we set a threshold on the distance between two positions (i.e. the translational components of the poses) for them to be considered as a loop candidate. For example, we set a distance threshold of 10% of the total trajectory length since the last loop closure. When the robot then travelled for 100 meter, we consider a location to be matchable if its computed position is within 10 meter. Once such a potential loop is detected, the next step tries to the match the ‘start’ and ‘end’ point cloud. In order to avoid several consecutive point clouds to be detected as loop, we enforce a minimal number of intermediate scans between the loop start and end.

The actual matching is done using a global signature that we compute for each of the two point clouds. The signature is based on the projection of the point cloud on several non-parallel 2D planes, similar as the method of [He 16]. If the matching residual of this method is sufficiently low, we consider it as a strong loop candidate. To guarantee that we are not dealing with a false positive, the next step seeks for the transformation that aligns the two point clouds. In case the loop candidate is a true positive, we expect the overlap of the two point clouds $\mathcal{P}$ and $\mathcal{Q}$ after registration to be very high. We define this overlap by the *largest common point set* (LCP) criterion, which denotes the set of points $S \in \mathcal{P}$ that are within a predefined distance Δ from a point in $\mathcal{Q}$ after applying the transformation.

To determine this transformation between the two ends of the loop, we adopt a *global* alignment technique, based on 4-points congruent sets. The alignment

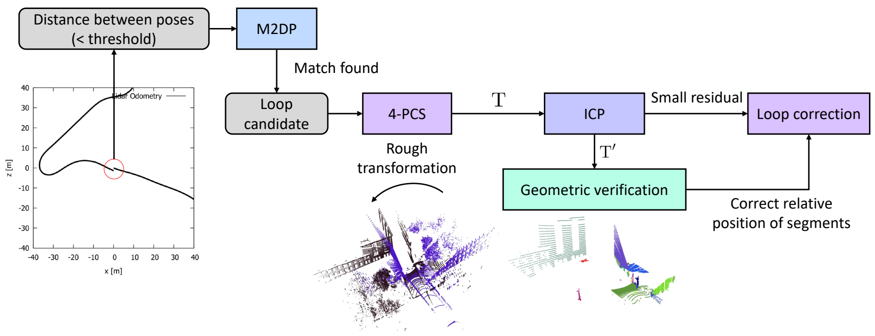


Figure 6.1: Our loop detection pipeline. The global descriptor MD2P detects loop candidates, after which the 4-PCS and ICP algorithm try to align both ends of the loop. Finally, the geometric verification step checks if the different segments in the scene are relatively at the same position.

of the two ends is essential to eventually close the loop, as it can happen that the same position is reached from different directions. The lidar scans can thus be acquired from different viewpoints. The use of an ICP-based method is in this case not effective as the point clouds might be too much rotated compared to each other, which would keep ICP from converging to the correct transformation. We still use the ICP algorithm after the rough alignment in order to refine the transformation estimate.

Eventually, we perform a quick geometric verification test to check if the objects in the two (transformed) point clouds are relatively still located at the same position. To that end, we use the point clustering algorithm described in chapter 4. Figure 6.1 depicts a schematic overview that summarizes our approach.

6.1.2.1 Multiview 2D-projection

The method of multiview 2D-projection (M2DP) was initially presented by [He 16]. The main idea is to define a set of planes on which the 3D points are projected. The planes are then further subdivided by circular sectors (or bins) and finally the number of points in each sector is counted. One of the main benefits of this approach is that it is not relying on the estimation of surface normals for the points, which are often inaccurate for lidar data given their non-uniform point density. When used in a global point cloud descriptor, a slight normal deviation can lead to non-repeatability issues.

The algorithm is summarized as follows. In order to achieve rotation invariance, we first compute the centroid of the point cloud $\mathcal{P}$ and shift it towards this centroid to achieve zero mean for the points. Second, we define a reference frame by estimating two dominant directions of the point cloud using PCA. The two principal components are then set as the x- and y-axis of the descriptor reference frame. The third step consists of generating several 2D signatures by projecting 3D data onto several predefined planes. Each plane can be represented using the *azimuth* angle θ and *elevation* angle ϕ. Thus, each pair of parameters $[\theta, \phi]$ leads to a unique plane X_j with normal vector $\mathbf{n}_j = [\cos\theta\cos\phi, \cos\theta\sin\phi, \sin\theta]^\top$. The projection of a point $\mathbf{p}_i \in \mathcal{P}$ on X_j is then given by:

$$\mathbf{p}_i^j = \mathbf{p}_i - \frac{\mathbf{p}_i^\top \mathbf{n}_j}{||\mathbf{n}_j||_2^2}\mathbf{n}_j. \tag{6.1}$$

To describe the distribution of the points on X_j, the 2D plane is further divided into bins as follows (cfr. Figure 6.2). Starting from the projected centroid on X_j, l concentric circles are generated, with radii $[r, 2^2r, \ldots, l^2r]$ and the maximum radius is set to the distance of the centroid and the furthest point. Each ring is divided in t bins, hence defining $l \times t$ different bins. For every bin, the number of projected points lying in it are counted, generating a signature vector v_j of length lt describing the points projected on plane X_j. This signature is computed for p different azimuth angles and q different elevation angles where the stride on azimuth is $\frac{\pi}{p}$ and the one on elevation $\frac{\pi}{2q}$. Hence, there are $p \times q$ different planes, leading to a signature matrix A of size $pq \times lt$, in which each row corresponds with a single signature vector v_j. Finally, a SVD decomposition is run on the matrix A and the first left and right singular vectors are concatenated to form the final descriptor.

Many parts of this algorithm can be computed in parallel, allowing us to exploit the multi-core nature of modern GPU's. For that reason, we developed a novel implementation that exploits the GPU to its maximum, making use of the Quasar programming language presented in [Goossens 14]. Specifically, three major parts were accelerated, the first one being the determination of the two dominant directions of the point cloud computed using PCA. Our PCA implementation uses a parallelized version of the SVD algorithm. The second part deals with the projection of the points on the different planes $X_j, j = 1 \ldots pq$, all of which can be computed in parallel. This leads to a speed up for this part of $pq \times N$, N being the number of points in $\mathcal{P}$, compared to a serial version of the algorithm. Finally, once the projections are computed for all planes and all points, the numbers of points belonging to each bin can also be determined in parallel.

6.1.2.2 4-points congruent sets

Using the M2DP descriptor described in the previous section, we can pairwise compare the descriptors for each scan. If the Euclidean distance between two descriptors is lower than a threshold, we consider the two locations as potential

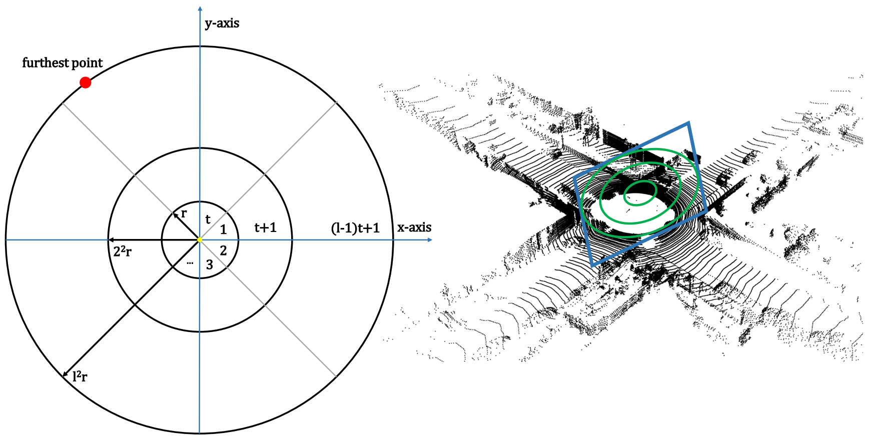


Figure 6.2: The generation of a 2D signature by projecting the 3D data onto a plane. The plane is divided into bins as follows: starting from the projected centroid, l concentric circles are generated and the maximum radius is set to the distance between the centroid and the furthest point. Each ring is subsequently subdivided into t bins, generating a signature vector of length lt. Finally, the number of points lying in each bin are counted.

loop candidates. However, as the same location can be visited from two different directions, we need to compute the transformation between the two point clouds in order to *discard* this transformation from the error back propagation. We therefore need a registration algorithm that can deal with large variations in orientation.

In chapter 3 we discussed point cloud registration in view of scan matching. In that case, we assumed that the two point clouds are only slightly shifted from each other. However, when the user is revisiting a place, this assumption no longer holds true as both point clouds can be heavily rotated compared to each other. The techniques described in chapter 3 such as ICP and NDT can, therefore, no longer be used. What we need is a registration algorithm that can deal with large variations in orientation, which we will denote as a *global* point cloud registration method. One such registration method is based on the matching of 4-points congruent sets, first presented by [Aiger 08] and later improved by [Mellado 14].

Figure 6.3 depicts a bird's-eye view of the point clouds at both ends of the loop and demonstrates why the registration is necessary. The black and gray point clouds are respectively the start and end point of a loop. The green point cloud is the transformed version of the gray one. If after registration the overlap between the two point clouds (green and black) is sufficiently high, we accept it is a true loop. Recall that we define the overlap between two point clouds using the largest common point set (LCP) criterion.

Another key thing to keep in mind is that the scene can change between the two

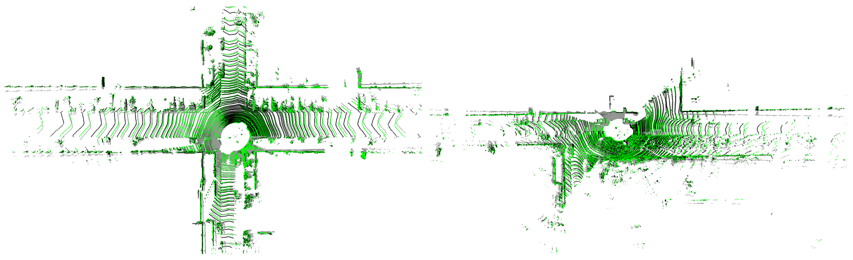


Figure 6.3: Bird's-eye view of two point clouds from the KITTI 00 sequence. The black and gray scans have been selected by the M2DP algorithm as the start and the end point of a potential loop. One point cloud is however slightly rotated compared to the other one. Our registration technique aligns both (the green one is the transformed version of the gray point cloud) and determines their overlap. When the overlap - defined by the *'largest common point set'* (LCP) criterion - is sufficiently high, the loop candidate is eventually selected as a true loop.

acquisitions processed in loop closing. For example, a parked car present in the first acquisition may no longer be present in the second. Local feature descriptors used in a BoW approach are vulnerable in these scenarios. For that reason, our global registration technique takes into account the main geometry of the scene instead of relying on keypoint positions.

The *4-points congruent sets* (4-PCS) method is thus a global point cloud registration technique that does not rely on the extraction of feature points. Instead it is matching congruent sets in both point clouds thereby adopting a *generate-and-test* paradigm, known from *random sample consensus* (RANSAC) solutions. In its most simple form, RANSAC randomly selects three points from both the source point cloud $\mathcal{P}$ and the target point cloud $\mathcal{Q}$ and subsequently computes the rigid transformation using these points. Next, it tries to *verify* this transformation by determining how many points from $\mathcal{P}$ are within a $\boldsymbol{\Delta}$-distance from points in $\mathcal{Q}$ after applying the transformation. If this count - usually referred to as the size of the *consensus set* - is sufficiently high, the transformation is accepted as a good solution. Otherwise, the process is repeated by randomly selecting another triplet of points. The transformation with the largest consensus set is finally accepted as the *best fit*.

The 4-PCS method builds on this randomized alignment approach, but instead of exhaustively testing all the triplets from $\mathcal{Q}$, it introduces the concept of *planar congruent sets* to select only a small subset of bases from $\mathcal{Q}$ that can potentially match a given base from $\mathcal{P}$. The first step in the 4-PCS method, therefore, consists of selecting a 4-point coplanar base B from the source point cloud $\mathcal{P}$. Next, from the target point cloud $\mathcal{Q}$, all 4-point sets $\{U_1, \ldots, U_N\} = \mathbf{U}$ that are approximately congruent to B are determined. Third, for all sets U_i, the rigid transformation T_i

that aligns B and U_i is computed and verified according to the *largest common point set* (LCP) criterion. This criterion denotes the set of points $S_i \in \mathcal{P}$ that are within a predefined distance Δ from a point in $\mathcal{Q}$ after applying the transformation. Finally, the best transformation $\mathbf{T}_{opt}$, i.e. the one leading to the set S_k with the highest cardinality, is kept.

In summary, the algorithm consists of three major steps: 1) selecting a coplanar base in one point cloud, 2) find the (approximate) congruent sets in the second point cloud and 3) compute and test the rigid transformations and select the best one. In the next sections we will describe each step in more detail.

Selecting a coplanar base

One of the main limitations of the original method by Aiger *et. al* [Aiger 08] is its random search for coplanar points, which is leading to a lot of unnecessary computations. Instead, we propose to cluster the 3D points and subject them to a plane fitting process, which we described earlier in section 4.2 of chapter 4. Once we have found some clusters that are part of a plane, we can extract coplanar bases very easily as any four points lying in the same cluster are coplanar. Following this procedure, we can omit the randomized base selection process of the original 4-PCS method.

Obviously, it is still beneficial to pick wide bases (by selecting points that are located far from each other) as they are in general leading to more stable alignments. To this end, we prioritize points lying at the boundaries of the planar cluster to serve as a base. Of course, the base should still lie in the overlap region between the two point clouds in order not to miss the desired solution. As we are considering point clouds that are captured at more or less the same position (but at different moments in time) we assume that the overlap will be quite large. Only in the case that a large object close to the scanner is causing a huge occlusion in one of the point clouds, this assumption might be violated. Therefore, we propose to compute for each planar region its convex hull and select a coplanar base by picking four points that are close to this convex hull.

Finding congruent sets

Once a coplanar base is selected, the next step consists of finding 4-points sets in the other point cloud that are congruent to this base. This matching step is based on a specific property of affine invariants of 4-points congruent sets, which we will explain with the help of Figure 6.4. In a nutshell, given 4 points, we can compute two independent ratios between the line segments they are defining that are preserved under affine transformations. Given a set of coplanar points

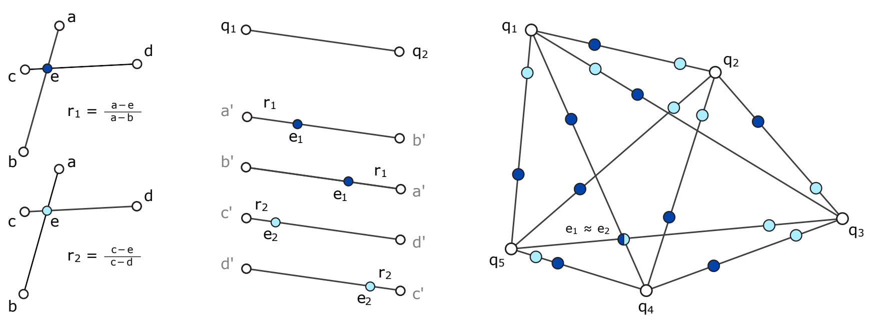


Figure 6.4: The principle of finding congruent sets in a point cloud $\mathcal{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_i, \ldots, \mathbf{q}_N\}$ given a 4-points coplanar base $\mathbf{B} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ selected from point cloud $\mathcal{P}$. The two ratios r_1 and r_2 are invariant under affine transformation, and uniquely define 4-points up to affine transformations. For each point $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{Q}$, we can compute two *intermediate* points $\mathbf{e}_1$, $\mathbf{e}_2$. Any two pairs whose intermediate points $\mathbf{e}_1$ and $\mathbf{e}_2$ are coincident, potentially correspond to a 4-points set that is an affine transformed copy of $\mathbf{B}$.

$\mathbf{B} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ from point cloud $\mathcal{P}$ that are not all collinear. Let $\mathbf{ab}$ and $\mathbf{cd}$ be the two lines that intersect at an *intermediate* point $\mathbf{e}$. The two ratios

$$r_1 = \frac{||\mathbf{a} - \mathbf{e}||}{||\mathbf{a} - \mathbf{b}||}, \qquad r_2 = \frac{||\mathbf{c} - \mathbf{e}||}{||\mathbf{c} - \mathbf{d}||} \tag{6.2}$$

are invariant under affine transformation, and uniquely define 4-points up to affine transformations. Now, for each pair of point $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{Q}$, we can compute two *intermediate* points:

$$\mathbf{e}_1 = \mathbf{q}_1 + r_1(\mathbf{q}_2 - \mathbf{q}_1), \qquad \mathbf{e}_2 = \mathbf{q}_1 + r_2(\mathbf{q}_2 - \mathbf{q}_1). \tag{6.3}$$

Any two pairs whose intermediate points $\mathbf{e}_1$ and $\mathbf{e}_2$ are coincident, potentially correspond to a 4-points set that is an affine transformed copy of $\mathbf{B}$. Of course, as this set of 4-points sets are the affine invariants of the base $\mathbf{B}$, it is a superset of the 4-points sets that are a rigid transformation of the base. For that reason, we also check the angle between the two line segments to determine if the 4-points set is a rigid transformation of the base $\mathbf{B}$.

Naturally, the intermediate points $\mathbf{e}_1$ and $\mathbf{e}_2$ will in practice never exactly be coincident due to noise and other inaccuracies. Instead, they will end up on being *nearby* points. For that reason, we set up a k-d tree search data structure that allows for fast spatial queries. We then accept the set as being congruent to the base $\mathbf{B}$ in case the distance of the two intermediate points $\mathbf{e}_1$ and $\mathbf{e}_2$ are within $\boldsymbol{\Delta}$-distance from each other.

Another limitation of the original method of [Mellado 14] is that it computes and tests for all possible combinations of points their intermediate points $\mathbf{e}_1$ and $\mathbf{e}_2$. This is leading to a tremendous amount of unnecessary computations. Instead, we propose to only process points that are lying in a physical plane that is present in the scene. Only these points qualify to be matchable with a given coplanar base, as the bases themselves were picked on the detected planar regions.

Test rigid transformation

The final step in the 4-PCS method is to test the rigid transformation computed using the base and its congruent sets. One way of verifying the transformation is by using the LCP criterion. Following this criterion one should count the number of points from the *source* point cloud that are within a $\mathbf{\Delta}$-distance from any point from the *target* point cloud after alignment. The transformation that yields the largest LCP is considered as the true transformation. We emphasize that we only compute the transformation and LCP for the congruent sets that we have selected in the previous step. This group of congruent sets is thus a lot smaller than the original method proposed by [Mellado 14].

6.1.2.3 Verification through ICP and geometrical consistency

The 4-PCS method yields a rough transformation from the source to the target point cloud but it will not perfectly align them. We therefore refine the transformation by using our improved ICP algorithm described in chapter 4. This ICP algorithm moreover offers an additional verification to conclude that the two point clouds yield a true loop. If the residual of ICP, i.e. the average distance between all corresponding points, is too large we still discard the loop candidate.

Finally, the object clustering mentioned in the previous section also provides a means of verification as we can check if all the object clusters are relatively still at the same position. To that end, we compute for each cluster its centroid and compute a bipartite matching using the Hungarian algorithm. In case these two verification steps are positive we can eventually proceed to the actual loop correction.

6.1.3 Evaluation of the loop detection

This evaluation section will cover two parts. First, we conduct an experiment to determine the speed-up factor of our GPU-accelerated descriptor computation. Second, we compare our loop detection accuracy to other commonly used methods from the literature.

Sensor	avg. $\vert\mathcal{P}\vert$	Matlab ([He 16]) in ms	GPU (**ours**) in ms
VLP-16	25446	250	**110**
HDL-32E	51687	273	**121**
HDL-64E	62594	476	**127**

Table 6.1: Computation times for our GPU implementation of the M2DP descriptor compared to the CPU version implemented in Matlab by [He 16]. The speed-up factor for data originated from the HDL-64E is almost x4 whereas for the VLP-16 and HDL-32E the speed-up factor is rather x2.5.

6.1.3.1 Speed analysis global descriptor computation

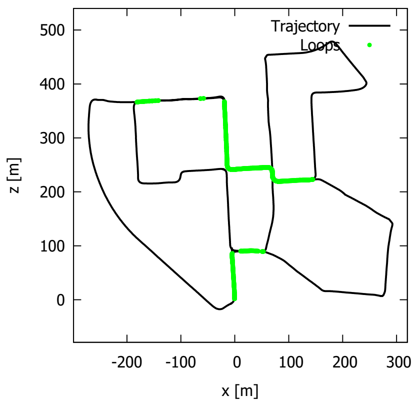
In order to compare our GPU implementation of the M2DP descriptor against the original CPU version implemented in Matlab by [He 16], we ran several experiments using data originated from several Velodyne scanners. To evaluate the performance on point clouds of different sizes, we use data from both the VLP-16, HDL-32E and HDL-64E lidar scanners. We used the Velodyne HDL-64E data from the KITTI dataset [Geiger 12] and extended it with our own recorded sequences in indoor and outdoor scenes using the Velodyne HDL-32E and VLP-16. The experiments were conducted on a computer with an Intel Core CPU i7-7820X @ 3.60Ghz, 128GB RAM and an nVidia GeForce GTX 1080ti GPU. The results are summarized in Table 6.1.

As can be seen, our GPU implementation scales well for larger point clouds. The overhead of copying data to the GPU memory is lower for larger point clouds, hence yielding a larger speed-up. For point clouds produced by the HDL-64E scanner our implementation only takes 127 ms on average compared to 476 ms for the original implementation of [He 16] which is a speed-up factor of nearly 4. For smaller point clouds - captured with the VLP-16 or HDL-32 - we notice a speed-up factor of almost 2.5.

6.1.3.2 Loop detection accuracy

In order to evaluate the accuracy of our *final* loop detector against the state-of-the-art, we once again use the KITTI dataset. More specifically, we used the sequences ‘00’, ‘05’ as these contain the most ‘revisited’ locations. To generate ground truth we used the known trajectories and considered a loop to be present when the distance between two poses is less than 1 meter. Thereby, we used a threshold of 100 scans between two loops to avoid that two subsequent poses would wrongly be classified as a loop.

In Figure 6.5 the ‘ground truth’ revisited locations in the two trajectories are



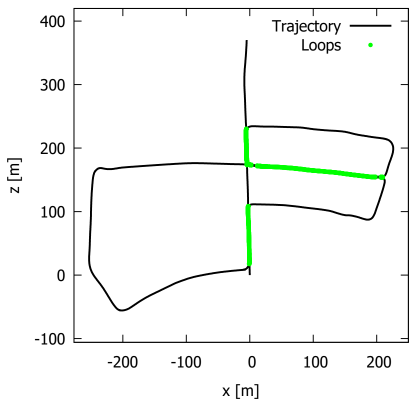


Figure 6.5: The ground truth loops in the sequences '00' and '05' of the Kitti benchmark [Geiger 12]. We consider a loop to be present when the distance between two positions is less than 1 meter.

depicted as green dots. For some sequences the same road has been taken multiple times, hence the whole part of the road is considered as a loop. However, sometimes green dots are missing along a road that has been visited multiple times, meaning that the difference in pose is larger than 1 meter. This could be due to the fact that some roads consist of multiple lanes and that a different lane was taken.

We used four different descriptors to compare our results with. The first one is the original M2DP method described in [He 16]. The other descriptors are the *ensemble of shape functions* (ESF) [Wohlkinger 11], *spin images* [Johnson 97] and *SHOT* [Salti 14]. The latter two are local descriptors so in order to use them as a global descriptor we computed the centroid of the point cloud and estimated the spin image and SHOT descriptor related to this centroid. Thereby we use the maximum distance from the centroid to any other point in the cloud as the radius to compute the descriptor.

Furthermore, both the spin images and the SHOT descriptors are based on normal vectors. To compute these, we use a radius that equals five times the average distance of a point to its closest neighbour. Regarding the spin images, the other parameters that need to be set are 1) the number of bins along one dimension, 2) the minimal allowed cosine of the angle between the normals of the input cloud and search surface (for the point to be retained in the support) and 3) the minimal number of points in the support to correctly estimate the spin image. We have set these parameters to respectively 8, 0.5 and 16. The SHOT descriptor does not have any other parameters to be set and for the ESF we again used the maximum distance of the centroid to any other point in the cloud. Finally, the M2DP method needs the

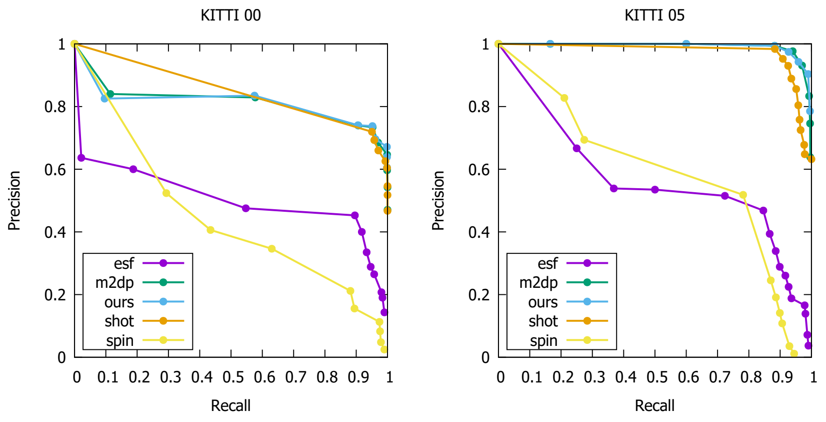


Figure 6.6: ROC curves for the several loop detection methods on the KITTI sequences ‘00’ and ‘05’. Our method performs better than the original M2DP method, the second best performer in our experiments. The SHOT detector also produces acceptable results. The results from the spin and ESF detectors are too unreliable to be used in practise.

number of bins for each plane (or expressed as the number of circles l and number of bins in one ring t) and the number of planes to use (or expressed as the azimuth p and elevation q). For our experiments we have set these values to $l = 8$, $t = 16$, $p = 4$ and $q = 16$ for all tests.

The ROC curves for the different loop detectors are depicted in Figure 6.6. We clearly see that our method along with the methods M2DP and SHOT result in the best detections. The performance of the ESF descriptor and spin images turns out to be insufficient to reliably recognize revisited areas. To obtain a recall of at least 90%, the precision drops to respectively 45% and 20% for the KITTI sequence ‘00’, which is unacceptable in an operational system. For the KITTI sequence ‘05’ the precisions corresponding to a recall of 90% are even worse, resp. 30% and 15%. On the contrary, the M2DP and SHOT descriptor are leading to a precision of 70% on the ‘00’-sequence and higher than 95% on the ‘05’-sequence for a 90% recall.

In Table 6.2, the exact precision is listed that corresponds to a recall of about 99.9%. We observe that our combined method further improves the performance of the M2DP detector. For the KITTI sequence ‘05’, to obtain 99.9% recall, we reach a precision of 90.4%, an improvement of more than 7.1%. For the KITTI sequence ‘00’, we obtain a smaller improvement of 0.8% reaching a precision of 60.5%. The lower performance on the KITTI ‘00’ dataset could be due to the inaccuracies in the ground truth. A higher threshold on the distance for a pose to be considered as a ground truth loop affects the results tremendously. The value of this experiment is therefore in the comparison between the different detectors rather than in the absolute numbers on the actual accuracy.

Sequence	Spin Image	SHOT	ESF	M2DP	**Ours**
KITTI00	0.025	0.575	0.143	0.597	**0.605**
KITTI05	< 0.01	0.632	0.037	0.833	**0.904**

Table 6.2: Precision at 99.9% recall of the different loop detection methods for the KITTI dataset. Clearly, only the detectors SHOT, M2DP and ours produce ‘acceptable’ results. Our combined algorithm improves the original M2DP method by 0.8% and 7% for KITTI sequences ‘00’ and ‘05’, leading to a precision of, respectively, 60.5% and 90.4%.

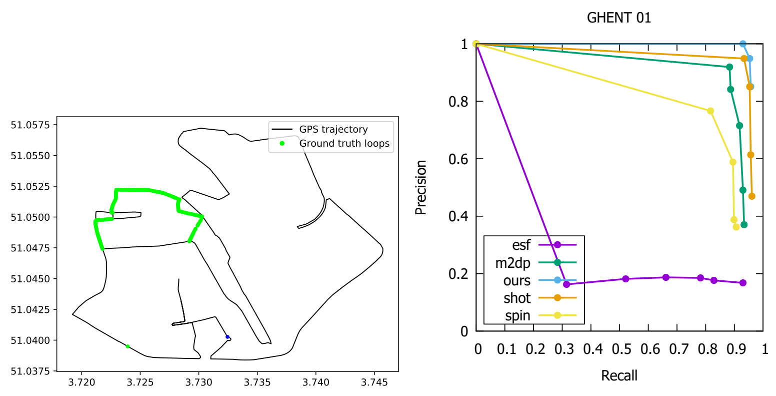


Figure 6.7: Left: the ground truth and estimated loops of the sequences recorded in the city center of Ghent, Belgium. We consider a loop to be present when the distance between two poses is less than 3 meter. Right: the ROC curve for the sequence recorded in the city center of Ghent, Belgium. Our loop detector is clearly outperforming the other methods. For a recall of 93% we still obtain a precision of 100%, which is a 5.2% gain compared to the SHOT descriptor. The latter along with the M2DP descriptor are the only two other point cloud matchers that are producing acceptable results. The loop detection quality of the spin images descriptor and ESF are too limited.

Besides experiments on the KITTI benchmark, we acquired a lidar sequence ourselves in the city of Ghent (Belgium) thereby mounting the Velodyne VLP-16 lidar scanner on top of a car. While acquiring the lidar data we used a Garmin GPS to generate ground truth. The trajectory is shown in Figure 6.7. As can be seen, a part of the trajectory was taken twice, making all the poses along this part act as loops. This time we used 3 meters as a threshold for two poses to be considered as a loop as this threshold was leading to more coherent loops along roads that were taken twice.

As the sequence was recorded in the historical city center of Ghent, the GPS signal was often times inaccurate leading to a noisy trajectory. To deal with these anomalies, we used the Google API to *clean-up* the trajectory by computing the most likely roads that were taken. This eventually lead to resp. 843 loops on a trajectory of approx. 15.7km travelled in 47 minutes thereby acquiring 31745 lidar point clouds. For this experiment we used the exact same parameters as for the KITTI sequences.

The ROC curve is depicted in Figure 6.7. In this experiment, our loop detector is clearly outperforming the other point cloud matchers. For a recall of 93% we still obtain a precision of 100%, which is an improvement of 5.2% compared to the SHOT descriptor. Only the latter method along with the M2DP descriptor are generating acceptable results. The spin images are performing better than for the KITTI sequences, but the overall accuracy is still too low to be usable in an operational system. Finally, for the ESF descriptor it is very difficult to achieve any acceptable precision, even for low recall values. Due to inaccuracies of the GPS data, and hence the ground truth, it was not possible to reach 100% recall for any of the methods, as was the case for the KITTI.

6.2 Loop correction

Once a loop has been detected, we can determine the error that has been accumulated through time and propagate it back in the pose graph. This loop correction process corresponds to the problem formulation of SLAM that we described in chapter 2. In chapter 2 we defined the SLAM problem as a maximum a posteriori (MAP) estimation. We also mentioned that this MAP formulation can be expressed by factor graphs and that for this reason it is also referred to as *graph*-SLAM. As it would lead us too far to give an exhaustive list of all solutions presented in the literature, we will limit ourselves mentioning the most important ones.

The history of MAP or graph SLAM started with the seminal work of [Lu 97], which was still conducted in 2D. Later on, [Borrmann 08] extended the approach to 3D. Since then, numerous improvements on the efficiency and robustness of the underlying optimization problem have been proposed in the literature. The most important ones are the Tree-based netwORk Optimizer (TORO) [Grisetti 09] and the g2o framework [Kümmerle 11].

6.2.1 The fundamentals

Consider an observer travelling along the trajectory $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_M)$ where at each pose $\mathbf{x}_i$ the sensor, either a lidar scanner or ToF camera, is capturing a point cloud $\mathcal{P}_i$. During the scan matching process a set of *neighbour* relations was determined between the poses. As mentioned in chapter 2, in a typical pose graph

SLAM formulation, the poses represent the nodes in the graph whereas the edges are represented by the *constraints*, in our case the neighbour relations. We now want to optimally estimate all poses to build a consistent map of the environment. We will use $\mathbf{z}_{i,j}$ to denote the 'difference' in pose $\mathbf{x}_i$ and $\mathbf{x}_j$. The true difference $\mathbf{z}_{i,j}$ is unknown and we model the 'observation' $\hat{\mathbf{z}}_{i,j}$ of the true underlying difference as $\hat{\mathbf{z}}_{i,j} = \mathbf{z}_{i,j} + \mathbf{e}_{i,j}$ where $\mathbf{e}_{i,j}$ is a Gaussian distributed error with zero mean and a covariance matrix $\mathbf{\Sigma}_{i,j}$, that is assumed to be known. The probability density function is hence given by:

$$p(\hat{\mathbf{z}}_{i,j} - \mathbf{z}_{i,j}) = \frac{1}{|2\pi\mathbf{\Sigma}_{i,j}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\hat{\mathbf{z}}_{i,j}-\mathbf{z}_{i,j})^{\mathsf{T}}\mathbf{\Sigma}_{i,j}^{-1}(\hat{\mathbf{z}}_{i,j}-\mathbf{z}_{i,j})}. \tag{6.4}$$

Instead of maximizing the above probability density function, one rather attempts to minimize its negative logarithm:

$$\ln p(\hat{\mathbf{z}}_{i,j} - \mathbf{z}_{i,j}) = -\frac{1}{2}\ln|2\pi\mathbf{\Sigma}_{i,j}| - \frac{1}{2}(\hat{\mathbf{z}}_{i,j} - \mathbf{z}_{i,j})^{\mathsf{T}}\mathbf{\Sigma}_{i,j}^{-1}(\hat{\mathbf{z}}_{i,j} - \mathbf{z}_{i,j}). \tag{6.5}$$

This provides more numerical stability as the magnitudes of the likelihoods can be very small and doing a log transform converts these small numbers to larger negative values.

Omitting the constant terms in this expression and using $\mathbf{\Omega}_{i,j} = \mathbf{\Sigma}_{i,j}^{-1}$, we obtain a cost function which is equivalent to:

$$\mathbf{F}(\mathbf{x}) \triangleq \sum_{i,j}(\hat{\mathbf{z}}_{i,j} - \mathbf{z}_{i,j})^{\mathsf{T}}\mathbf{\Omega}_{i,j}(\hat{\mathbf{z}}_{i,j} - \mathbf{z}_{i,j}) \tag{6.6}$$

$$= \sum_{i,j}\mathbf{e}_{i,j}^{\mathsf{T}}\mathbf{\Omega}_{i,j}\mathbf{e}_{i,j}. \tag{6.7}$$

The goal of generating a globally consistent map is to find the configuration of poses $\mathbf{x}^*$ that minimizes this log-likelihood function $\mathbf{F}(\mathbf{x})$:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \mathbf{F}(\mathbf{x}). \tag{6.8}$$

Almost all pose graph optimization or bundle adjustment techniques in the literature use this cost function. They only differ in the specific method they use to find the global minimum of Eq. 6.7. Using the signed incidence matrix $\mathbf{H}$, the concatenated measurement vector $\mathbf{z}$ can be written as $\mathbf{z} = \mathbf{H}\mathbf{x}$ and Eq. 6.7 can be re-formulated as follows:

$$\mathbf{F} \triangleq (\hat{\mathbf{z}} - \mathbf{H}\mathbf{x})^{\mathsf{T}}\mathbf{\Omega}(\hat{\mathbf{z}} - \mathbf{H}\mathbf{x}). \tag{6.9}$$

The concatenation of all observations $\hat{\mathbf{z}}_{i,j}$ forms the vector $\hat{\mathbf{z}}$, and $\mathbf{\Omega}$ is a block-diagonal matrix comprised of $\mathbf{\Omega}_{i,j}$ as sub-matrices. The solution that minimizes this equation is given by:

$$\mathbf{x} = (\mathbf{H}^{\mathsf{T}}\mathbf{\Omega}\mathbf{H})^{-1}\mathbf{H}^{\mathsf{T}}\mathbf{\Omega}\hat{\mathbf{z}}. \tag{6.10}$$

The matrix $\mathbf{G} \triangleq \mathbf{H}^{\mathsf{T}}\mathbf{\Omega}\mathbf{H}$ and the vector $\mathbf{b} \triangleq \mathbf{H}^{\mathsf{T}}\mathbf{\Omega}\hat{\mathbf{z}}$ simplify the notation of the solution. The matrix $\mathbf{G}$ consists of sub-matrices

$$\mathbf{G}_{i,j} = \begin{cases} \sum_{j=0}^{M} \mathbf{\Omega}_{i,j} & \text{if}(i = j) \\ \mathbf{\Omega}_{i,j} & \text{if}(i \neq j) \end{cases} \tag{6.11}$$

and the entries of $\mathbf{b} = (\mathbf{b}_0, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_M)$ are obtained by:

$$\mathbf{b}_i = \sum_{j=0, j\neq i}^{M} \mathbf{\Omega}_{i,j}\hat{\mathbf{z}}_{i,j}. \tag{6.12}$$

Solving equation 6.10 is thus equivalent to solving the following linear equation system:

$$\mathbf{G}\mathbf{x} = \mathbf{b}. \tag{6.13}$$

6.2.2 Error minimization by matching 3D scan data

In order to be able to solve the above equation, the pose differences $\mathbf{z}_{i,j}$ need to be linearized. In other words, we need to derive relations for the 6DoF poses by matching data obtained by the 3D sensor. We define a 6D pose relation at the time that scan i is acquired as $\mathbf{x}_i = (x_i, y_i, z_i, \theta_{x_i}, \theta_{y_i}, \theta_{z_i})$. At scan j, the pose has been changed by $\mathbf{\Delta x} = (x, y, z, \theta_x, \theta_y, \theta_z)$ to end up at pose $\mathbf{x}_j = (x_j, y_j, z_j, \theta_{x_j}, \theta_{y_j}, \theta_{z_j})$. The two poses $\mathbf{x}_i$ and $\mathbf{x}_j$ are related by the 'compounding' operation $\mathbf{x}_j = \mathbf{x}_i \oplus \mathbf{\Delta x}$. Similarly, a 3D point $\mathbf{p}_k = (x_k, y_k, z_k)$ is compounded with the pose $\mathbf{x}_i$ by $\mathbf{p}'_k = \mathbf{x}_i \oplus \mathbf{p}_k$ given by:

$$\mathbf{p}'_k = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z\mathbf{p}_k + \begin{bmatrix} x_i \\ y_i \\ x_i \end{bmatrix}, \tag{6.14}$$

in which $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$ denotes the rotation matrix derived from the Euler angles θ_x, θ_y and θ_z.

The scan matching described in chapter 3 determined a set of N corresponding point pairs $\mathbf{p}_k^i$, $\mathbf{p}_k^j$ between two scans i and j, each representing a single physical point. The positional error made by identifying these two points in different scans, is described by:

$$f_{i,j}(\mathbf{x}_i, \mathbf{x}_j) \triangleq \sum_{k=1}^{N} ||\mathbf{x}_i \oplus \mathbf{p}_k^i - \mathbf{x}_j \oplus \mathbf{p}_k^j||_2^2 \tag{6.15}$$

$$= \sum_{k=1}^{N} ||f_k(\mathbf{x}_i, \mathbf{x}_j)||_2^2 \tag{6.16}$$

If the global coordinates of a pair of matching points $\mathbf{p}_k = (x_k, y_k, z_k)$, then $(\mathbf{p}_k^i, \mathbf{p}_k^j)$ fulfils the following equation:

$$\mathbf{p}_k \approx \mathbf{x}_i \oplus \mathbf{p}_k^i \approx \mathbf{x}_j \oplus \mathbf{p}_k^j. \tag{6.17}$$

Consider $\hat{\mathbf{x}}_i = (\hat{x}_i, \hat{y}_i, \hat{z}_i, \hat{\theta}_{x_i}, \hat{\theta}_{y_i}, \hat{\theta}_{z_i})$ and $\hat{\mathbf{x}}_j = (\hat{x}_j, \hat{y}_j, \hat{z}_j, \hat{\theta}_{x_j}, \hat{\theta}_{y_j}, \hat{\theta}_{z_j})$ to be close estimates of $\mathbf{x}_i$ and $\mathbf{x}_j$. For small errors $\mathbf{\Delta x}_i = \hat{\mathbf{x}}_i - \mathbf{x}_i$ and $\mathbf{\Delta x}_j = \hat{\mathbf{x}}_j - \mathbf{x}_j$, we can approximate the error function $f_k(\mathbf{x}_i, \mathbf{x}_j)$ by its first order Taylor expansion, leading to:

$$\begin{aligned} f_k(\mathbf{x}_i, \mathbf{x}_j) &\triangleq \mathbf{x}_i \oplus \mathbf{p}_k^i - \mathbf{x}_j \oplus \mathbf{p}_k^j && (6.18)\\ &\approx f_k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j) - \Big[\nabla_{\hat{\mathbf{x}}_i}(f_k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j))\mathbf{\Delta x}_i - \nabla_{\hat{\mathbf{x}}_j}(f_k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j))\mathbf{\Delta x}_j\Big] && (6.19)\\ &= \hat{\mathbf{x}}_i \oplus \mathbf{p}_k^i - \hat{\mathbf{x}}_j \oplus \mathbf{p}_k^j - \Big[\nabla_{\hat{\mathbf{x}}_i}(\hat{\mathbf{x}}_i \oplus \mathbf{p}_k^i)\mathbf{\Delta x}_i - \nabla_{\hat{\mathbf{x}}_j}(\hat{\mathbf{x}}_j \oplus \mathbf{p}_k^j)\mathbf{\Delta x}_j\Big]. && (6.20) \end{aligned}$$

where $\nabla_{\hat{\mathbf{x}}_i}(\hat{\mathbf{x}}_i \oplus \mathbf{p}_k^i)$ denotes the gradient of the pose compounding operation. Using the matrix decomposition

$$\mathbf{M}_k\mathbf{H}_i \triangleq \nabla_{\hat{\mathbf{x}}_i}(\hat{\mathbf{x}}_i \oplus \mathbf{p}_k^i), \tag{6.21}$$

$$\mathbf{M}_k\mathbf{H}_j \triangleq \nabla_{\hat{\mathbf{x}}_j}(\hat{\mathbf{x}}_j \oplus \mathbf{p}_k^j), \tag{6.22}$$

Eq. 6.20 simplifies to:

$$\begin{aligned} f_k(\mathbf{x}_i, \mathbf{x}_j) &\approx \hat{\mathbf{x}}_i \oplus \mathbf{p}_k^i - \hat{\mathbf{x}}_j \oplus \mathbf{p}_k^j - \mathbf{M}_k\Big[\mathbf{H}_i\mathbf{\Delta x}_i - \mathbf{H}_j\mathbf{\Delta x}_j\Big] && (6.23)\\ &= \hat{f}_k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j) - \mathbf{M}_k\mathbf{z}_{i,j} && (6.24) \end{aligned}$$

with $\mathbf{z}_{i,j} = \mathbf{H}_i\mathbf{\Delta x}_i - \mathbf{H}_j\mathbf{\Delta x}_j$ the new linearized measurement equation,

$$\mathbf{M}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & -y_k & -z_k \\ 0 & 1 & 0 & z_k & x_k & 0 \\ 0 & 0 & 1 & -y_k & 0 & x_k \end{bmatrix}, \tag{6.25}$$

$$\mathbf{H}_i = \begin{bmatrix} 1 & 0 & 0 & 0 & \hat{z}_i\cos(\hat{\theta}_{x_i}) + \hat{y}_i\sin(\hat{\theta}_{x_i}) & \hat{y}_i\cos(\hat{\theta}_{x_i})\cos(\hat{\theta}_{y_i}) - \hat{z}_i\cos(\hat{\theta}_{y_i})\sin(\hat{\theta}_{x_i}) \\ 0 & 1 & 0 & -\hat{z}_i & -\hat{x}_i\sin(\hat{\theta}_{x_i}) & -\hat{x}_i\cos(\hat{\theta}_{x_i})\cos(\hat{\theta}_{y_i}) - \hat{z}_i\sin(\hat{\theta}_{y_i}) \\ 0 & 0 & 1 & \hat{y}_i & -\hat{x}_i\cos(\hat{\theta}_{x_i}) & \hat{x}_i\cos(\hat{\theta}_{y_i})\sin(\hat{\theta}_{x_i}) + \hat{y}_i\sin(\hat{\theta}_{y_i}) \\ 0 & 0 & 0 & 1 & 0 & \sin(\hat{\theta}_{y_i}) \\ 0 & 0 & 0 & 0 & \sin(\hat{\theta}_{x_i}) & \cos(\hat{\theta}_{x_i})\cos(\hat{\theta}_{y_i}) \\ 0 & 0 & 0 & 0 & \cos(\hat{\theta}_{x_i}) & -\cos(\hat{\theta}_{y_i})\sin(\hat{\theta}_{x_i}) \end{bmatrix} \tag{6.26}$$

and $\mathbf{H}_j$ given analogously. For the details on the derivation of the matrix decomposition and $\mathbf{H}_i$ and $\mathbf{H}_j$ we refer the reader to the work of [Borrmann 08]. We can now rewrite Eq. 6.15 in matrix form as follows:

$$\mathbf{F}(\mathbf{z}_{i,j}) \approx (\mathbf{f} - \mathbf{M}\mathbf{z}_{i,j})^{\mathsf{T}}(\mathbf{f} - \mathbf{M}\mathbf{z}_{i,j}). \tag{6.27}$$

The concatenated matrix $\mathbf{M}$ consists of all $\mathbf{M}_k$'s and the concatenated vector $\mathbf{f}$ consists of all f_k's. The vector $\hat{\mathbf{z}}$ that minimizes $\mathbf{F}(\mathbf{z})$ is given by

$$\hat{\mathbf{z}}_{i,j} = (\mathbf{M}^{\mathsf{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathsf{T}}\mathbf{f}. \tag{6.28}$$

Since minimizing $\mathbf{F}(\mathbf{z}_{i,j})$ constitutes a least squares linear regression, we model the Gaussian distribution of the solution with mean $\hat{\mathbf{z}}_{i,j}$ and standard covariance estimation

$$\mathbf{\Sigma}_{i,j} = s^2(\mathbf{M}^{\mathsf{T}}\mathbf{M}) \tag{6.29}$$

where s^2 is the unbiased estimate of the covariance of the identically, independently distributed errors of f_k, given by:

$$s^2 = \frac{(\mathbf{f} - \mathbf{M}\mathbf{z}_{i,j})^{\mathsf{T}}(\mathbf{f} - \mathbf{M}\mathbf{z}_{i,j})}{2N - 3} = \frac{\mathbf{F}(\mathbf{z}_{i,j})}{2N - 3}. \tag{6.30}$$

The algorithm for graph optimization can be summarized as follows. From the scan matching process (front-end) collect the point correspondences $\mathbf{p}_k^i$, $\mathbf{p}_k^j$. Next, for any link (i, j) in the given graph compute the measurement vector $\hat{\mathbf{z}}_{i,j}$ given by eq. 6.28 and its covariance $\mathbf{\Sigma}_{i,j}$ given by eq. 6.29. Using $\mathbf{z}_{i,j}$ and $\mathbf{\Sigma}_{i,j}$, construct the linear system $\mathbf{G}\mathbf{x} = \mathbf{b}$ with $\mathbf{G}$ and $\mathbf{b}$ given by eqn. 6.11 and 6.12. Finally, solve for $\mathbf{x}$ and update the poses and their covariances.

The above procedure is usually repeated until convergence is reached. Thus, after one iteration, the updated poses of the SLAM back-end are given to the front-end, the scan matching process is rerun and the correspondences between the point clouds are redetermined. After that, the outcome is fed back to the back-end to check if this re-investigation has lead to a better result. Based on the new correspondences, the graph optimization is eventually recomputed and the whole process is repeated.

6.2.3 Heuristic loop closing

The procedure described in the previous section is computationally very expensive. When the goal is to implement loop closure as an online process one has to seek refuge in heuristic loop closing approaches. These are often times a lot faster, but generally lead to a sub-optimal solution. In our solution, presented in [Vlaminck 18a], the idea is to bypass the iteration between the SLAM front- and back-end. Recall that the front-end is referring to the scan matching process whereas the back-end deals with the global consistency of the 3D map and hence the correction of the loops. We propose to avoid the iterative behaviour by passing the *local* information from the scan matching just once to the back-end, but not the other way round. The idea is to investigate how much each pose contributes to the final accumulated error, thereby considering the residuals for every transformation estimated during the scan matching process.

The actual correction is then done as follows. We assume that the loop detection method provided us with two point clouds $\mathcal{P}_s$ and $\mathcal{P}_e$, resp. the *start* and *end* of the loop and let us denote their pose matrices as respectively $\mathbf{P}_s = (\mathbf{R}_s, \mathbf{t}_s)$ and $\mathbf{P}_e = (\mathbf{R}_e, \mathbf{t}_e)$. Next, the difference in pose, i.e. the loop transform, is given by $\boldsymbol{\Delta} = (\mathbf{T}_{s,e}\mathbf{P}_e)^{-1}\mathbf{P}_s$. Note that the transformation $\mathbf{T}_{s,e}$ is the result of aligning $\mathcal{P}_s$ and $\mathcal{P}_e$. It should be 'discarded' from the loop correction process as it does not yield an error, but results from visiting the same place from different directions. The transformation moreover 'corrects' small translations between the two 'loop' poses, as it is unlikely that a scan was taken from exactly the same spot.

The loop transform $\boldsymbol{\Delta}$ is considered the true *error* as both poses $\mathbf{T}_{s,e}\mathbf{P}_e$ and $\mathbf{P}_s$ should be equal. We therefore suggest to project $\boldsymbol{\Delta}$ back in the pose graph. The key idea is to use the residuals of the scan matching process, i.e. the final (ICP) costs after transformation, to assign a weight $c_{i,j}$ to each edge in the pose graph. We assign a higher weight to those transformations that yield a high residual in the scan matching step. This is motivated by the fact that a high residual indicates that two consecutive point clouds were potentially inaccurately aligned. Thereby we assume that the scan matching process was already converging in the right direction.

Next, we define the *distance* between two poses $\mathbf{P}_k$ and $\mathbf{P}_l$ as $d(\mathbf{P}_k, \mathbf{P}_l) = \sum_{i,j} c_{i,j}$. Herein, $\{i, j\}$ denotes the set of all edges in the path from $\mathbf{x}_k$ to $\mathbf{x}_l$. Finally, we define a weight

$$w_i = \frac{d(\mathbf{P}_s, \mathbf{P}_i)}{d(\mathbf{P}_s, \mathbf{P}_e)} \tag{6.31}$$

for each pose in the graph that specifies the fraction of the matrix $\boldsymbol{\Delta}$ by which the pose has to be transformed. The poses $\mathbf{P}_k$ are then updated replacing $\mathbf{t}_k$ by $\mathbf{t}_k w_k \boldsymbol{\Delta}$ and $\mathbf{R}_k$ by $slerp(\mathbf{R}_k, w_k\boldsymbol{\Delta})$, *slerp* denoting the spherical linear interpolation function as described in [Shoemake 85].

6.2.4 Evaluation of the loop correction

To assess the quality of our loop correction algorithm, we use the same 'Hasselt' dataset as in chapter 5. In order to obtain the *initial* trajectory we use the W-GSS-ICP scan matching approach in combination with scan-to-model alignment. In Figure 6.8, the result of this reconstruction process, before loop closure, is shown in the left image whereas the right picture depicts the point cloud after loop closure. As can be seen in the left point cloud, the end of the loop is not connected with the start of the loop. After correction the two ends are connected and the accumulated

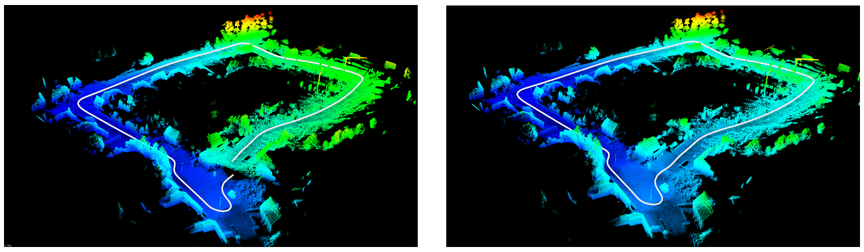


Figure 6.8: The two resulting reconstructions for the ‘Hasselt’ dataset, before loop closure (left) and after loop closure (right). After loop correction, the two ends of the loops are attached and the error is propagated back in the pose graph.

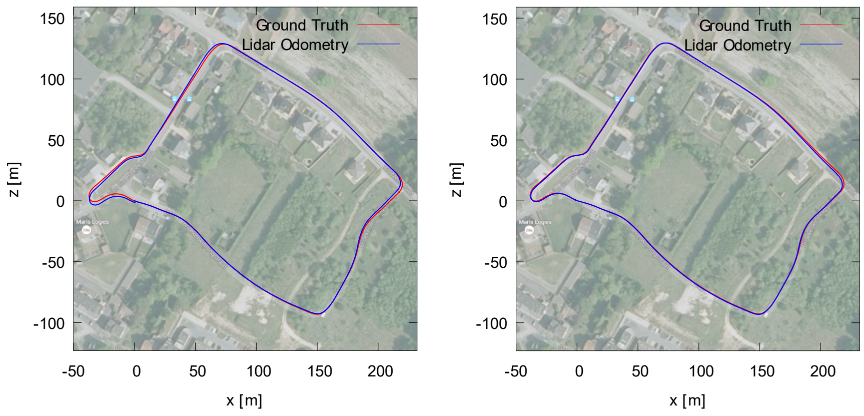


Figure 6.9: The two resulting trajectories for the ‘Hasselt’ dataset corresponding to the reconstructions in Figure 6.8, before loop closure (left) and after loop closure (right).

error is propagated back in the pose graph. In order to evaluate this quantitatively, we again used the POS-LV positioning system from Applanix as ground truth. Figure 6.9 and Figure 6.10 depict both the ground truth and the *final* estimated trajectory before and after performing loop closure. After loop closure, there is almost no difference noticeable between the estimated and ground truth trajectory.

To evaluate the performance quantitatively, we computed the average Euclidean distance between the positions (i.e. the translational components of the poses) in the estimated trajectory and the positions in the ground truth trajectory. Before loop closure the average Euclidean distance is 5.65 meter whereas after loop closure it is reduced to 3.13 meter. Furthermore, the median errors before and after loop closure are respectively 3.98 meter and 2.13. The loop closure process thus reduces the total error by a factor of almost 2.

As small errors are quickly leading to large errors between the estimated and

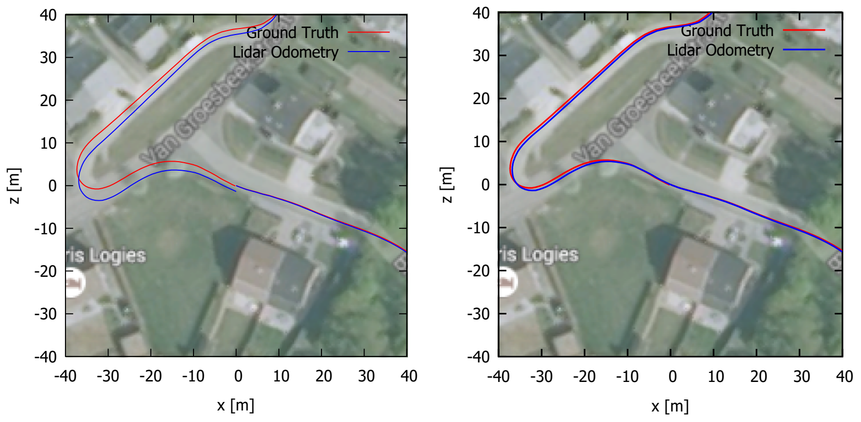


Figure 6.10: Zoomed in version of the resulting trajectory of the Hasselt survey at the start and endpoint of the sequence before loop closure (left) and after loop closure (right)

true positions, we also consider the *absolute pose error* (APE), which was defined in eq. 2.16 of chapter 2. To evaluate the APE of eq. 2.16 the estimated trajectory is first transformed in such a way that it is mapped onto the ground truth trajectory in a least-squares manner (referred to as the *'Umeyama'* method). Doing so, the influence of a severe outlier poses is limited. The result of this mapped trajectory before and after loop closure is depicted in respectively the left and right image of Figure 6.11. Note that the colouring has a different scale for each of the two graphs. The RMSE of the APE before and after loop closure is respectively 2.35 meter and 0.90 meter, which is an improvement with a factor 2.6.

The colouring of the right graph of Figure 6.11 also demonstrates in which areas our method has the most difficulties. The right upper turn of the trajectory is coloured in red, meaning that the largest part of the total error is made there. This is not surprising as it corresponds to a turn which is part of an intersection. At the time of acquisition, some cars were passing by, disturbing the scan matching process. On top of that, one side of the road is fully covered with large trees. The cyan to yellow parts of the trajectory also denote areas that are surrounded by vegetation and lack the presence of buildings.

The total time to correct the loop on an Intel Core i7-4712HQ CPU @ 2.30 GHz, was only 11 milliseconds. This extremely fast operation is thanks to the bypass of the iteration between the SLAM front- and back-end. During the SLAM front-end a residual score was computed that is later on used in the loop correction phase. The loop correction itself only involves one single manipulation of the poses.

In a second experiment, we close the loop in sequence '09' of the Kitti dataset. The result is shown in Figure 6.12. The trajectory after loop closure (in green) and

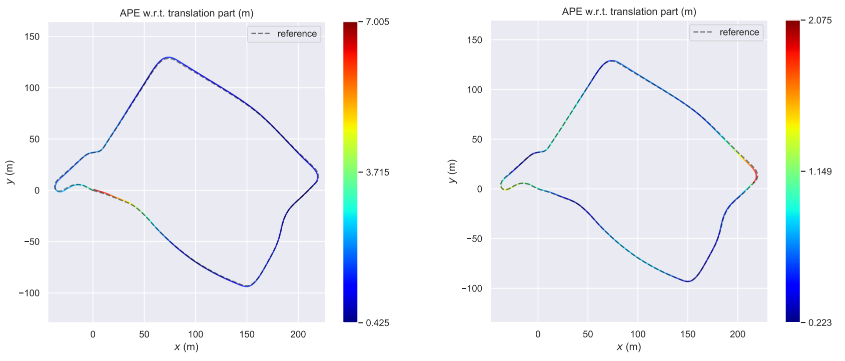


Figure 6.11: The two resulting trajectories for the 'Hasselt' dataset corresponding to the reconstructions in Figure 6.8, before loop closure (left) and after loop closure (right), together with the absolute pose error (APE). The RMSE error of the APE before and after loop correction is respectively 2.35 and 0.90 meter, hence proving the effectiveness of the loop correction.

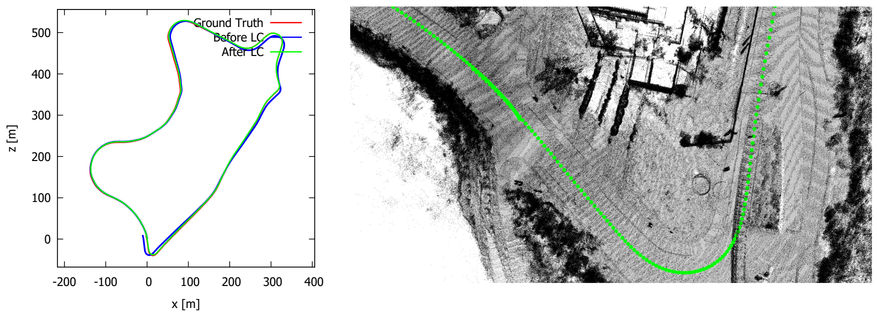


Figure 6.12: The resulting trajectory (left) of sequence '09' of the Kitti benchmark, before loop closure (blue) and after loop closure (green). The estimated and the ground truth trajectory are almost entirely overlapping after loop correction. Right: the corresponding reconstruction at the start and end-point of the loop after correction. Little to no artefacts can be seen in the reconstruction. The pose error before and after loop correction is respectively 9.89 and 4.80 meter, hence proving its effectiveness.

the 'ground truth' trajectory (in red) are almost entirely overlapping. On the right image, the point clouds at the start and end-point of the loop are shown. Little to no artefacts are noticeable in the point cloud. The average pose error before and after loop correction is respectively 9.89 meter and 4.80 meter. The median value before and after loop correction is 7.85 meter and 3.53 meter respectively. In summary, we can conclude that we are able to reduce the error with a factor 2 after loop closure. The results of both experiments are summarized in Table 6.3.

Sequence	**Length (m)**	**Avg. Error (m)**			**Med. Error (m)**		
		no LC	LC	gain	no LC	LC	gain
HASSELT	715	5.65	3.13	$\times$ 1.81	3.98	2.13	$\times$ 1.87
KITTI09	1705	9.89	4.80	$\times$ 2.06	7.85	3.53	$\times$ 2.22

Table 6.3: The average and median pose error before and after loop closure for the Hasselt and KITTI "09" sequence. Our correction algorithm reduces the error by a factor of approximately 2 after closing the loop.

6.3 Conclusion

In this chapter, we presented several contributions on both loop detection and correction. Regarding loop detection, we accelerated the process by developing a GPU-accelerated global feature descriptor for point clouds that is very effective to provide loop candidates. We showed that our implementation is a factor 2.5 to 4 times faster than the original version.

In addition, we proposed a registration technique based on 4-points congruent sets (4-PCS) that is able to align two point clouds with a large deviation in orientation. We improve over existing techniques regarding both accuracy and speed. The former is achieved by taking wide bases to compute the transformation upon. The latter is achieved by avoiding the randomized planar base selection from the original 4-PCS method. More specifically, we exploit the knowledge of the planar areas in the scene - which we estimated using our clustering algorithm presented in chapter 4 - to select 4-point bases that are coplanar.

Furthermore, we showed the effectiveness of combining this registration technique with the global feature descriptor to evaluate potential loop candidates. The experiments have demonstrated that we can gain up to 7% in precision for a recall of 99.9%. The experiments also showed that our method works for data acquired with different Velodyne scanners consisting of 16, 32 or 64 lasers.

Finally, we proposed a novel loop correction method with superior execution time. We achieved this by avoiding the iteration between the SLAM front-end and back-end. Instead we exploit the residuals from the ICP scan matching front-end to assign a weight to each pose in the graph and we only optimize the graph once. Experiments demonstrated that our loop correction method is able to reduce the absolute pose error before and after loop closure by a factor of approximately 2.

7 Use cases of lidar-based mapping

This chapter will describe several use cases that demonstrate the effectiveness and applicability of our mobile mapping system. The use cases cover a broad variety of environments from typical 'Manhattan world' scenes to more advanced sculptural buildings or complex industrial sites. The experiments also include outdoor environments, which are inherently challenging due to the presence of unstructured natural phenomena. In all this, we tested multiple lidar devices with different specifications regarding the number of lasers, the field of view or angular resolution. We also experimented with both a horizontally mounted scanner and a tilted one, the latter yielding the additional difficulty that the 'horizontal' field of view of the scanner becomes limited, making consecutive scan matching challenging.

To ease the 3D reconstruction from a practical point of view, we furthermore developed an autonomous robot, which we named Liborg as its main sensor is based on lidar technology. All of the 3D point cloud models that we created served their own specific purpose. In one of the use cases we dealt with the mapping of a chemical plant that was disused and planned to be partly demolished. The chemical site was mapped in view of a *'built-as-planned'* AR-application and had to be mapped with high precision, i.e. sub-centimeter accurate. The other 3D models either served as a digital twin or were meant to be used in VR applications or to be 3D-printed in view of exposing them at an architectural exhibition.

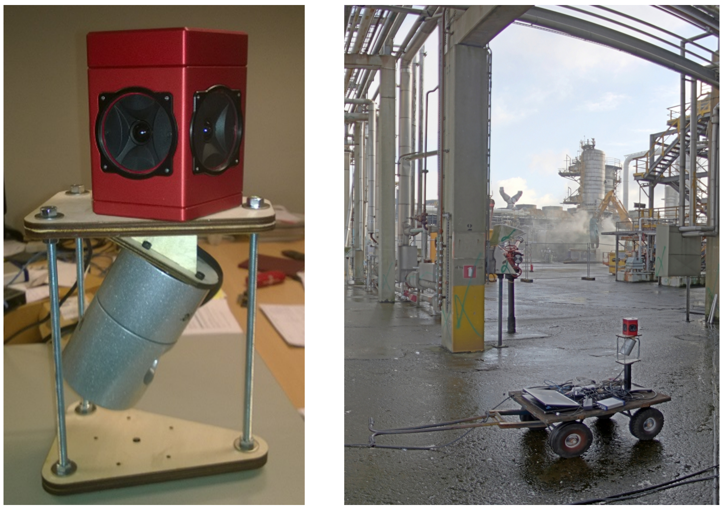


Figure 7.1: Our *Vellady* mobile acquisition platform in close-up (left) and mounted on a four wheel trolley (right).

7.1 The Vellady platform

Prior to the development of the Liborg robot, we designed the *Vellady* mobile mapping platform, which consists of a Velodyne HDL-32e scanner and a Ladybug panoramic camera system (hence its name). As can be seen in Figure 7.1, the Ladybug camera system is mounted perpendicular to the ground plane, whereas the Velodyne is tilted on its head making an angle of approximately 66° with the ground plane. The Ladybug is a fixed system that comes in the form of a pentagonal prism consisting of five vertical-oriented sides, each one incorporating a camera. A sixth camera is mounted on top, pointing upwards. During the experiments, we mounted this Vellady platform on a moving vehicle, either a kitchen cart or a four wheel trolley, cfr. Figure 7.1.

A rotation of the Velodyne scanner around its roll or pitch axis will only occur in case the ground plane has a slope but these situations were not encountered during the experiments. Our system still operates as a full 6 degrees of freedom (6DoF) SLAM algorithm, incorporating three unknown position coordinates x, y, z and three rotation angles θ_x, θ_y, θ_z. The main benefit of adopting 6 DoF is that we are able to cope with small vibrations of the lidar scanner. For clarity, the main specifications of the Vellady sensors are summarized in table 7.1.

	Sensor	Feature
	Ladybug system	- 6 camera's in pentagonal prism - resolution of 1600×1200 per image - 1 frame (6 images) per second - perpendicular w.r.t. ground plane
	Velodyne HDL-32E	- 360° horizontal FOV, 41.3° vertical FOV - 32 lasers spinning at 10 sweeps per second - ± 700.000 points per sweep - angle of 66° w.r.t. ground plane

Table 7.1: A summary of the main specifications of the Vellady platform.

7.1.1 Synchronisation

In order to synchronise the Velodyne scanner and the Ladybug, a synchronisation module was developed on a an Arduino development board. The latter emits a pulse to the Ladybug every second, triggering each of the six cameras to take a picture. The frequency of this pulse can be increased or decreased depending on the needs of the application. At the same time as the pulse, a signal is sent to the Velodyne scanner in order to receive a 'GPRMC NMEA'-record. The timestamp of this record corresponds with the amount of seconds past since the synchronisation device was powered on. In case an external GPS system is added to the Velodyne, the exact location can be added in this record as well.

The synchronisation device is useful for many reasons. First, it is necessary to match the pixels of the image to the right 3D points acquired by the lidar scanner. Second, the 3D reconstruction can be computed using a combination of both the camera and the lidar scanner. Finally, loop detection can be conducted using both sensors, making it more reliable.

7.1.2 Calibration

As the Velodyne and the Ladybug each have their own coordinate system, we need to define a reference coordinate system to harmonize both. To ease this task, the Ladybug was mounted straight on top of the Velodyne scanner in such a way that the vector connecting both origins is perpendicular to the ground plane. Doing so, it is sufficient to determine the orientation of the Velodyne sensor with respect to the ground plane and to measure the offset of the origins of both sensors.

To determine the orientation of the Velodyne scanner, we use the output of its accelerometer. Accelerometers measure the difference between any linear acceleration in the accelerometer's reference frame and the earth's gravitational

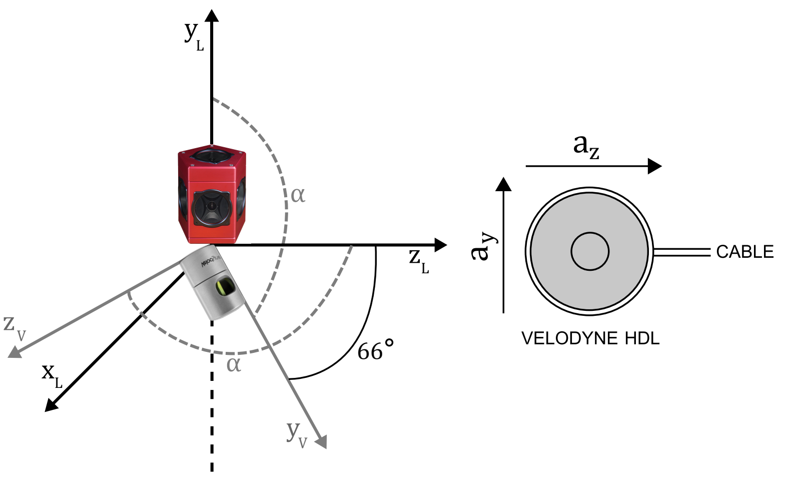


Figure 7.2: Schematic drawing of the rotation performed after nullifying the offset of the sensor origins, i.e. after subtracting the two vectors that project both origins to the ground plane. The angle about the x-axis, i.e. the pitch angle, was computed using the output of the 2-axis accelerometer $\mathbf{a} = (a_y, a_z)$ from the Velodyne HDL. The subscript L denotes the coordinate system of the Ladybug whereas V denotes the coordinate system of the Velodyne.

field vector. In the absence of linear acceleration, the output is a measurement of the rotated gravitational field vector and hence it can be used to determine the accelerometer pitch and roll orientation angles. Specifically, given the accelerometer output $\mathbf{a} = (a_y, a_z)$, cfr. Figure 7.2, the angle about the x-axis, i.e. the pitch angle, can be computed as $\alpha = \mathrm{atan}(a_z/a_y)$. The final transformation $\mathbf{T}^{(vel2lady)} \in \mathbb{R}^{4\times4}$ from the Velodyne coordinate system to the Ladybug coordinate system is then given by:

$$\mathbf{T}^{(vel2lady)} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}_3^\top & 1 \end{array}\right] = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & \cos\alpha & \sin\alpha & t_y \\ 0 & -\sin\alpha & \cos\alpha & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7.1}$$

Herein, the values of $\mathbf{t} = (t_x, t_y, t_z)$ were determined by nullifying the offset of the sensor origins, i.e. subtracting the two vectors that project both origins to the ground plane. In case of the Velodyne coordinate system, the projection of the origin is performed after applying the rotation $\mathbf{R}$.

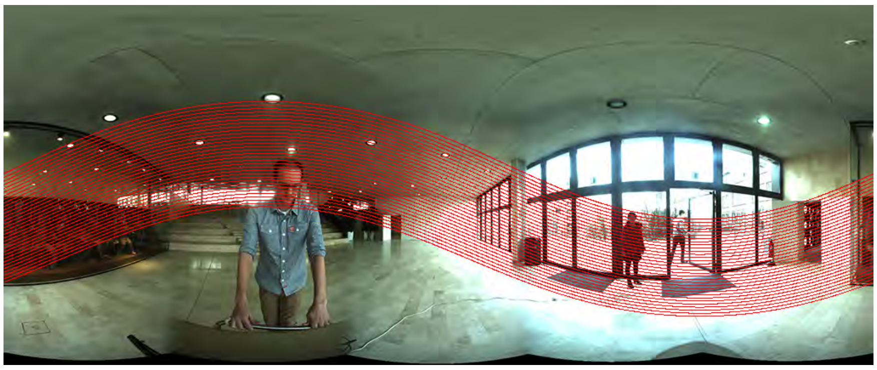


Figure 7.3: The laser points projected on the stitched panoramic image.

7.1.3 UFO building

The first series of sequences were recorded inside the UFO building of Ghent University, in order to obtain an accurate 3D point cloud of the building. Additionally, we conducted a registration of the color images onto the point cloud in order to colorize the points. To map the colors onto the point cloud, we first applied the extrinsic calibration using eq. 7.1. Based on the pinhole camera model, each point in the cloud is subsequently projected on the image plane of each lens, cfr. Figure 7.3. In order to obtain the stitched panoramic image, the stitching algorithm (provided by Ladybug) assumes that all points in the scene are at the same radius from the camera. This assumption forces the real world onto a sphere surrounding the camera, which causes parallax stitching errors. We have set the sphere radius at 20 meters, as most objects in the scene are within this distance, which limited the parallax stitching errors.

As shown in Figure 7.3, the number of pixels in the stitched image does not equal the number of points acquired in one sweep. Moreover, a 3D point is often ‘seen’ from multiple cameras. In that case, a weighted average is used depending on the distance of the pixel location to the center of the image: the further from the center, the lower the weight. Doing so, we cope with the vignetting of the lenses, which causes a reduction in brightness or saturation towards the edges of the image. In Figure 7.4, a part of the reconstructed, colored, 3D point cloud is shown.

In Figure 7.5, some example images and reconstructed 3D models are depicted. The blue line represents the path that was followed. Visually, the reconstructions look accurate, apart from the window reflections causing a part of the interior to be mirrored ‘outside’. Unfortunately, we do not have any ground truth for the trajectories available, nor do we have a reference 3D model of the campus. Therefore, we cannot evaluate the results quantitatively.

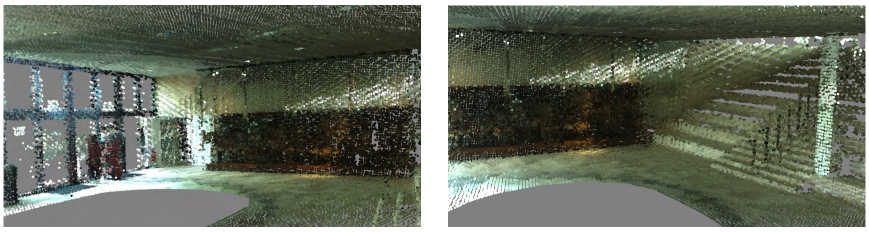


Figure 7.4: A colored 3D point cloud of the UFO building of Ghent University obtained by our 3D reconstruction algorithm using data acquired by the Vellady platform.

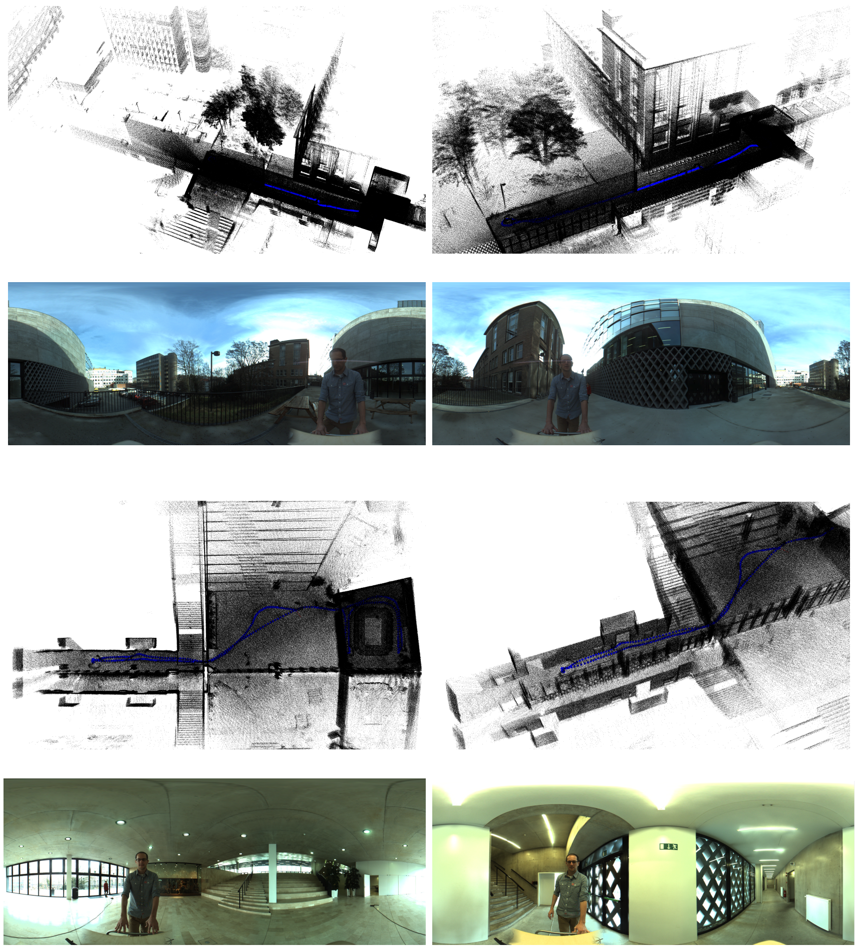


Figure 7.5: Some example images of the 3D reconstructions obtained by applying our algorithm on the recordings of a campus of Ghent University.

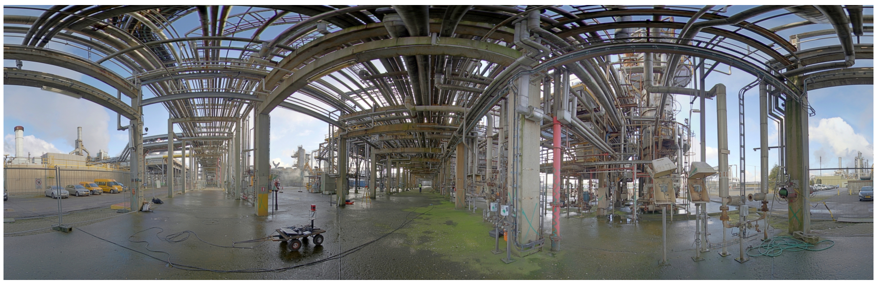


Figure 7.6: Several experiments were conducted at a chemical site of the company DOW in Terneuzen, in the Netherlands.

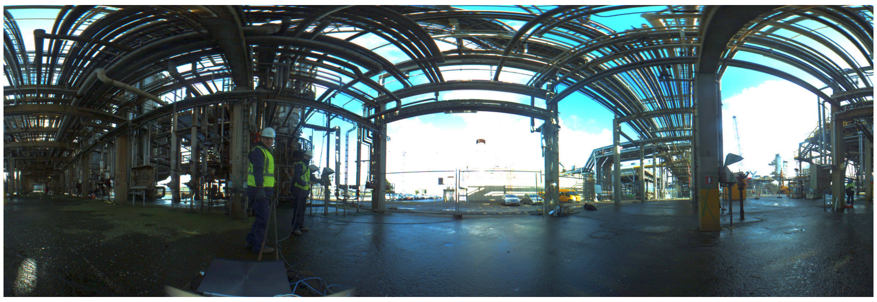


Figure 7.7: An image acquired by stitching the six images of the Ladybug together. It depicts the starting point of a video sequence captured at a chemical site of the company DOW in Terneuzen in the Netherlands.

7.1.4 DOW chemicals

During the GIPA project, a set of sequences was captured at a chemical site of the Dow Company in Terneuzen in the Netherlands. The environment is part of a disused area that is planned to be demolished. It consists of a lot of pipelines that were formerly used to carry liquids or gases, as can be seen in figure 7.6. Although the environment is outdoors, the GPS signal is far too unreliable due to the abundance of pipelines.

The acquired data consists of video sequences recorded with our Vellady platform, which was moving at walking speed, i.e. approximately 4 km/h. A stitched image obtained by the Ladybug camera system is shown in Figure 7.7. The corresponding point cloud is depicted in Figure 7.8. To reconstruct the DoW scene, we used our W-GSS-ICP scan matching algorithm. The geometric stable point sampling, proportioning the observability in all directions, is highly beneficial as the tilt of the scanner causes a lot of points to be sampled on the ground plane, which would bias the transformation estimation towards the identity matrix. Furthermore, we

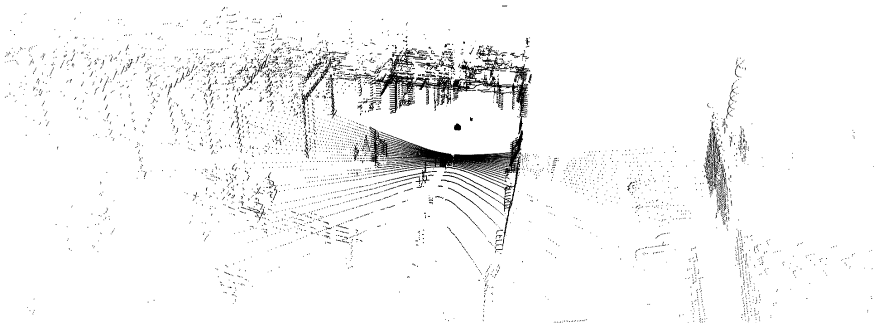


Figure 7.8: A 360° point cloud associated with the stitched Ladybug image of Figure 7.7, captured at the DOW chemical company.

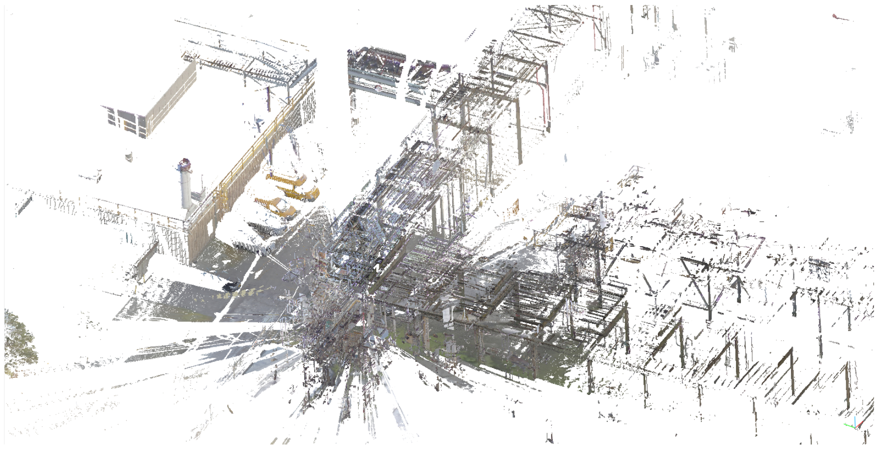


Figure 7.9: Part of the 'ground truth' 3D point cloud of the DOW chemical site acquired using a Leica static terrestrial lidar scanning system.

adopt our feature-based weighting integrated in the IRLS-ICP framework. Finally, we combine the scan-to-scan matching with the scan-to-model matching described in chapter 5.

In order to evaluate this approach on the DoW dataset, an accurate 3D point cloud was acquired by means of static terrestrial laser scanning using a Leica system.[1] We will use this 'Leica point cloud' - depicted in Figure 7.9 - as the ground truth, as we do not have any ground truth for the actual trajectories. For the comparison it is necessary that both point clouds are aligned. Therefore, we first conducted a rough alignment based on key points that we manually selected from both point clouds. Subsequently, we performed ICP to align them in a finer way.

[1]www.leica-geosystems.be

plane	$d(\phi_g, \phi_r)$ in degrees	$d_\mu(H_g, H_r)$ in cm
1	0.76	0.8
2	0.28	0.7
3	1.13	0.8
4	0.29	1.2
5	0.29	1.6
6	0.49	1.1
7	1.81	1.2
8	1.68	0.1
avg.	$\mathbf{0.84}$	$\mathbf{0.9}$

Table 7.2: The difference in angle between two corresponding planes $\phi(H_s, H_t) \triangleq \text{acos}(\mathbf{n}_s \cdot \mathbf{n}_t)$ (in degrees) and the average distance of two corresponding planes $d(H_s, H_t) \triangleq \frac{1}{|H_t|} \sum_{\mathbf{p}_t \in H_t} \langle \mathbf{p}_t, \mathbf{n}_s \rangle - d_s$ (in centimeter) for the eight most dominant planes in the scene.

The final residual of the ICP process, i.e. the Euclidean distance between all closest point pairs, could be considered as a measure for the accuracy of the reconstruction. The experiments demonstrated that the final residual of this ICP process is approximately 2.1 cm for our reconstruction. However, as the closest point pairs do not necessarily represent the same physical points in space, the ICP residual is not the best measure for the evaluation of the accuracy. For that reason we conducted a second evaluation in which we use the dominant planes in the scene to make the comparison.

To that end, we estimated the parameters of the most dominant planes in both the Leica and reconstructed point cloud and determined the corresponding planes between the two. Subsequently, we computed for all inliers of the planes from the reconstructed (source) point cloud the distance to its corresponding plane in the Leica (target) point cloud. We define the average distance of two corresponding planes $H_s \triangleq [\mathbf{n}_s, d_s]$ and $H_t \triangleq [\mathbf{n}_t, d_t]$ as $d(H_s, H_t) \triangleq \frac{1}{|H_t|} \sum_{\mathbf{p}_t \in H_t} \langle \mathbf{p}_t, \mathbf{n}_s \rangle - d_s$. In addition, we define the difference in angle of the two corresponding planes as $\phi(H_s, H_t) \triangleq \text{acos}(\mathbf{n}_s \cdot \mathbf{n}_t)$.

The results for the eight most dominant planes in the scene are summarized in Table 7.2. The table shows that on average, the angle deviation of two corresponding planes is approximately $0.84°$ whereas the average distance between two planes is approximately 9 mm. In Figure 7.10, an image is depicted in which both the ground truth and estimated point cloud are shown after alignment seen from a bird's eye view. Visually, we can see that the walls and edges of both point clouds are overlapping, which demonstrates the accuracy of our 3D reconstruction system. This is best visible in areas for which the ground truth point cloud has a lidar

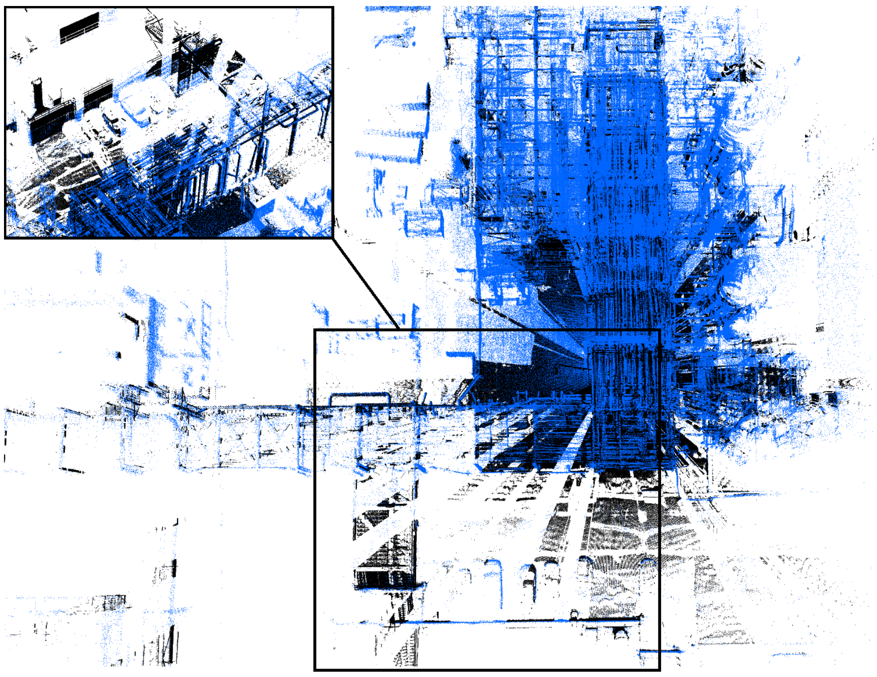


Figure 7.10: Bird's-eye view of the ground truth (black) and reconstructed (blue) point cloud aligned with each other. Visually, one can see that the walls and edges of both point clouds are overlapping, which demonstrates the accuracy of our 3D reconstruction system. This is best visible in areas for which the ground truth point cloud has a *lidar shadow* due to occlusion.

shadow due to occlusion, such as the wall at the bottom of the image.

In Figure 7.11 the same two point clouds are depicted, overlaid on each other, but after they have been projected onto the ground plane to make it easier to evaluate the accuracy. To obtain the image, the ground plane was removed from both point clouds and a quantization in 2D cells was conducted. Brighter areas denote a higher point density in the cell and hence emphasize the overlap of both point clouds. Visually, there are little to no discrepancies noticeable.

The 3D reconstruction from Figure 7.10 was obtained by pose estimation solely based on lidar data, i.e. lidar odometry. In fact, we also have the cameras available and therefore we also estimated the pose of the Vellady system based on *visual odometry* or *structure from motion*. The estimated trajectories are plotted in figure 7.12. Herein, six plots are depicted representing the trajectories obtained by the visual structure from motion approach using SIFT features, presented in [Wu 13] (in red) and the trajectory obtained by our proposed lidar mapping system (in blue).

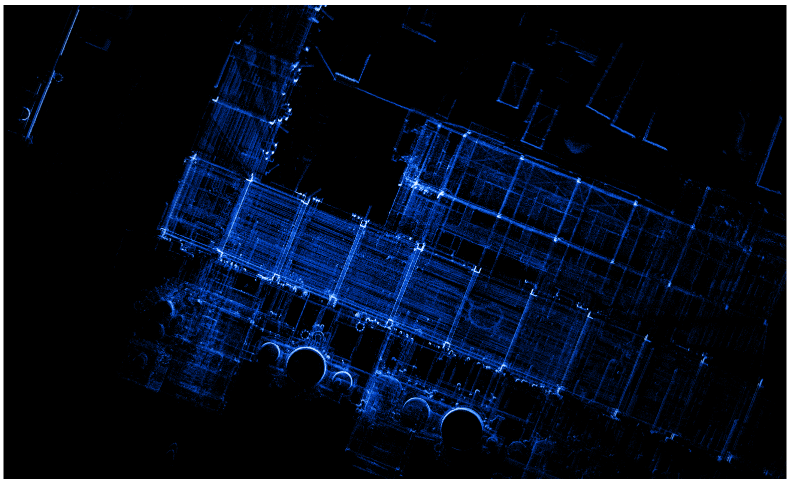


Figure 7.11: 2D projection on the ground plane of both the reconstructed and the 'ground truth' 3D model in order to evaluate the accuracy qualitatively. To obtain the image, the ground plane was removed from both point clouds and a quantization in 2D cells was conducted. Brighter areas denote a higher point density in the cell and hence emphasize the overlap of both point clouds. Visually, one can notice little to no discrepancies as the walls and edges of both models are almost perfectly overlapping.

Each red graph is the result of the visual SfM method run on the output of one camera of the Ladybug system.

As can be clearly seen, for some images (or set of images) the pose could not be estimated, cfr. the gaps in the trajectory. This is due to the fact that sometimes too few good matches could be found between one image and the others. As a consequence, the point cloud resulting from visual SfM is far more sparse than the one obtained by the lidar system. In addition, a lot of *outlier* poses are present. The SfM framework of [Wu 13] is also far from real time. We can, therefore, conclude that the performance of visual SfM is far less compared to what we obtain from our odometry and mapping algorithm using lidar data. The former is less suited to generate accurate 3D models of large-scale environments compared to our mobile mapping solution.

7.2 The Liborg platform

During the previous experiments, the Vellady platform was mounted on a kitchen cart or a four-wheel trolley that was moved manually. The goal is, however, to design a fully automated system, by means of a robot, that can explore and map the environment on its own. For that reason we started the development of Liborg, a

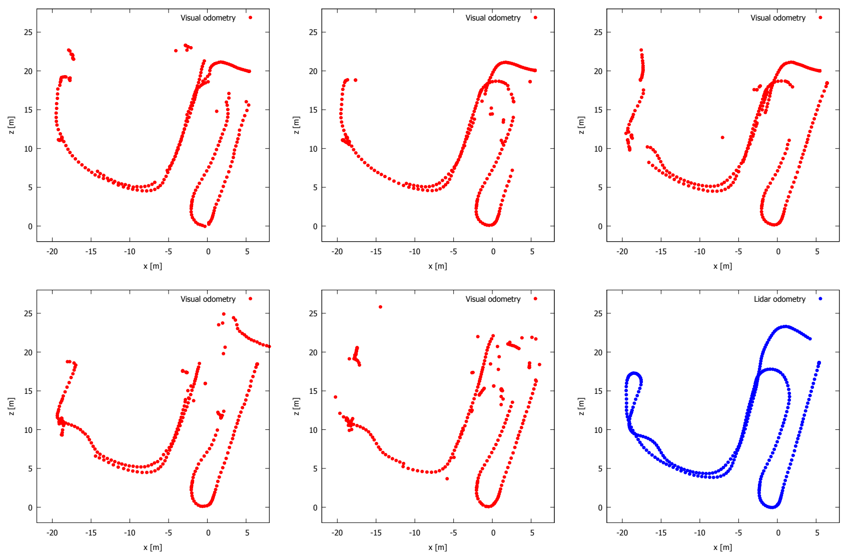


Figure 7.12: Five plots of the trajectories estimated by the visual *structure from motion* (SfM) framework presented in [Wu 13] using the images of each Ladybug camera separately (red color). The result of the camera pointing upwards is omitted. The blue plot represents the trajectory estimated using our method. For the visual SfM approach, some poses could not be estimated due to the lack of good feature matches. Also, a lot of *outlier* poses are present. The visual SfM needs several hours to compute, whereas our system can generate the full 3D model in less than one hour.

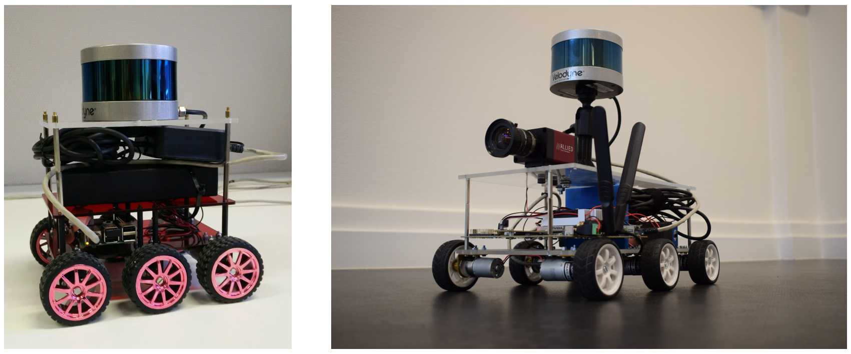


Figure 7.13: Our prototype mapping platform Liborg 1.0 (left) and Liborg 2.0 (right). The first version is controlled by a raspberry pi and is only able to acquire, save and transmit the data. The second version is able to perform the actual 3D mapping online allowing a whole range of additional applications, e.g. live monitoring for inspection purposes. Liborg 2.0 also integrates a RGB-camera.

lidar-equipped robot that can drive autonomously. In order to ease the experiments we also made it possible to control it remotely by ourselves.

The first version of Liborg was running on a simple raspberry-pi computer, which is just powerful enough to control the engines, acquire and save the data and transmit it to a remote server. The actual mapping had to be done on a desktop computer or server. However, in order to make Liborg truly autonomous, we later on replaced the raspberry-pi with an NVidia Jetson TX2. This development board integrates both a CPU and NVidia GPU and is powerful enough to do the 3D mapping real time. The latter allows a whole range of additional applications for which the robot can be used such as live monitoring of the acquired 3D model, for instance to perform inspection or assess damages in areas that are difficult to reach. The two versions of Liborg are depicted in Figure 7.13.

As can be seen, Liborg 2.0 is also equipped with a Allied Vision Prosilica RGB-camera, in view of being able to extend the 3D model with visual information. To fuse the information of the camera and lidar scanner, we developed a synchronisation and calibration method, similar to the one that was developed for the Vellady platform.

In order to evaluate the effectiveness of Liborg, we recorded several sequences at the Telin department, located in the Technicum building of Ghent University. All of them were acquired while the scanner was put under a tilt angle of approximately 30 degrees, in order to capture overhanging structures such as the ceiling. In Figure 7.14, four different 3D reconstruction results are depicted. The results were obtained using our own S-GICP based scan matching technique, combined with the scan-to-map matching integrating surface reconstruction and re-sampling as explained in

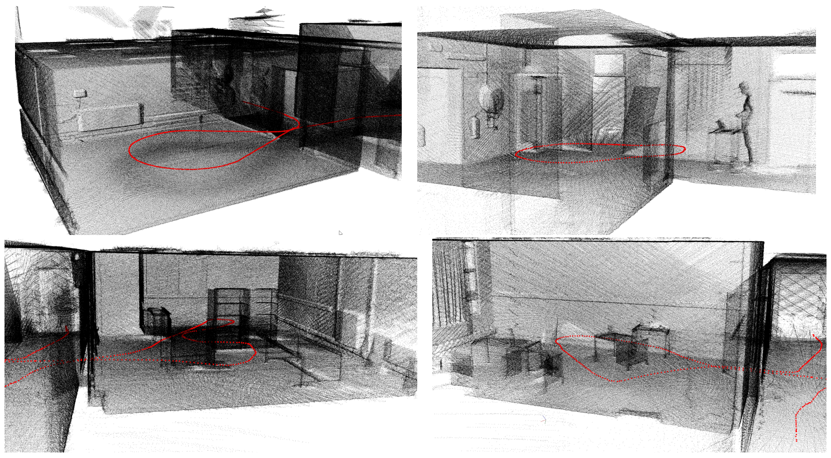


Figure 7.14: Four 3D reconstructions obtained by our Liborg platform with a tilted lidar scanner. Top left: an empty office at the Telin department, top right: the corridor at the Telin department. The red dots denote the trajectory of Liborg.

chapter 5. Unfortunately, we do not have ground truth for the trajectories. Visually, no discrepancies could be noticed

7.3 Sculpture House Gillet

For the final use case, we created a 3D point cloud of the special sculpture house Gillet, located in Angleur, close to Liège, in Belgium. The house is extraordinary as it was entirely built using shotcrete - also known as sprayed concrete - which allowed the creation of walls that are irregularly shaped, as can be seen in Figure 7.15. As a result, the house does not meet the 'Manhattan-world' assumption and lacks any kind of organization in the form of orthogonal and parallel planes. This made it the ideal use case for testing our mobile mapping system in a highly challenging environment.

The actual goal of the 3D reconstruction was to print a miniature 3D model of the house in order to expose it at an architecture exhibition. As the house consists of steps and bumpy parts on the floor, it was not feasible to use Liborg. We thus held the Velodyne VLP-16 scanner in our hand while walking through the house, as is depicted in the bottom left image of Figure 7.15. This way of working yielded another difficulty as the trajectory of the lidar scanner was far more jaggy while walking compared to the smooth trajectories usually resulting from Liborg.

The final 3D point cloud and trajectory are depicted in Figure 7.16. The 3D reconstruction is the result of our surface-based GICP scan matching technique,

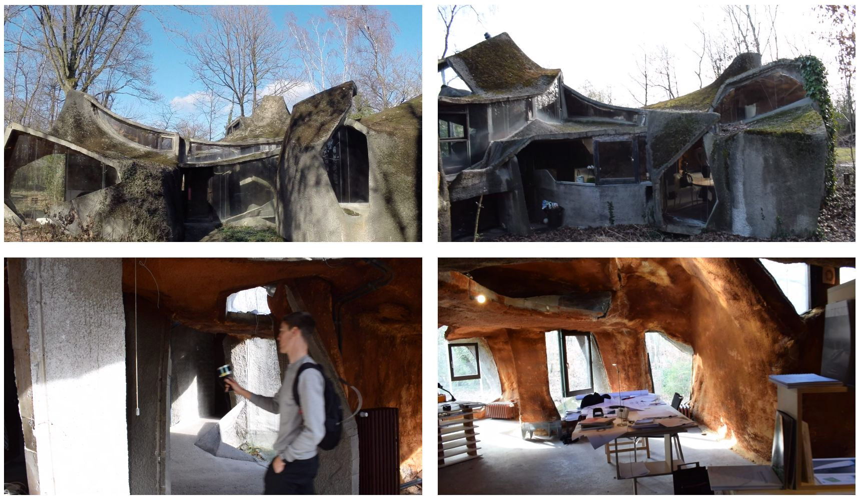


Figure 7.15: The extraordinary shaped sculpture house Gillet in Angleur. The house was built using shotcrete, a.k.a. sprayed concrete, which allows the walls to be irregularly shaped. The lidar scanner was hand-held during acquisition because the floor had too many irregularities for Liborg to be able to move around.

combined with scan-to-model matching integrating surface reconstruction and resampling. In Figure 7.17, two cross sections of the model are depicted, which show the difference in elevation of the house. Unfortunately, we do not have any ground truth model, nor can we compare with a different type of odometry. It is, however, hard to see any artefacts in the 3D model, which demonstrates the effectiveness of our system.

7.4 Conclusion

In this chapter we presented a few use cases that show the effectiveness and usability of our mobile 3D mapping system. In an early stage, we developed the Vellady platform consisting of a Velodyne HDL-32e lidar scanner and a Ladybug panoramic camera system. Back then, we used the Vellady in combination with a kitchen cart or four-wheel trolley to create the 3D reconstructions. Later on, we developed the Liborg robot, which is able to drive autonomously but that can also be controlled using a remote control. The first version of Liborg was running on a Raspberry-pi computer, which was just able to collect the data and stream it to a server. The second version of Liborg has an NVidia Jetson TX2 integrated and is able to perform live mapping on the embedded platform itself.

We further demonstrated that our 3D mapping system is able to produce ac-

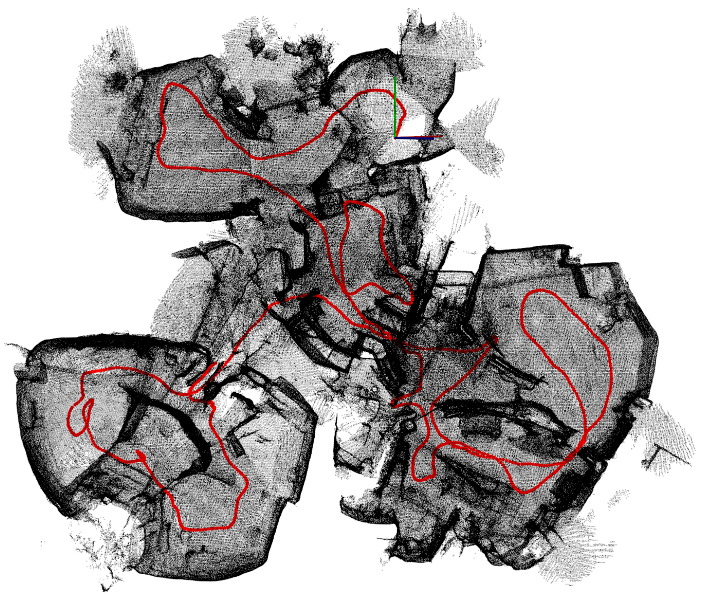


Figure 7.16: Bird's-eye view of the 3D reconstruction of the Sculpture House Gillet in Angleur. The red line represents the trajectory of the lidar scanner.

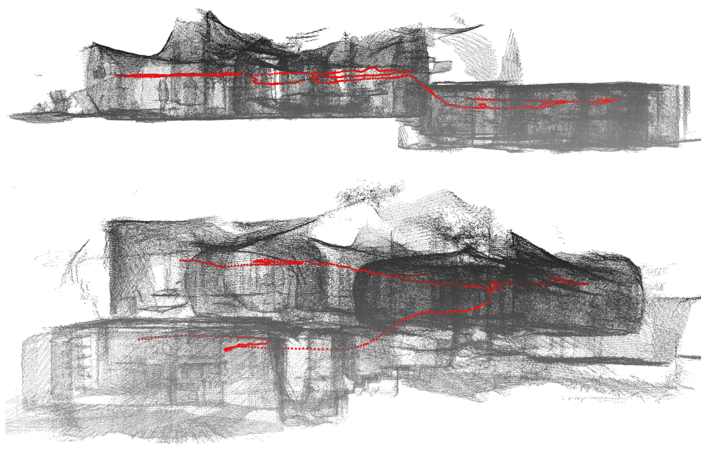


Figure 7.17: Two intersections of the 3D reconstruction of the Sculpture House Gillet in Angleur. The red line represents the trajectory of the lidar scanner.

curate 3D reconstructions in different kinds of environments, going from typical 'Manhattan world' scenes recorded in the buildings of our department to a complex industrial site with lots of overhanging pipelines as well as an extra-ordinary shaped sculpture house located in Angleur.

Throughout this PhD we developed several slightly different algorithms that are each beneficial in a specific environment. For the indoor scenes like the UFO building and the sculpture house Gillet we used our S-GICP scan matching method, presented in chapter 3. For the mapping of the chemical plant, for which the point clouds were very sparse, we used our W-GSS-ICP scan matching method integrating both geometrical stable point sampling and weighting based on low-level shape features. Finally, we also showed that our algorithm is able to cope with data captured with a tilted scanner limiting the horizontal field of view.

8 Localization and mapping using depth cameras

In the previous chapters we mainly focused on 3D mapping using data acquired from lidar scanners. For some applications, however, a lidar scanner is not feasible, for example due to its size and weight. This chapter will deal with *projective* depth devices or depth cameras based on either ToF or structured light, sometimes accompanied by a regular camera, referred to as RGB-D cameras in that case. Those sensors are more appropriate for user-based applications, e.g. in the domain of augmented reality or the guidance of visually impaired people, as they are more lightweight and usually a lot cheaper. The main difference between these depth cameras and lidar scanners is their lower accuracy and shorter range. The latter is of the order of magnitude of a few meters, up to 4 or 5, compared to 100 meters or more for the more expensive lidar scanners. For user-based applications in the domain of AR this is not a major problem as the operation range there is usually a lot smaller. On the other hand, projective depth devices have a higher resolution, which is beneficial for AR-applications to obtain a more dense 3D model of the workspace.

8.1 Introduction

More and more augmented reality (AR) applications are using head-mounted devices to give users a fully *immersive* experience. Among the examples are the

HoloLens and its successor, the HoloLens 2, introduced by Microsoft in 2015 and 2019 or the Meta 2[1] released in 2017. One of the main prerequisites for these AR-devices to fully operate is the precise tracking of the user's head at all times. This is necessary to find out what the person is looking at and to eventually augment his view with virtual objects. Current AR-devices achieve this tracking by using accelerometers and gyroscopes. However, these instruments are either too expensive or accumulate too much drift over time, causing virtual objects to be shifted from their 'initial' - let's say intended - position.

As a result, developers of head mounted devices are trying to use cameras in order to apply visual SLAM to track the head. Many visual tracking solutions have been presented in the literature so far, but none of them have been really successful. The main reason is that current SLAM methods often lack robustness, especially in case of abrupt head motion. They are also subject to subtle changes in lighting conditions or shadows and heavily rely on the presence of textured objects in the scene. As a result, many applications still use artificial markers that are put onto the scene instead of using *natural landmark* tracking or full-blown SLAM solutions. However, this way of working is not very scalable.

With the advent of affordable depth cameras, more and more researchers explored ways to integrate depth information in the tracking process. Using depth cameras has clear advantages: they are tolerant to common problems that appear in monocular camera tracking, including changes in illumination, repetitive textures and lack of features. However, the available RGB-D SLAM solutions are still limited in terms of accuracy and robustness. The main problem with existing techniques is that they are usually only incorporating scan-to-scan matching instead of integrating the entire 3D (point cloud) map being built. The main reason is that scan-to-model matching is often too time-consuming - especially in case the 3D map has become too large - making real-time execution impossible.

Another issue is that in AR we often want to know the precise location and orientation of the person related to an existing 3D model in order to be able to fully interact with it. This is especially the case for industrial applications, where tasks have to be performed based on specific components of machinery. In addition, this 'ground truth' 3D model from the database can also be used to relocate the camera and to cope with drift. The latter brings us to another benefit of integrating depth cameras: it is far better suited to estimate the pose related to an existing 3D model compared to regular cameras. In industrial companies, existing BIM or annotated CAD models of plants or machinery are often available. These models usually consist of additional semantic information or even instructions about some components that could be overlaid in the view of the user. Linking them with the estimated pose of the user, allows for AR-applications such as augmented assembly

[1] http://www.metavision.com/

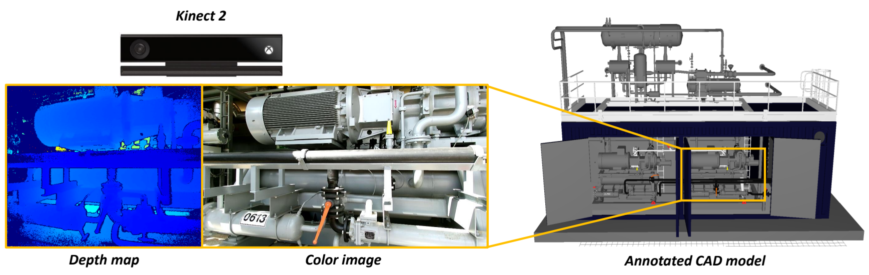


Figure 8.1: Sometimes a model of the scene is already present by means of a (annotated) CAD or BIM model. It can be used to relate the pose of a ToF sensor such as a Kinect or HoloLens in order to localize the user in the scene, for example in case of AR applications. Left: a depth map and color image acquired with a Kinect. Right: a BIM model of a cooling container to which we want to map the user's pose.

or augmented maintenance of machinery.

In this chapter, we will propose a dual solution, in which we combine the registration of consecutive point clouds as well as a model-based approach for the fight against drift. The latter can be performed using an available 'ground truth' model but also using the 3D model that is being built on the fly (similar to the algorithms proposed in chapter 5. The former makes it possible to link with a 3D model from a database and hence to relate the view of the user to what is known about this part of the scene as is depicted in Figure 8.1. The model-based tracking can be conducted at a lower frequency to keep the processing time limited. Doing so, we keep the best of two worlds by being both fast, robust and being able to cope with drift. Regarding the model-based tracking, we will use the same surface reconstruction and re-sampling process described in chapter 5. In addition, we apply the multi-resolution registration scheme of chapter 5 to speed-up the pose estimation.

The main application that we focused on is that of an *engineering coach* in which a layman is guided towards the completion of a particular task without having the knowledge to do so. The precise instructions are provided to the layman by means of visual cues. The task the layman has to deal with in our application is the replacement of an oil filter of a cooling machine. In Figure 8.2, an image is depicted of the working environment (left) and the layman performing the task (right).

8.2 Related work

As our solution consists of two different parts, we will split this related work section into two subsections. The first one will deal with the RGBD-SLAM literature

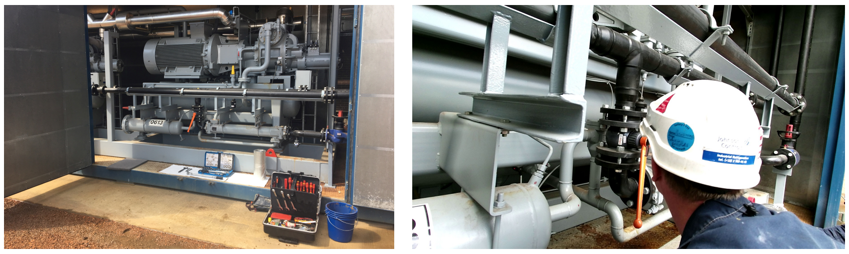


Figure 8.2: The cooling container that was the subject of an AR-applications supporting its maintenance. Maintenance instructions can be virtually attached to an object of interest in the real world.

whereas the second one will deal with the registration of a point cloud derived from a depth camera with an existing CAD-based model from the scene.

8.2.1 RGBD-SLAM

Many of the current methods for *handheld* or user-based SLAM are using regular cameras. They either use features such as ORB-SLAM [MurArtal 15], Fast Odometry from Vision Systems (FOVIS) [Huang 11] and PTAM [Klein 07] or they adopt a *direct* approach by registering two consecutive images directly upon each other by minimizing a photometric error such as LSD-SLAM [Engel 14] or the work of [Kerl 13a; Kerl 13b].

As mentioned in the introduction, all of these systems lack robustness as they are subject to subtle changes in lighting conditions or shadows and heavily rely on the presence of textured objects in the scene. With the advent of cheap RGB-D sensors such as the Kinect, many researchers have started to focus on the combination of RGB-SLAM with the output of an accompanied depth camera [Endres 12; Sturm 12; Kerl 13a].

In [Endres 12], for example, the authors compute ORB keypoints in the image and determine the corresponding 3D point using the accompanied depth map. Thanks to the full 3D position, they can efficiently avoid outlier matches by refusing the ones for which the Euclidean distance in 3D space is too large. Moreover, they can use the knowledge of a point's 3D position to compute the transformation between two sensor poses instead of relying on triangulation.

ICP-based

We consider the system *"KinectFusion"* presented in [Izadi 11] to be the true milestone in RGB-D SLAM as it was the first solution for projective devices integrating an ICP-based point cloud alignment that runs in real-time. However, due to this real-time constraint, the initial system also came with a few limitations, mainly related to its lack of robustness in case of moving objects or abrupt movements of the camera. Therefore, many researches have proposed improvements over the initial algorithm, some of them incorporating visual odometry to enhance the performance in geometrically deprived environments.

In [Henry 12], the authors propose RGB-D ICP, in which visual features and their associated depth values are used to obtain an initial alignment. The final transformation is then determined by jointly optimizing over the sparse feature matches and dense point cloud correspondences. A similar approach was followed by [Whelan 13], in which depth-based ICP is combined with FOVIS visual odometry. The authors propose a switching strategy in which the ICP estimate is favoured in case of geometrically rich scenes and the FOVIS estimate is favoured in case of monotone planar environments.

The same authors also 'spatially' extended the original KinectFusion system [Whelan 12], as the latter was bounded by its initial *truncated signed distance function* (TSDF) volume, a volumetric representation of the scene where each location stores the distance to the closest surface. The system of [Whelan 12] maintains, at any point in time, a TSDF of the region of space that is currently being mapped. However, the region represented by this TSDF varies dynamically during processing and when new regions enter the TSDF, previously mapped ones are extracted into a more parsimonious triangular mesh representation.

Apart from integrating visual odometry to improve the robustness of the tracking, other work has focused towards making ICP more robust itself. In [Serafin 15b], the authors determine the transformation by minimizing the distance between the corresponding features instead of minimizing the point-to-plane distance between corresponding points as in standard ICP. More specifically, they minimize the Mahalanobis distance of each corresponding point pair and their normals, resulting in minimizing a distance among 6D vectors instead of 3D points.

3D keypoints

The ICP-based methods from the previous section have at their core the alignment of consecutive scans by minimizing distance measures between all (or the majority) of the points. Just like with scan matching for spinning lidar sensors, an alternative approach is the extraction and matching of (sparse) 3D features for use in RGB-D SLAM applications. One advantage over spinning lidar data is that the point clouds generated from projective devices are in general less sparse, which makes it more suitable to construct more sophisticated keypoint detectors.

Several papers have proposed such 3D keypoint detectors, including the intrinsic shape signatures (ISS) [Zhong 09], Harris 3D [Sipiran 10b], Sift 3D [Scovanner 07] and NARF [Steder 10]. Others have proposed descriptors such as spin images [Johnson 97] or the SHOT descriptor [Salti 14]. However, none of these keypoint detectors and descriptors have been widely used to conduct real-time scan matching as they are in general too slow and they suffer from the lack of high repeatability and descriptive power.

Deep learning

To deal with the shortcomings of hand-crafted features, many researchers shifted towards the use of deep local feature descriptors obtained by a neural network, for example *MatchNet* [Han 15], DeepCompare [Zagoruyko 15], *3DMatch* [Zeng 17b], DeepCD [Yang 17] or the work of [He 18]. The authors of [Kang 19] eventually used those deep local feature descriptors to build their full-blown *DF-SLAM* system.

Another way of incorporating neural networks is to let them predict depth data from the images themselves. Recently, the performance of depth prediction from images is rising strongly and as a result, more SLAM researches are focusing on combining them with direct monocular SLAM, hence ‘avoiding’ the need to use an accompanied depth sensor [Tateno 17].

Plane-based

Finally, also for projective devices, some plane-based SLAM solutions have been proposed. In those works, the authors exploit the Manhattan world assumption, in which a scene consists of large vertical planes that are perpendicular to the floor and ceiling and in which many are parallel to each other. Examples are the *Dense Planar SLAM* method of [SalasMoreno 14] or the work of [Le 17]. As mentioned before, the applicability of these methods is limited to indoor scenes with a lot of structure.

8.2.2 Model-based alignment

Regarding model-based alignment, the academic literature is much more limited, especially when it comes down to the registration of point clouds from noisy depth cameras. In [Kim 13], a system was designed to automatically register 3D CAD models with 3D point cloud data from a construction site. The authors propose to use principal component analysis (PCA) to find a coarse registration and subsequently refine the alignment using ICP. The main difference with our use case is, however, that the point clouds will never comprise the whole CAD model, so in our situation PCA is not applicable.

In [Corsini 13], the authors present another fully automatic registration framework to align image sets with 'approximate geometry', which can be seen as a CAD model. To this end, they adopt the 4-points congruent sets (4-PCS) method [Aiger 08], based on the matching of 4-points coplanar bases using the *generate-and-test* paradigm. They propose a scale independent version of 4-PCS by introducing two modifications. The first modification is a preprocessing step to overcome the difference in sampling between the CAD model and the point cloud that was obtained by structure from motion. The main idea is to partition the two point clouds into a set of quasi planar regions and then re-sample the clouds uniformly with respect to their area. The second modification consists of a rasterization based algorithm that considerably reduces the time needed to test candidate transformations.

In [Chen 16], the authors also tried to match a point cloud acquired using structure from motion with a CAD model. The main idea of their method also comprises a 'sampling' of the CAD model, which they denote as a *virtual scan*. To overcome the density difference between the sampled model and the 'SfM point cloud' they propose a coarse-to-fine down-sampling strategy on the SfM point cloud.

The aforementioned methods were developed in view of aligning a large point cloud obtained by SfM with a CAD model, which is different from aligning a single image or depth map from a depth camera with a CAD model in view of AR applications. A few studies focused on localization through the alignment of regular RGB images with a (CAD) model [Lepetit 05; Bleser 05; Wuest 05; Klein 06]. Those model-based tracking methods try to fit features, such as edges, extracted from the camera image to 2D projections of the 3D model of the reference target to estimate the transformation between them. However, as mentioned in the previous section, regular images often lack detectable 2D features, making it very difficult to estimate the camera pose. This is especially the case when the captured scene has untextured monochromatic surfaces or when the lighting conditions are bad. Strong shadows can be indistinguishable from actual edges and reflections or dim illumination can disturb the feature detection.

Motivated by these shortcomings, the authors of [Korkalo 16] present a method for tracking a depth camera with an existing CAD model. The main idea of their approach is to construct a 3D point cloud from the latest incoming raw depth frame and align it with a point cloud that they generate from the reference model using the sensor intrinsic and extrinsic parameters from the previous time step. The actual alignment is done using the ICP algorithm. In our work, we will use a similar strategy, but we will adopt the multi-resolution ICP registration scheme from chapter 5 to guarantee that the tracking can be conducted in real-time.

8.3 Proposed approach

In this section we will propose a depth-based system that consists of two main parts: 1) initial pose estimation based on a CAD model and 2) camera tracking using point cloud registration. Regarding the initial pose estimation, we start from the initial idea proposed by the authors of the 4-PCS method [Aiger 08; Mellado 14]. As opposed to the methods of [Corsini 13] and [Chen 16] we do not sample the CAD model to determine the initial pose. Instead we exploit the knowledge of adjacent triangles in the mesh-based model to determine 4-points sets that are coplanar. The latter is an easy operation as it only involves a check on the direction of the normal vectors of two adjacent triangles. The resulting 4-points sets are then used as coplanar bases. Using this strategy, we avoid the randomized base selection process of the original 4-PCS method, which makes our method fully deterministic.

8.3.1 Initial pose estimation

Finding the initial pose of the camera relative to a CAD model using data from depth cameras is a challenging task as this data typically contains a lot of noise. As a result, extracting low-level geometrical features is not trivial, let alone matching them with features derived from the CAD model. A CAD model is in fact the opposite as it contains no noise and is usually pretty accurate. On the other hand, a CAD model lacks detail, but the model itself is already a set of geometrical primitives which can be exploited to easily align it with the point cloud derived from the depth camera.

The main idea of our algorithm is to exploit the 4-points congruent sets (4-PCS) properties, which we also used to align two lidar point clouds in view of loop detection, cfr. chapter 6. To summarize, the 4-PCS method is a global point cloud registration technique that is based on the matching of congruent sets in two point clouds. It uses the same *generate-and-test* paradigm known from RANSAC-based solutions, but instead of exhaustively testing all the bases from a source point cloud P, it exploits *planar congruent sets* to select only a small subset of bases from P that can potentially match a given base from a target point cloud Q.

Our final algorithm consists of the following steps: 1) selecting a coplanar base in the CAD-model, 2) finding the (approximate) congruent sets in the point cloud derived from the depth camera, 3) computing and testing the rigid transformations and selecting the best one.

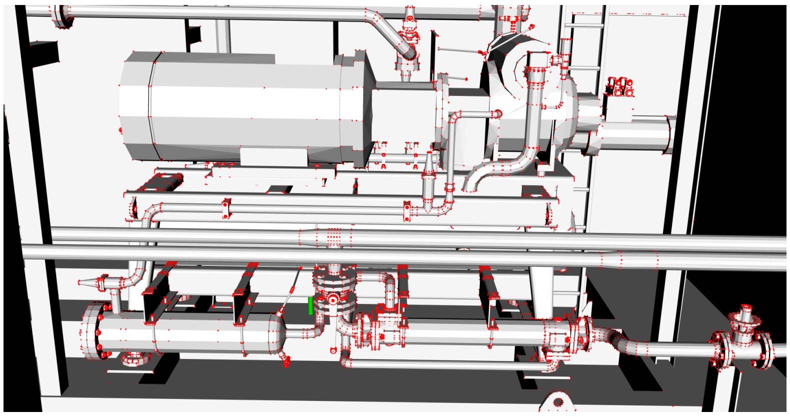


Figure 8.3: A part of the CAD model of Figure 8.1, which is defined as a triangular mesh. The vertices of the CAD model, shown by red dots, are used in our 4-PCS method to select coplanar bases consisting of 4 points that are not all collinear.

8.3.1.1 Selecting a coplanar base

As we are dealing with a CAD model as target instead of an acquired point cloud, we can extract coplanar points more easily as they are represented by the vertices of planar patches. One way of selecting coplanar bases is to consider adjacent triangles and determine whether their normal vectors point in the same direction. Doing so, we omit the randomized base selection process of the original 4-PCS method. We can test all the bases found this way but as wide bases are leading to more stable alignments we prioritize those. Thereby, we verify whether they are still lying in the overlap region between the point cloud and the CAD model in order not to miss the desired solution. In Figure 8.3 all the vertices are drawn for a part of the CAD model shown in Figure 8.1.

8.3.1.2 Finding congruent sets

This step is the main component of the algorithm and is based on affine invariants of 4-points sets. As this step was already detailed in chapter 6, we will briefly summarize it here.

Suppose we have a set of coplanar points $\mathbf{B} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ from the CAD model that are not all collinear, cfr. Figure 8.4. Let $\mathbf{ab}$ and $\mathbf{cd}$ be the two lines that intersect at an *intermediate* point $\mathbf{e}$. Recall from chapter 6 that the two ratios, $r_1 = ||\mathbf{a} - \mathbf{e}||/||\mathbf{a} - \mathbf{b}||$ and $r_2 = ||\mathbf{c} - \mathbf{e}||/||\mathbf{c} - \mathbf{d}||$ are invariant under affine transformation, and uniquely define 4-points up to affine transformations. For each point $\mathbf{p}_1, \mathbf{p}_2 \in P$, we compute two *intermediate* points $\mathbf{e}_1 = \mathbf{p}_1 + r_1(\mathbf{p}_2 - \mathbf{p}_1)$

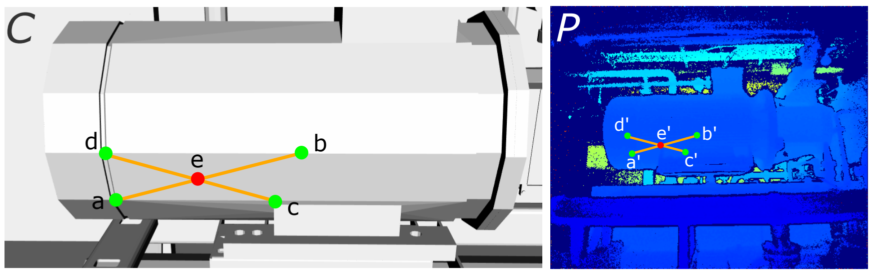


Figure 8.4: A 4-points coplanar base $\mathbf{B} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ selected from the CAD model C (left) and the corresponding congruent set $\mathbf{B}' = \{\mathbf{a}', \mathbf{b}', \mathbf{c}', \mathbf{d}'\}$ from the point cloud (right). The ratios $r_1 = ||\mathbf{a} - \mathbf{e}||/||\mathbf{a} - \mathbf{b}||$ and $r_2 = ||\mathbf{c} - \mathbf{e}||/||\mathbf{c} - \mathbf{d}||$ are preserved under affine transformation.

and $\mathbf{e}_2 = \mathbf{p}_1 + r_2(\mathbf{p}_2 - \mathbf{p}_1)$. When the intermediate points $\mathbf{e}_1$ and $\mathbf{e}_2$ of two point pairs are coincident, they potentially correspond to a 4-points set that is an affine transformed copy of $\mathbf{B}$. The set of 4-points sets are the affine invariants of the base $\mathbf{B}$, so it is a superset of the 4-points sets that are a rigid transformation of the base. Just like in chapter 6, we check the angle between the two line segments to determine if the 4-points set is a rigid transformation of the base $\mathbf{B}$.

Figure 8.4 demonstrates the main idea of the algorithm: a potential coplanar base in the CAD model (left) is selected and its corresponding congruent set in the point cloud (right) - here depicted as a depth map - is determined. The procedure can be summarized as follows: given a selected coplanar base from the CAD model, determine all 4-points sets in the point cloud P that are approximately congruent. This means that the distance of the two intermediate points $\mathbf{e}_1$ and $\mathbf{e}_2$ should be close to each other. Because of noise and inaccuracies, we use a threshold Δ to decide whether the two points $\mathbf{e}_1$ and $\mathbf{e}_2$ are coincident.

8.3.1.3 Computing the best rigid transformation

In the original 4-PCS method, the largest common point set (LCP) criterion is used to verify the estimated transformation and to determine the ‘best one’. This LCP is determined by counting the number of points from the *source* point cloud that are within a predefined distance Δ from any point in the target point cloud, i.e. the CAD model in this case, after alignment. However, the number of vertices in this CAD-model is too small to reliably check whether the transformation is a good one or not. A better way, therefore, is to compute for all the points that are part of the plane the distance to its corresponding plane in the CAD model and consider the sum of all distances as the criterion. The transformation leading to the smallest sum can then be selected as the ‘best’ transformation.

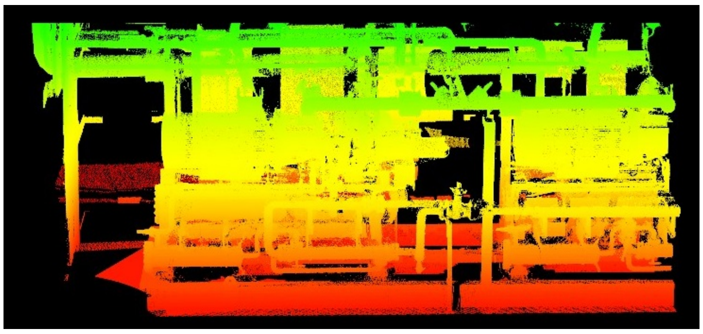


Figure 8.5: A point cloud sampled from the CAD model of Figure 8.3.

Alternatively, the CAD model can be sampled in order to obtain a dense point cloud model. Even though this method is more complex, it allows to use the ICP algorithm to refine the transformation. After applying ICP, we obtain the 'closest' points between both point clouds (the scan and the ground truth). The final residual, i.e. the sum of Euclidean distances between all corresponding points serves as an additional verification for the correct alignment. Figure 8.5 depicts a point cloud obtained by sampling the CAD model from Figure 8.3.

8.3.2 Camera tracking

Once the initial pose of the user has been related to the CAD model, the next step is to track the movements of the user. Our camera tracking consists of three main steps. First, we conduct a preprocessing step to convert the depth measurements from the depth camera to an organized point cloud and to lower the noise. In addition, we compute a *surfel* representation from the point cloud by estimating several local features, including the normal vector for each point in the point cloud, cfr. chapter 4. The second step deals with the alignment of consecutive point clouds, which gives us an initial guess for the current pose. Third, we align the point cloud derived from the depth camera with the point cloud of the scene. The latter can either be the one that is built on-the-fly, or it can comprise the CAD model that has been sampled, if available. The main idea of our scan-to-model alignment is to derive a multi-resolution representation of the model and to adopt a coarse-to-fine registration, similar to what we have proposed in chapter 5.

8.3.2.1 Preprocessing

Low-cost depth cameras such as the Kinect have a limited accuracy and the point clouds acquired by them usually contain a lot of noise. In order to cope with this, we apply the surface approximation algorithm from chapter 5. This algorithm estimates the underlying surface, which allows us to re-sample the points in the point cloud. Recall that our algorithm estimates for each point a second order polynomial through the distances of its neighbours to its tangent plane. Therefore, we first need to estimate the surface normals for each point, for which we use the same approach as in chapter 4 by first computing an optimal neighbourhood set for each point. This procedure gives us for each point $\mathbf{p}_k$ a set of neighbours $\mathbf{N}_k$ that we use to compute the normal vector $\mathbf{n}_k$ and thus the tangent plane defined as $H_k \triangleq [\mathbf{n}_k, d_k]$ in which d_k represents the distance to the origin.

For all points lying in $\mathbf{N}_k$, we compute the distance to this tangent plane and we fit a polynomial in this set of distances. This second degree polynomial $\tilde{p}_k$ is estimated by minimizing the following weighted least-squares error:

$$\tilde{p}_k = \underset{p_k}{\operatorname{argmin}} \sum_{\mathbf{p}_i \in \mathbf{N}_k} (p_k(\mathbf{x}_i) - f_i)^2 \theta(||\mathbf{p}_i - \mathbf{p}_k||), \tag{8.1}$$

where $\mathbf{x}_i$ is the orthogonal projection of the point $\mathbf{p}_i$ onto the tangent plane and $f_i \triangleq \langle \mathbf{p}_i, \mathbf{n} \rangle - d$ is the distance of $\mathbf{p}_i$ to the tangent plane. Finally, $\theta(x) = e^{-(\frac{x}{\sigma_r})^2}$ represents the weighting function that is based on the distances to the tangent plane and the average separation σ_r of the 3D points.

Once the parameters of the polynomials $\tilde{p}_k$ are known, we project the original points on this surface. This procedure manipulates the points in such a way that the noise is reduced and also allows to fill small gaps in the point cloud. For the latter, we first define a voxel grid upon the point cloud, which we dilate with new points and the center positions. Subsequently, we project the newly created points to the approximated surface of the closest point in the point cloud.

8.3.2.2 Scan-to-scan alignment

In chapter 3 we described several scan matching techniques to align two point clouds that are slightly shifted. The two most prominent ones are the normal distribution transform (NDT) and the ICP algorithm. Regarding ICP, we evaluated many cost functions including the point-to-point, point-to-plane and plane-to-plane cost function. We also proposed our own S-GICP algorithm. For AR-applications, it is important that the scan matching is fast enough. Chapter 3 revealed that the point-to-plane cost function is the fastest among them and therefore we will adopt that one to conduct the scan-to-scan matching here:

$$E(\mathbf{R}, \mathbf{t}; \mathcal{P}^s, \mathcal{P}^t) = \sum_{i=1}^{N} \mathbf{w}_i((\mathbf{R}\mathbf{p}_i^s + \mathbf{t} - \mathbf{p}_i^t) \cdot \mathbf{n}_i^t)^2. \tag{8.2}$$

In this equation, $\mathcal{P}^s$ and $\mathcal{P}^t$ are respectively the *source* and *target* point cloud, which are indexed according to their N correspondences. In this case, the target and source point cloud are respectively the current and previous one acquired by the depth camera. Finally, $\mathbf{n}^t$ represents the normal vector of target point $\mathbf{p}^t$.

Usually, the closest point correspondences are determined using the Euclidean norm in 3D space. However, in chapter 3 we learnt that due to the non-uniform and sparse point density, it is unlikely that exactly the same physical point is sampled in two consecutive point clouds. This so called correspondence ambiguity problem is less severe for point clouds derived from depth cameras compared to point clouds acquired from laser scanners, but it is still present and causes the point correspondences to be inaccurate. Therefore, we again propose to introduce a weight for every corresponding pair, which is determined using the feature distance as explained in chapter 4. The actual weight is computed using the Beaton-Tukey estimator defined in eq. 4.13, and is updated in every iteration in order to filter out inaccurate correspondences, hence denoted as an iteratively re-weighted least squares (IRLS) ICP solution.

8.3.2.3 Scan-to-model alignment

The scan-to-scan ICP described in the previous section provides an initial estimate on the camera pose. As this estimation is subject to drift we perform an additional alignment step using the 3D model of the scene. This model can either be the one that is being built or the one that is derived from the CAD model.

The former one is resulting from fusing all the acquired 3D points into one aggregated map. However, if we would store all the points that were captured by the depth camera, the map would quickly contain billions of points. It would, therefore, become intractable to determine corresponding points between the point cloud derived from the depth camera and the one derived from the 3D model. For that reason, we convert the model into an octree-based data structure as we did in chapter 5. Recall that this octree comprises a hierarchical space partitioning, subdividing the 3D space recursively into 8 smaller sub cubes of the same size. At the leaf level, only one point is kept, which is computed using an approximation of the underlying surface, as follows. First, we select the closest points of the centroid of the (leaf) voxel, which can be easily determined based on the space partitioning of the octree. Second, using this neighbour set we estimate the underlying surface using the same surface approximation algorithm described in chapter 5. Finally, we project the centroid point on the estimated surface. Apart from reducing the noise, this process allows us to extract point clouds with a lower point density.

Following this procedure, we can select multiple point clouds with different resolutions and perform the ICP algorithm in a coarse-to-fine manner like we did in chapter 5. The octree data structure thereby alleviates the search for approximate nearest neighbours as we only have to traverse the tree to the leaf voxel, leading to

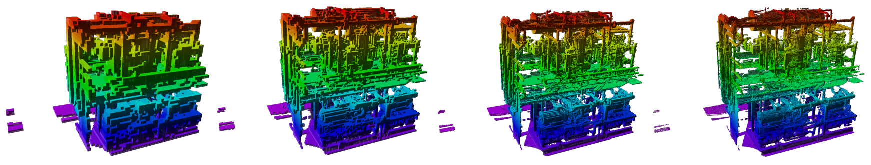


Figure 8.6: Several resolution levels of the cooling machine from Figure 8.1. From left to right, the voxel size is respectively 0.32, 0.16, 0.08 and 0.04 meter corresponding to resolution level 11 to 14. This multi-resolution representation is the core of our scan-to-model matching, which allows real-time camera tracking.

$O(\log n)$ time complexity.

In case a 3D model from the scene is available, we skip the map building part of the algorithm and we use the available model instead. If the latter is available in the form of a CAD-model, we first sample it in order to be able to conduct ICP. To that end, we again build an octree data structure on top of the CAD-model in order to be able to extract point cloud models of different resolutions. In Figure 8.6, the same cooling machine as in Figure 8.1 is depicted for five different resolutions with a leaf size ranging from 0.32 to 0.04 cm.

8.4 Evaluation

In order to evaluate our method, we use the video sequences that are part of the Freiburg RGBD-SLAM dataset, presented in [Sturm 12]. All of the sequences are recorded with the Kinect and have both a color video stream and a depth video stream. The sequences come with accurate ground truth on the camera trajectory and for that reason many (visual) SLAM algorithms have been evaluated on this dataset in the past. As with the KITTI dataset, the sequences do not come with a ground truth model of the scene and therefore we cannot perform a geometrical comparison. Instead we compare the different poses of the camera through time. To that end, we carefully selected five sequences that cover some of the main difficulties in existing SLAM systems, including a high (angular) velocity or the absence of salient features. The main specifications of the sequences are listed in Table 8.1.

In Figure 8.7, three images from three different sequences are depicted. The sequence *fr1_floor* is challenging because there is not a lot of structure in the depth maps as a a large part of the scan covers the ground plane. The other sequences are recorded around a desk and the depth maps contain more structure thanks to the clutter on top of the desk. For sequences *fr1_floor* and *fr1_desk*, the speed of the camera movement is fairly high, causing another challenge for the algorithms.

sequence name	length (m)	duration (s)	avg. ang. velocity (deg/s)	avg. transl. velocity (m/s)	frames
fr1_xyz	7.11	30.09	8.92	0.24	788
fr1_floor	12.57	49.87	15.07	0.26	1214
fr1_desk	9.26	23.40	23.33	0.41	575
fr2_xyz	7.029	121.48	1.716	0.058	3666
fr2_desk	18.880	69.15	6.338	0.193	2965

Table 8.1: Information about the five video sequences that were used in this work to evaluate our system and to compare it against existing SLAM methods. The sequences are part of the RGB-D SLAM benchmark presented in [Sturm 12].

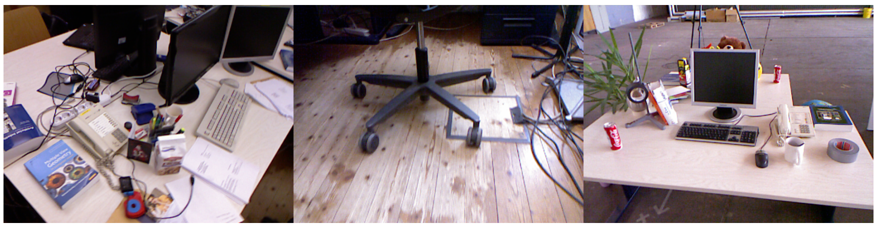


Figure 8.7: The sequences *fr1_desk*, *fr1_floor* and *fr2_desk* from the RGBD-SLAM benchmark presented in [Sturm 12]. The other two sequences that were used, *fr1_xyz* and *fr2_xyz* were recorded in the same environment as *fr1_desk* and *fr2_desk* but have different camera trajectories.

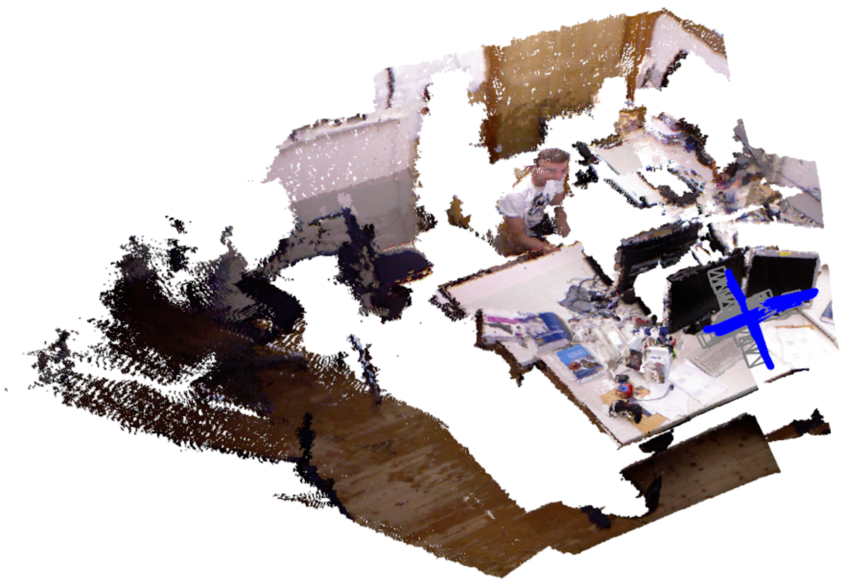


Figure 8.8: The 3D reconstruction and trajectory (in blue) obtained by our system for the sequence *fr1_desk* from the RGBD-SLAM benchmark presented in [Sturm 12].

In Figures 8.8, 8.9 and 8.10, the 3D-reconstructions obtained by our algorithm are depicted for the sequences *fr1_desk*, *fr1_floor* and *fr2_xyz*. Note that the 3D point clouds were generated in real-time on a laptop computer with 16GB RAM, an Intel Core i7-4712HQ, 2.30Ghz CPU and an nVidia Quadro K1100M inside. The small color discrepancies that are noticeable near the edges of objects are mainly due to small errors in the calibration of the color and depth camera as well as small synchronisation issues.

Relative pose error

We again use the relative pose error as one of the metrics to evaluate our camera tracking algorithm. This time we measure the local accuracy of the trajectory over a fixed distance Δ = 1.0 m. Between two poses $\mathbf{P}_i$ and $\mathbf{P}_j$, the average rotation and translation error, is again defined as in eqs. 2.13 and 2.14. The results of our method for this RPE metric are given in Table 8.2.

As can be noticed, the scores for the different sequences differ a lot from each other, which is mainly due to their difference in (angular) velocity. Our method is prone to quick camera motions because of the ‘closest point’ criterion of the ICP registration process. This imposes a maximum (rotational) speed given a certain

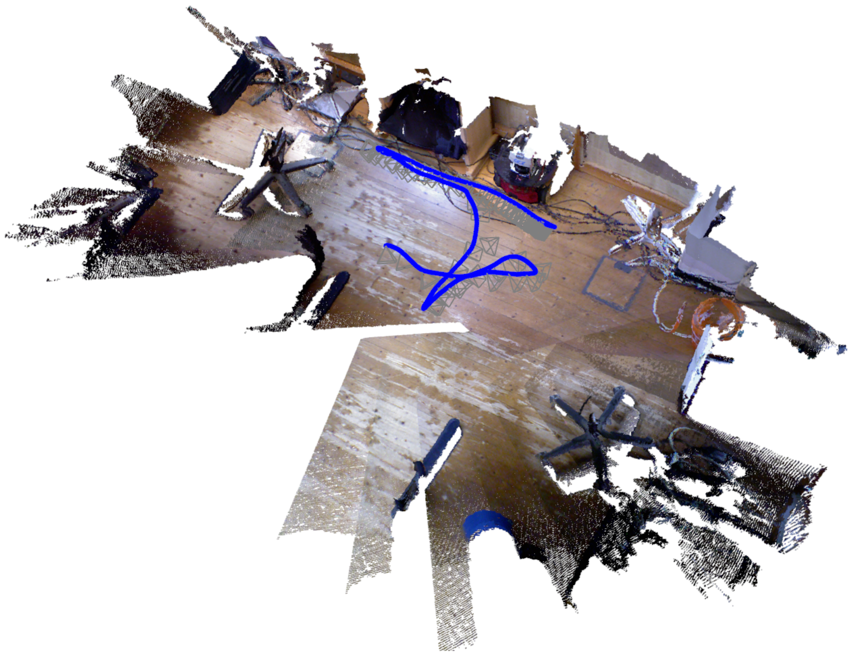


Figure 8.9: The 3D reconstruction and trajectory (in blue) obtained by our system for the sequence *fr1_floor* from the RGBD-SLAM benchmark presented in [Sturm 12].

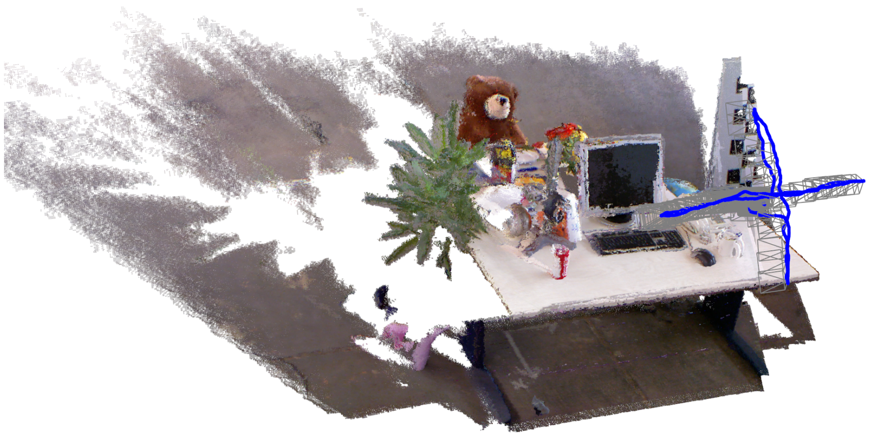


Figure 8.10: The 3D reconstruction and trajectory (in blue) obtained by our system for the sequence *fr2_xyz* from the RGBD-SLAM benchmark presented in [Sturm 12].

sequence name	$E_t(\Delta)$	$E_r(\Delta)$
fr1_xyz	0.036 m	$2.44°$
fr1_floor	0.092 m	$3.85°$
fr1_desk	0.143 m	$8.03°$
fr2_xyz	0.011 m	$0.65°$

Table 8.2: The relative pose error (RPE) for the sequences of the RGB-D Slam benchmark using $\Delta = 1.0$ meter. The RPE is deviating a lot for the different sequences, which is mainly due to the variations in angular velocities, cfr. Table 8.1. Sequence *fr2_xyz* is giving the best results as its average angular velocity is quite low.

frame rate in order for our solution still to work properly.

Sequences *fr1_floor* and *fr1_desk* have a much higher average (angular) velocity compared to the other sequences and for that reason the results are worse. The average angular and translational velocity of sequence *fr2_xyz*, respectively 1.7 deg/s and 0.058 m/s, is very low and the best results are, therefore, obtained for that sequence. Our algorithm leads to an average translational error of approximately 0.01 m and a rotational error of $0.65°$ degrees. Thus, as we picked $\Delta = 1.0$ m, we can conclude that under good conditions, i.e. a sufficiently low velocity of the camera, the translational error is near 1%.

In order to obtain a better understanding what causes the drift of our system, we examined the RPE at different time intervals. For brevity, we only include the graphs for sequences *fr1_desk* (Figure 8.11) and *fr2_xyz* (Figure 8.12) as these are the sequences for which our system generates respectively the worst and best results. Regarding sequence *fr1_desk*, we see that the large errors are due to a few outlier pose estimates, e.g. the large one around 16s caused by an abrupt movement of the camera. Figure 8.12 shows that for sequence *fr2_xyz* there is a still some variation in RPE for different time intervals, but there are no sever outlier pose estimates.

Absolute pose error

Alternatively, we consider the *absolute pose error* (APE), computed by comparing the absolute distances between the estimated and the ground truth trajectory, cfr. eq. 2.15. The APE scores for our solution on the five proposed sequences are listed in Table 8.3. As can be seen, the results measured by the APE metric are in line with the results measured by the RPE metric: sequences *fr1_floor* and *fr1_desk* have a higher error because of the higher velocity. Pure feature-based methods, such as ORB-SLAM, naturally have a higher resistance against fast and abrupt movements

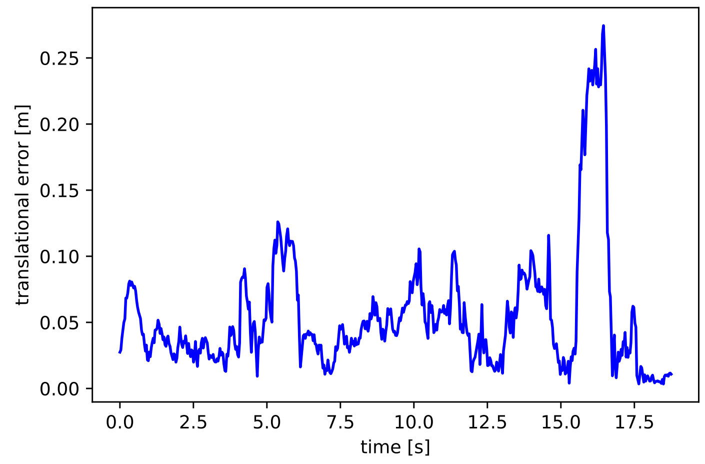


Figure 8.11: The RPE (RMSE) in function of time for sequence *fr1_desk* (with high angular velocity). The large APE of our solution is due to some outlier pose estimates, e.g. the large outlier around 16s. The cause of this is an abrupt movement of the camera at that time.

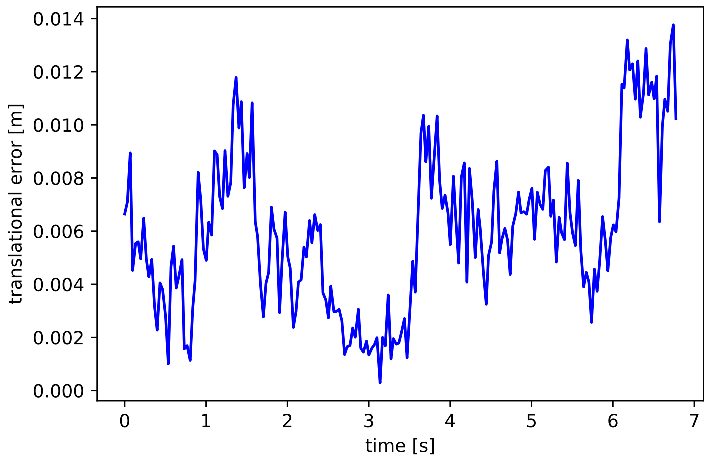


Figure 8.12: The RPE (RMSE) in function of time for sequence *fr2_xyz* (with low angular velocity). As in Figure 8.11, there is still a lot of variation along the sequence but there are no severe outliers of multiple centimeters.

Sequence	ORB-SLAM	PTAM	LSD-SLAM	RGBD-SLAM	D-VSLAM	Ours
fr1_xyz	0.90	1.15	9.00	1.34	1.1	2.45
fr2_xyz	0.30	0.20	2.15	2.61	-	0.69
fr1_floor	2.99	-	38.07	3.51	-	6.09
fr1_desk	1.69	-	10.65	2.58	2.1	8.12
fr2_desk	0.88	-	4.57	9.50	1.7	1.36

Table 8.3: The absolute trajectory error (APE) for the sequences of the RGB-D SLAM benchmark presented in [Sturm 12]. Our approach is compared with 6 other state-of-the-art systems. The best result is obtained for sequence *fr2_xyz*, while the sequences *fr1_floor* and *fr1_desk* have the worst outcome. The reason for this is that the first sequence has a very low angular velocity (1.7 deg/s) compared to the latter two sequences (15.1 and 23.3 deg/s respectively).

of the camera. On the other hand, LSD-SLAM, which is - like ours - a dense and direct approach, is suffering a lot from high (angular) velocities of the camera. Still, our solution outperforms LSD-SLAM in case of abrupt camera movements.

Similar as for the RPE, the best results are obtained for sequence *fr2_xyz*. With an APE of 0.69 it outperforms both the methods LSD-SLAM and RGBD-SLAM. The algorithm D-VSLAM suffered from tracking losses and no meaningful APE could be computed for sequence *fr2_xyz*. The systems ORB-SLAM and PTAM, generated better results. However, they rely on visual features in the scene and the RGBD sequences from [Sturm 12] are particularly well suited as they are highly textured. The PTAM method still suffered from tracking losses for the sequences *fr1_floor*, *fr1_desk* and *fr2_desk*.

Even though it seems that the ORB-SLAM system is the best performing one according to Table 8.3, it is important to note that this system integrates an extensive loop closure mechanism, whereas our results are obtained without any form of loop closure. The techniques for loop detection described in chapter 6, more specifically the M2DP descriptor, is not perfectly suited to cope with data acquired with a Kinect camera or another ToF camera. The main reason is that the FOV of these cameras is too small and for that reason, the M2DP descriptor will be different when the scene is captured from the same position but with a different orientation. As a result, the technique should be adapted in order to be able to deal with data from ‘projective’ ToF cameras.

For lengthy sequences (in terms of travelled distance), such as the *fr1_floor* sequence, our method will, therefore, suffer more from error propagation compared to ORB-SLAM. Another point of difference is that ORB-SLAM is a feature-based visual method that only results in a sparse 3D point cloud, whereas our method results in a dense 3D point cloud.

Because of the fact that there is no loop closure mechanism present in our camera tracking algorithm, the relative pose error is a better way to express its the accuracy. We conclude that for sequences having an angular velocity lower than approximately 10 deg/s, we are able to achieve a translational error between 1% and 4%. This means that when the camera moves 1 m, the error on the estimated position lies between 1 cm and 4 cm.

8.5 Conclusion

In this chapter, we presented a novel algorithm for finding the 3D pose of a depth camera relative to a CAD model as well as an algorithm to track the pose of the depth camera through time. The former algorithm can be used to estimate the initial pose of the user once he puts on a head-mounted device (with integrated depth camera), as well as to relocate the user in case of severe drift or tracking loss. The second algorithm can be used to continuously provide the direction in which the user is looking in order to augment his view with virtual objects.

The first algorithm is based on the idea of 4-points congruent sets [Aiger 08] and improves over the standard solution by replacing its randomized *generate-and-test* paradigm. The latter is achieved by exploiting the geometry of the available CAD model to select co-planar bases instead of randomly looking for them.

The second algorithm is based on the registration of point clouds derived from the depth camera. It integrates low-level shape features to provide weights to the ICP-based alignment. Our algorithm also includes scan-to-model matching using surface reconstruction and re-sampling. The latter allows to extract multi-resolution representations from the ‘model’, being either the ‘current’ aggregated point cloud or a point cloud derived from an existing CAD model. This multi-resolution representation facilitates our fast coarse-to-fine ICP registration algorithm. Together with the GPU-based point-to-plane implementation, it allows to track the (depth) camera in real-time.

Regarding the accuracy we conclude that in absence of abrupt camera movements and given a low angular velocity (lower than 10 deg/s), the translational error of our system is situated between 1% and 4%. Regarding the robustness, our system outperforms current state-of-the-art trackers, as many of them were leading to tracking losses in cases where our system continued to work properly. Although our camera tracker also has an upper limit in terms of (angular) velocity, it will hardly lead to a tracking loss, but instead its accuracy will be gently reduced.

9 Conclusions

As mentioned in the introduction, our main contributions to the 3D mapping problem are threefold: increasing the robustness and accuracy, improving the scalability and accelerating the process. The combination of all our improvements allows applications such as inspection, damage assessment or construction site monitoring to be fully automated and accelerated using robots or drones. All these applications share the need for highly accurate 3D models that have to be built in a very quick manner. Below we summarize our contributions in more detail.

9.1 Summary of contributions

9.1.1 Increasing the robustness and accuracy

In order to increase the robustness and accuracy of 3D SLAM, we contributed in four different ways. First, we devised a new cost function based on the underlying surface model, which we denoted as *surface-based* GICP (S-GICP). This cost function performs well in typical man-made environments with strong Manhattan-world properties that yield a high degree of orthogonality.

Second, as outdoor environments are less structured, we additionally invented techniques to sample the point clouds in a geometrical stable way. More specifically, we ensured that the observability in all directions is proportioned in order to get rid of directional biases. Thereby we mainly use points that are sampled in stable regions that are characterized by a high degree of planarity. For those regions, surface normals can be more accurately estimated, leading to better transformation estimates while used in the point-to-plane or plane-to-plane cost function.

Third, we proposed a more robust algorithm to estimate the surface normals, taking into account the topology of the point cloud. To that end, we devised a method based on point clustering and entropy-based radius selection to determine the optimal neighbourhood for each point. The entropy-based radius selection is further exploited to compute a low-level feature representation of the point cloud. This representation is eventually used to deal with outlier correspondences. More specifically, we integrate it in a feature-based weighting step amending both the robustness and final accuracy of our SLAM system.

Fourth, we combined scan-to-scan matching with scan-to-model alignment. To that end, we developed a surface reconstruction algorithm, which allows us to re-sample the points in the aggregated point cloud or '3D map'. The surface reconstruction is in fact an approximation defined by a local neighbourhood of points, which is more robust to noise compared to interpolation methods or triangular meshing techniques. More specifically, for each point we compute a bivariate polynomial height function defined on a robustly computed reference plane derived from the surface normal. Our surface approximation reduces the noise and previously made errors in the aggregated point cloud, which is highly beneficial for future pose estimates in terms of robustness and accuracy.

9.1.2 Improving the scalability

Regarding the scalability, we provided mechanisms to gently trade off accuracy for speed. To that end, we developed an octree-based hierarchical data structure facilitating dynamic adjustments of the level of detail. In fact, we combined the octree data structure with our surface approximation, storing the parameters of the polynomial function in the octree cells. The surface is thus stored in its parametric form in contrast to implicit forms usually found in literature, which are wasteful in terms of memory. Our surface representation on the other hand allows to strongly compress the map. Furthermore, it is way more easy to update and modify it compared to interpolation methods or triangular meshes, which makes incremental scan-to-model alignment a lot more efficient. All this means that our solution is ready to be used in a distributed mapping approach using multiple agents.

9.1.3 Accelerating registration and loop closure

The acceleration of the 3D mapping process is achieved by three major developments. First, the re-sampling of the point cloud map reduces the redundancy and makes it possible to represent it using fewer points, which speeds up the scan-to-map registration.

Second, the octree data structure makes the data association more efficient as spatial queries, e.g. nearest neighbour searches, can be conducted in logarithmic time. The octree can also quickly be updated and hence we can avoid to rebuild it from scratch every time a new scan has to be aligned. Finally, based on the octree data structure, we developed a coarse-to-fine ICP-based registration algorithm that converges faster than the 'full-resolution' variants while keeping the same performance.

Third, we developed a GPU-based global point cloud descriptor that is highly effective to detect loop candidates. The loop candidates are further refined by checking the geometrical consistency between the two point clouds after registration. This registration differs from scan matching as the two point clouds may be largely rotated if the same location was reached from two different directions. We developed a 'global' registration technique based on the matching of 4-points congruent sets to align those point clouds. The main novelty is the base selection part, which makes it possible to omit the randomized base selection, yielding another acceleration.

9.2 Future research

In this future work section, we will distinguish two main parts. The first one deals with research related to the open challenges of SLAM itself. In the second part, we will look beyond the problem of SLAM and enumerate which research is needed to enable new applications.

9.2.1 SLAM

Concerning SLAM, a few future research directions were already given in chapter 2. The most important ones that we envision are related to the *robustness* and *scalability* of SLAM solutions.

Regarding the robustness we explained that there are many sources of failures, located at both the algorithmic and hardware level. However, more important than how to prevent these failures is the way future SLAM systems will cope with them in terms of re-localization and recovery. Today failures are almost always leading to an entire shut-down and restart of the system which also requires manual intervention. Future SLAM systems will be fully autonomous with regard to recovery and re-localization. Related to this is the *'curse of manual parameter tuning'* from which all current SLAM systems suffer. An important role will be played by advanced machine learning techniques to alleviate this parameter tuning.

Regarding scalability, future SLAM systems will no longer operate alone, but instead they will cooperate in distributed mapping. A lot of research is still necessary to investigate how future systems will share information: whether it will be in a centralized or decentralized way. Scalability also concerns the efficient processing of the tremendous amount of sensor data, the representation of the map and the aggregation of new data in this map. Our work has taken some major steps towards efficiency, but more research is needed to learn which information is redundant or should be forgotten.

Apart from the map, the representation of the SLAM pose graph is another future research topic. It is clear that when we visit the same place over and over again, new nodes will contribute less and less in improving the overall map. Future SLAM systems will thus have to integrate *pruning* mechanisms to lower the number of poses in the SLAM graph. In order to come to a truly scalable system, the use of *continuous-time* trajectory estimation will be necessary. Instead of working with discrete poses, temporal basis functions, such as B-splines, will be used to keep the state size manageable.

Also, the number and variety of devices that will integrate SLAM will increase tremendously in the future and will no longer be limited to well-equipped laptops in research labs. Instead they will be incorporated in domestic robots, inspection robots, surveillance robots or cleaning machines in public places and so on. They will be integrated in drones in order for them to operate in indoor environments or hard-to-reach areas where there is no GPS signal. All these platforms have in common that they are small and resource-constrained, both in terms of computation power and battery live. More research is therefore needed to make SLAM systems even more efficient.

Finally, a major future research direction is the domain of semantic SLAM which will provide rich object-level maps [Feng 19]. An important role in this will be played by machine learning and more specifically by deep learning. The maps of the future will no longer be purely metric, but will contain semantic information of the scene which will be derived from classification using deep learning. The supremacy of CNN-based descriptors for classification is by now a commonly-known fact, but they will also be applied to perform other subtasks of SLAM such as feature matching, automatic parameter tuning or place recognition.

9.2.2 Beyond SLAM

Besides the open challenges in SLAM, there is still research needed in view of fully automated 3D mapping or modelling. One future research direction is the simplification of 3D models generated by SLAM. The main output of our system is a large aggregated point cloud. Even though there is an octree datastructure and surface reconstruction behind it, it was mainly developed in order to increase the precision of the 3D map and to reduce the noise and the overall drift. The reduction of noise facilitates the creation of a triangular mesh, but the triangular mesh is often still too complex to be used for other purposes such as VR-rendering or 3D-printing. More research on automated mesh decimation techniques or re-topology methods is, therefore, needed. In order to obtain a 3D model suited for example in CAD design or to be 3D-printed we still need an efficient conversion tool. In several applications such as construction site monitoring, one is often also interested in creating a building information model (BIM) from the scan data, a problem usually referred to as *scan2BIM*. To come to fully automated scan2BIM, current techniques to derive semantic information from the scene still need to be improved.

References

[Agamennoni 16] Gabriel Agamennoni, Simone Fontana, Roland Siegwart, and Domenico Sorrenti. “Point Clouds Registration with Probabilistic Data Association”. In: Oct. 2016, pp. 4092–4098.

[Aiger 08] Dror Aiger, Niloy J. Mitra, and Daniel Cohen-Or. “4pointss Congruent Sets for Robust Pairwise Surface Registration”. In: *ACM SIGGRAPH 2008 Papers*. SIGGRAPH ’08. Los Angeles, California: ACM, 2008, 85:1–85:10.

[Alismail 14] Hatem Alismail, L. Douglas Baker, and Brett Browning. “Continuous trajectory estimation for 3D SLAM from actuated lidar”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), pp. 6096–6101.

[Arun 87] K. S. Arun, Thomas S. Huang, and Steven D. Blostein. “Least-Squares Fitting of Two 3-D Point Sets.” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 9.5 (1987), pp. 698–700.

[Bay 08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. “Speeded-Up Robust Features (SURF)”. In: *Comput. Vis. Image Underst.* 110.3 (June 2008), pp. 346–359. ISSN: 1077-3142.

[Beno 16] Peter Beno, Vladimir Pavelka, Frantiek Duchon, and Martin Dekan. “Using Octree Maps and RGBD Cameras to Perform Mapping and A* Navigation”. In: *International Conference on Intelligent Networking and Collaborative Systems (INCoS)*. IEEE, Sept. 2016. ISBN: 978-1-5090-4124-4.

[Besl 92] Paul J. Besl and Neil D. McKay. “A Method for Registration of 3-D Shapes”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 14.2 (Feb. 1992), pp. 239–256.

[Biber 03] Peter Biber and Wolfgang Strasser. “The normal distributions transform: a new approach to laser scan matching.” In: *IROS*. IEEE, 2003, pp. 2743–2748.

[Bleser 05] Gabriele Bleser, Yulian Pastarmov, and Didier Stricker. “Real-time 3D Camera Tracking for Industrial Augmented Reality Applications”. In: *WSCG*. 2005.

[Borrmann 08] Dorit Borrmann, Jan Elseberg, Kai Lingemann, Andreas Nüchter, and Joachim Hertzberg. “Globally consistent 3D mapping with scan matching”. In: *Robotics and Autonomous Systems* 56.2 (2008), pp. 130–142.

[Bosse 10] Michael Bosse and Robert Zlot. “Continuous 3D scan-matching with a spinning 2D laser.” In: *ICRA*. IEEE, Jan. 28, 2010, pp. 4312–4319.

[Bosse 12] Michael Bosse, Robert Zlot, and Paul Flick. “Zebedee: Design of a Spring-Mounted 3-D Range Sensor with Application to Mobile Mapping”. In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1104–1119.

[Bosse 13] Michael Bosse and Robert Zlot. “Place recognition using keypoint voting in large 3D lidar datasets.” In: *ICRA*. IEEE, 2013, pp. 2677–2684.

[Brand 14] Christoph Brand, Martin J. Schuster, Heiko Hirschmüller, and Michael Suppa. “Stereo-vision based obstacle mapping for indoor/outdoor SLAM”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014), pp. 1846–1853.

[Calonder 10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. “BRIEF: Binary Robust Independent Elementary Features”. In: *Proceedings of the 11th European Conference on Computer Vision: Part IV*. ECCV’10. Heraklion, Crete, Greece: Springer-Verlag, 2010, pp. 778–792. ISBN: 3-642-15560-X, 978-3-642-15560-4.

[Carr 01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. “Reconstruction and Representation of 3D Objects with Radial Basis Functions”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 67–76. ISBN: 1-58113-374-X.

[Chen 16] Heen Chen, Hanwu He, Jianqing Mo, Jinfang Li, and Xian Yang. “3D registration based perception in augmented reality environment”. In: *Cogent Engineering* 3.1 (2016).

[Chen 92] Yang Chen and Gérard Medioni. “Object Modeling by Registration of Multiple Range Images”. In: 10 (Jan. 1992), pp. 145–155.

[Civera 11]	Javier Civera, Dorian Gálvez-López, Luis Riazuelo, Juan D. Tardós, and José Maria Montiel. “Towards semantic SLAM using a monocular camera”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2011), pp. 1277–1284.
[Corsini 13]	M. Corsini, M. Dellepiane, F. Ganovelli, R. Gherardi, A. Fusiello, and R. Scopigno. “Fully Automatic Registration of Image Sets on Approximate Geometry”. In: *International Journal of Computer Vision* 102.1 (Mar. 2013), pp. 91–111. ISSN: 1573-1405.
[Costante 16]	Gabriele Costante, Michele Mancini, Paolo Valigi, and Thomas A. Ciarfuglia. “Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation”. In: *IEEE Robotics and Automation Letters* 1.1 (2016), pp. 18–25.
[Curless 96]	Brian Curless and Marc Levoy. “A Volumetric Method for Building Complex Models from Range Images”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New York, NY, USA: ACM, 1996, pp. 303–312. ISBN: 0-89791-746-4.
[Dame 13]	Amaury Dame, Victor A. Prisacariu, Carl Y. Ren, and Ian Reid. “Dense Reconstruction Using 3D Object Shape Priors”. In: *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1288–1295. ISBN: 978-0-7695-4989-7.
[Dewan 18]	Ayush Dewan, Tim Caselitz, and Wolfram Burgard. “Learning a Local Feature Descriptor for 3D LiDAR Scans”. In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain, 2018.
[Droeschel 18]	David Droeschel and Sven Behnke. “Efficient Continuous-Time SLAM for 3D Lidar-Based Online Mapping”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 1–9.
[Dubé 16]	Renaud Dubé, Daniel Dugas, Elena Stumm, Juan I. Nieto, Roland Siegwart, and Cesar Cadena. “SegMatch: Segment based loop-closure for 3D point clouds”. In: *CoRR* abs/1609.07720 (2016).
[Einhorn 15]	Erik Einhorn and Horst-Michael Gross. “Generic NDT Mapping in Dynamic Environments and Its Application for Lifelong SLAM”. In: *Robot. Auton. Syst.* 69.C (July 2015), pp. 28–39.

[Endres 12] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. “An evaluation of the RGB-D SLAM system.” In: *ICRA*. IEEE, 2012, pp. 1691–1696.

[Engel 14] J. Engel, T. Schöps, and D. Cremers. “LSD-SLAM: Large-Scale Direct Monocular SLAM”. In: Sept. 2014.

[Feng 19] Q. Feng, Y. Meng, M. Shan, and N. Atanasov. “Localization and Mapping using Instance-specific Mesh Models”. In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2019.

[Fitzgibbon 01] A. W. Fitzgibbon. “Robust Registration of 2D and 3D Point Sets”. In: *British Machine Vision Conference*. 2001, pp. 662–670.

[Furgale 12] Paul Timothy Furgale, Timothy D. Barfoot, and Gabe Sibley. “Continuous-time batch estimation using temporal basis functions.” In: *ICRA*. IEEE, 2012, pp. 2088–2095. ISBN: 978-1-4673-1403-9.

[Galliani 15] Silvano Galliani, Katrin Lasinger, and Konrad Schindler. “Massively Parallel Multiview Stereopsis by Surface Normal Diffusion”. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV ’15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 873–881. ISBN: 978-1-4673-8391-2.

[Geiger 12] Andreas Geiger. “Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 3354–3361.

[Goossens 14] Bart Goossens, Jonas De Vylder, and Wilfried Philips. “Quasar: a new heterogeneous programming framework for image and video processing algorithms on CPU and GPU”. In: *IEEE International Conference on Image Processing ICIP*. Paris, France: IEEE, 2014, pp. 2183–2185.

[Grant 13] W.S. Grant, R.C. Voorhies, and Laurent Itti. “Finding planes in LiDAR point clouds for real-time registration”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2013.

[Grisetti 09] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Nonlinear constraint network optimization for efficient map learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 10.3 (2009), pp. 428–439.

[Han 15] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C. Berg. "MatchNet: Unifying feature and metric learning for patch-based matching." In: *CVPR*. IEEE Computer Society, 2015, pp. 3279–3286. ISBN: 978-1-4673-6964-0.

[Hartley 09] Richard I. Hartley and Fredrik Kahl. "Global Optimization through Rotation Space Search". In: *International Journal of Computer Vision* 82.1 (2009), pp. 64–79.

[He 16] Li He, Xiaolong Wang, and Hong Zhang. "M2DP: A novel 3D point cloud descriptor and its application in loop closure detection." In: *IROS*. IEEE, 2016, pp. 231–237.

[He 18] Kun He, Yan Lu, and Stan Sclaroff. "Local Descriptors Optimized for Average Precision". In: *CoRR* abs/1804.05312 (2018).

[Henry 12] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 647–663.

[Holz 14] D. Holz and S. Behnke. "Registration of Non-Uniform Density 3D Point Clouds using Approximate Surface Reconstruction". In: *ISR/Robotik 2014; 41st International Symposium on Robotics*. June 2014, pp. 1–7.

[Holz 15] Dirk Holz and Sven Behnke. "Registration of Non-uniform Density 3D Laser Scans for Mapping with Micro Aerial Vehicles". In: *Robot. Auton. Syst.* 74.PB (Dec. 2015), pp. 318–330. ISSN: 0921-8890.

[Hong 10] Sengpyo Hong, Heedong Ko, and Jinwook Kim. "VICP: Velocity updating iterative closest point algorithm." In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 1893–1898.

[Horn 87] Berthold K. P. Horn. "Closed-form solution of absolute orientation using unit quaternions". In: *Journal of the Optical Society of America A* 4.4 (1987), pp. 629–642.

[Horn 88] Berthold K. P. Horn, Hugh M. Hilden, and Shahriar Negahdaripour. "Closed-form solution of absolute orientation using orthonormal matrices". In: *J. Opt. Soc. Am. A* 5.7 (July 1988), pp. 1127–1135.

[Hornung 13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees". In: *Autonomous Robots* (2013).

[Huang 11] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Dieter Fox, and Nicholas Roy. "Visual odometry and mapping for autonomous flight using an RGB-D camera". In: *Proc. of the Intl. Sym. of Robot. Research*. 2011.

[Ila 10] Viorela Ila, Josep M. Porta, and Juan Andrade-Cetto. "Information-based Compact Pose SLAM". In: *Trans. Rob.* 26.1 (Feb. 2010), pp. 78–93. ISSN: 1552-3098.

[Izadi 11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera". In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. Santa Barbara, California, USA: ACM, 2011, pp. 559–568. ISBN: 978-1-4503-0716-1.

[Johnson 97] Andrew Johnson. "Spin-Images: A Representation for 3-D Surface Matching". PhD thesis. Pittsburgh, PA: Carnegie Mellon University, Aug. 1997.

[Kaess 15] M. Kaess. "Simultaneous Localization and Mapping with Infinite Planes". In: *Proc. IEEE Intl. Conf. on Robotics and Automation, ICRA*. Seattle, WA, May 2015, pp. 4605–4611.

[Kang 19] Rong Kang, Jieqi Shi, Xueming Li, Yang Liu, and Xiao Liu. "DF-SLAM: A Deep-Learning Enhanced Visual SLAM System based on Deep Local Features". In: *CoRR* abs/1901.07223 (2019).

[Kendall 15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. "Convolutional networks for real-time 6-DOF camera relocalization". In: *CoRR* abs/1505.07427 (2015).

[Kerl 13a] Christian Kerl, Jürgen Sturm, and Daniel Cremers. "Dense visual SLAM for RGB-D cameras." In: *IROS*. IEEE, 2013, pp. 2100–2106.

[Kerl 13b] Christian Kerl, Jürgen Sturm, and Daniel Cremers. "Robust odometry estimation for RGB-D cameras." In: *ICRA*. IEEE, 2013, pp. 3748–3754. ISBN: 978-1-4673-5641-1.

[Kim 13] Changmin Kim, Hyojoo Son, and Changwan Kim. "Fully automated registration of 3D data to a 3D CAD model for project progress monitoring". English. In: *Automation in Construction* 35.Complete (2013), pp. 587–594.

[Kim 16] Jae-Hak Kim, Cesar Cadena, and Ian D. Reid. "Direct semi-dense SLAM for rolling shutter cameras". In: *ICRA*. IEEE, 2016, pp. 1308–1315.

[Klein 06] Georg Klein and David W. Murray. "Full-3D Edge Tracking with a Particle Filter". In: *BMVC*. 2006.

[Klein 07] Georg Klein and David Murray. "Parallel Tracking and Mapping for Small AR Workspaces". In: *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*. Nara, Japan, Nov. 2007.

[Kolluri 08] Ravikrishna Kolluri. "Provably Good Moving Least Squares". In: *ACM Trans. Algorithms* 4.2 (May 2008), 18:1–18:25. ISSN: 1549-6325.

[Korkalo 16] Otto Korkalo and Svenja Kahn. "Real-time depth camera tracking with CAD models and ICP". English. In: *Journal of Virtual Reality and Broadcasting* 13.1 (2016). SDA: SHP: Pro-Io-T. ISSN: 1860-2037.

[Krishnan 15] Rahul G. Krishnan, Uri Shalit, and David A Sontag. "Deep Kalman Filters". In: *ArXiv* abs/1511.05121 (2015).

[Kümmerle 11] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. "G^{2}o: A general framework for graph optimization". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 3607–3613.

[Le 17] Phi-Hung Le and Jana Kosecka. "Dense piecewise planar RGB-D SLAM for indoor environments". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Sept. 2017).

[Lepetit 05] Vincent Lepetit and Pascal Fua. "Monocular Model-based 3D Tracking of Rigid Objects". In: *Found. Trends. Comput. Graph. Vis.* 1.1 (Jan. 2005), pp. 1–89. ISSN: 1572-2740.

[Li 01] ZL Li, Z Xu, MY Cen, and XL Ding. "Robust surface matching for automated detection of local deformations using least-median-of-squares estimator". In: *Photogrammetric Engineering and Remote Sensing* 67.11 (11 2001), 1283–1292.

[Lorensen 87] William E. Lorensen and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 163–169. ISBN: 0-89791-227-6.

[Low 04] Kok-Lim Low. *Linear least-squares optimization for point-to plane ICP surface registration*. Tech. rep. 2004.

[Lowe 04] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 0920-5691.

[Lu 97] F. Lu and E. Milios. “Globally Consistent Range Scan Alignment for Environment Mapping”. In: *Auton. Robots* 4.4 (Oct. 1997), pp. 333–349.

[Luong 16] Hiep Luong, Michiel Vlaminck, Werner Goeman, and Wilfried Philips. “Consistent ICP for the registration of sparse and inhomogeneous point clouds”. In: *International Conference on Communications and Electronics (ICCE)*. Ha Long, Vietnam: IEEE, 2016, pp. 262–267.

[Magnusson 07] Martin Magnusson, Achim J. Lilienthal, and Tom Duckett. “Scan registration for autonomous mining vehicles using 3D-NDT.” In: *J. Field Robotics* 24.10 (2007), pp. 803–827.

[Magnusson 09a] Martin Magnusson, Henrik Andreasson, Andreas Nüchter, and Achim J. Lilienthal. “Appearance-based loop detection from 3D laser data using the normal distributions transform”. In: *2009 IEEE International Conference on Robotics and Automation* (2009), pp. 23–28.

[Magnusson 09b] Martin Magnusson, Andreas Nüchter, Christopher Lörken, Achim J. Lilienthal, and Joachim Hertzberg. “Evaluation of 3D registration reliability and speed - A comparison of ICP and NDT.” In: *ICRA*. IEEE, 2009, pp. 3907–3912.

[Mellado 14] Nicolas Mellado, Dror Aiger, and Niloy J. Mitra. “Super 4PCS: Fast Global Pointcloud Registration via Smart Indexing”. In: *Computer Graphics Forum* (2014).

[Moosmann 11] Frank Moosmann and Christoph Stiller. “Velodyne SLAM”. In: *Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 393–398.

[Moral 13] Eduardo Fernández Moral, Walterio W. Mayol-Cuevas, Vicente Arévalo, and Javier González Jiménez. “Fast place recognition with plane-based maps.” In: *ICRA*. IEEE, 2013, pp. 2719–2724.

[Morton 66] G.M. Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, 1966.

[Mueggler 15] Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza. “Continuous-Time Trajectory Estimation for Event-based Vision Sensors”. In: *Robotics: Science and Systems*. 2015.

[MurArtal 15] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”. In: *IEEE Trans. Robotics* 31.5 (2015), pp. 1147–1163.

[Newcombe 11] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. "Kinect-Fusion: Real-Time Dense Surface Mapping and Tracking". In: *IEEE ISMAR*. IEEE, Oct. 2011.

[Nüchter 04] Andreas Nüchter, Hartmut Surmann, Kai Lingemann, Joachim Hertzberg, and Sebastian Thrun. "6D SLAM with an Application in Autonomous Mine Mapping". In: *ICRA*. IEEE, 2004, pp. 1998–2003.

[Nüchter 05] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. "6D SLAM with Approximate Data Association". In: *In Proc. of the 12th Int. Conference on Advanced Robotics (ICAR*. 2005, pp. 242–249.

[Nüchter 07] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. "6D SLAM - 3D mapping outdoor environments". In: *Journal of Field Robotics* 24.8-9 (2007), pp. 699–722.

[Pathak 10] Kaustubh Pathak, Andreas Birk, Narunas Vaskevicius, and Jann Poppinga. "Fast Registration Based on Noisy Planes with Unknown Correspondences for 3D Mapping". In: *IEEE Transactions on Robotics* 26.3 (2010), pp. 424–441.

[PatronPerez 15] Alonso Patron-Perez, Steven Lovegrove, and Gabe Sibley. "A Spline-Based Trajectory Representation for Sensor Fusion and Rolling Shutter Cameras". In: *International Journal of Computer Vision* 113 (2015), pp. 208–219.

[Pomerleau 13] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. "Comparing ICP variants on real-world data sets: Open-source library and experimental protocol". In: *Autonomous Robots* 34.3 (2013), pp. 133–148.

[Rublee 11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An Efficient Alternative to SIFT or SURF". In: *Proceedings of the 2011 International Conference on Computer Vision*. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2564–2571. ISBN: 978-1-4577-1101-5.

[Rusinkiewicz 01] Szymon Rusinkiewicz and Marc Levoy. "Efficient Variants of the ICP Algorithm". In: *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*. June 2001.

[Rusu 09] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast Point Feature Histograms (FPFH) for 3D Registration". In: *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*. ICRA'09. Kobe, Japan: IEEE Press, 2009, pp. 1848–1853.

[Rusu 10] Radu Bogdan Rusu, Gary R. Bradski, Romain Thibaux, and John M. Hsu. “Fast 3D recognition and pose using the Viewpoint Feature Histogram.” In: *IROS*. IEEE, 2010, pp. 2155–2162.

[Saeedi 16] Sajad Saeedi, Michael Trentini, Mae Seto, and Howard Li. “Multiple-Robot Simultaneous Localization and Mapping: A Review”. In: *J. Field Robot.* 33.1 (Jan. 2016), pp. 3–46. ISSN: 1556-4959.

[SalasMoreno 13] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. “Slam++: Simultaneous localisation and mapping at the level of objects”. In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 1352–1359.

[SalasMoreno 14] Renato F. Salas-Moreno, Ben Glocker, Paul H. J. Kelly, and Andrew J. Davison. “Dense planar SLAM”. In: *ISMAR*. IEEE Computer Society, 2014, pp. 157–164.

[Salti 14] Samuele Salti, Federico Tombari, and Luigi di Stefano. “SHOT: Unique signatures of histograms for surface and texture description”. In: *Computer Vision and Image Understanding* 125 (2014), pp. 251–264.

[Sarvrood 16] Yashar Balazadegan Sarvrood, Siavash Hosseinyalamdary, and Yang Gao. “Visual-LiDAR Odometry Aided by Reduced IMU.” In: *ISPRS Int. J. Geo-Information* 5.1 (2016).

[Schönberger 16] Johannes Lutz Schönberger and Jan-Michael Frahm. “Structure-from-Motion Revisited”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Scovanner 07] Paul Scovanner, Saad Ali, and Mubarak Shah. “A 3-dimensional Sift Descriptor and Its Application to Action Recognition”. In: *Proceedings of the 15th International Conference on Multimedia*. MULTIMEDIA ’07. Augsburg, Germany: ACM, 2007, pp. 357–360.

[Segal 09] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. “Generalized-ICP.” In: *Robotics: Science and Systems*. Ed. by Jeff Trinkle, Yoky Matsuoka, and José A. Castellanos. The MIT Press, 2009.

[Serafin 15a] Jacopo Serafin and Giorgio Grisetti. “NICP: Dense normal based point cloud registration”. In: *IROS*. IEEE, 2015, pp. 742–749.

[Serafin 15b] Jacopo Serafin and Giorgio Grisetti. “NICP: Dense normal based point cloud registration”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), pp. 742–749.

[Servos 17] James Servos and Steven L. Waslander. “Multi-Channel Generalized-ICP: A robust framework for multi-channel scan registration”. In: *Robotics and Autonomous Systems* 87 (2017), pp. 247–257.

[Shoemake 85] Ken Shoemake. “Animating Rotation with Quaternion Curves”. In: *SIGGRAPH Comput. Graph.* 19.3 (July 1985), pp. 245–254.

[Sipiran 10a] Ivan Sipiran and Benjamin Bustos. “A Robust 3D Interest Points Detector Based on Harris Operator”. In: *Eurographics Workshop on 3D Object Retrieval*. Ed. by Mohamed Daoudi and Tobias Schreck. The Eurographics Association, 2010.

[Sipiran 10b] Ivan Sipiran and Benjamin Bustos. “A Robust 3D Interest Points Detector Based on Harris Operator”. In: *Eurographics Workshop on 3D Object Retrieval*. Ed. by Mohamed Daoudi and Tobias Schreck. The Eurographics Association, 2010. ISBN: 978-3-905674-22-4.

[Siritanawan 17] Prarinya Siritanawan, Moratuwage Diluka Prasanjith, and Danwei Wang. “3D feature points detection on sparse and non-uniform pointcloud for SLAM”. In: *ICAR*. IEEE, 2017, pp. 112–117.

[Steder 10] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. “NARF: 3D Range Image Features for Object Recognition”. In: *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan, Oct. 2010.

[Steder 11] Bastian Steder, Michael Ruhnke, Slawomir Grzonka, and Wolfram Burgard. “Place recognition in 3D scans using a combination of bag of words and point feature based relative pose estimation.” In: *IROS*. IEEE, 2011, pp. 1249–1255.

[Sturm 12] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A benchmark for the evaluation of RGBD SLAM systems”. In: *International Conference on Intelligent Robot Systems (IROS)*. 2012.

[Sun 13] You Xia Sun and Jin Liang Li. “Mapping of Rescue Environment Based on NDT Scan Matching”. In: *Optoelectronics Engineering and Information Technologies in Industry*. Vol. 760. Advanced Materials Research. Trans Tech Publications, Oct. 2013, pp. 928–933.

[Sun 18] Tiezhu Sun, Wei Zhang, Zhijie Wang, Lin Ma, and Zequn Jie. “Image-level to Pixel-wise Labeling: From Theory to Practice”. In: July 2018, pp. 928–934.

[Tateno 17] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. “CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction”. In: *CoRR* abs/1704.03489 (2017).

[Ulas 13] Cihan Ulas and Hakan Temeltas. “3D Multi-Layered Normal Distribution Transform for Fast and Long Range Scan Matching”. In: *Journal of Intelligent and Robotic Systems* 71 (July 2013).

[Umeyama 91] Shinji Umeyama. “Least-Squares Estimation of Transformation Parameters Between Two Point Patterns”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 13.4 (Apr. 1991), pp. 376–380. ISSN: 0162-8828.

[Valentin 15] Julien P. C. Valentin, Matthias Nießner, Jamie Shotton, Andrew W. Fitzgibbon, Shahram Izadi, and Philip H. S. Torr. “Exploiting uncertainty in regression forests for accurate camera relocalization”. In: *CVPR*. IEEE Computer Society, 2015, pp. 4400–4408.

[Vlaminck 13] Michiel Vlaminck, Ljubomir Jovanov, Peter Van Hese, Bart Goossens, Wilfried Philips, and Aleksandra Pizurica. “Obstacle detection for pedestrians with a visual impairment based on 3D imaging”. In: *International Conference on 3D Imaging (IC3D)*. Liege, Belgium: IEEE, 2013, pp. 6–12.

[Vlaminck 16a] Michiel Vlaminck, Hiep Luong, Werner Goeman, and Wilfried Philips. “3D Scene Reconstruction Using Omnidirectional Vision and LiDAR: A Hybrid Approach”. In: *Sensors* 16.11 (2016).

[Vlaminck 16b] Michiel Vlaminck, Hiep Luong, Werner Goeman, Peter Veelaert, and Wilfried Philips. “Towards online mobile mapping using inhomogeneous lidar data”. In: *Intelligent Vehicles Symposium (IV)*. Gothenburg, Sweden: IEEE, 2016, pp. 845–850.

[Vlaminck 16c] Michiel Vlaminck, Hiep Luong, Hoang Van Nam, Hai Vu, Peter Veelaert, and Wilfried Philips. “Indoor assistance for visually impaired people using a RGB-D camera”. In: *Southwest Symposium on Image Analysis and Interpretation (SSIAI)*. Santa Fe, NM, USA: IEEE, 2016, pp. 161–164.

[Vlaminck 17a] Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “A markerless 3D tracking approach for augmented reality applications”. In: *2017 International Conference on 3D Immersion (IC3D)*. Brussels, Belgium, Dec. 2017, pp. 1–7.

[Vlaminck 17b] Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Liborg: a lidar-based robot for efficient 3D mapping”. In: *SPIE Applications On Digital Imaging*. Vol. 10396. 1. San Diego, USA: SPIE, 2017.

[Vlaminck 17c] Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Multi-resolution ICP for the efficient registration of point clouds based on octrees”. In: *International Conference On Machine Vision Applications (MVA)*. Nagoya, Japan: IAPR, 2017, pp. 334–337.

[Vlaminck 18a] Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Have I Seen This Place Before? A Fast and Robust Loop Detection and Correction Method for 3D Lidar SLAM”. In: *Sensors* 19.1 (2018).

[Vlaminck 18b] Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “LiBorg 2.0: a robot for on-the-fly 3D mapping of environments”. In: *Imec magazine* (Oct. 2018).

[Vlaminck 18c] Michiel Vlaminck, Hiep Luong, and Wilfried Philips. “Surface-based GICP”. In: *Conference On Computer and Robot Vision (CRV)*. Toronto, Canada: IEEE, 2018.

[Walker 91] Michael W. Walker, Lejun Shao, and Richard A. Volz. “Estimating 3-D Location Parameters Using Dual Number Quaternions”. In: *CVGIP: Image Underst.* 54.3 (Oct. 1991), pp. 358–367.

[Whelan 12] T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard, and J. McDonald. “Kintinuous: Spatially Extended KinectFusion”. In: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia, July 2012.

[Whelan 13] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. “Robust Real-Time Visual Odometry for Dense RGB-D Mapping”. In: *IEEE Intl. Conf. on Robotics and Automation, ICRA*. Karlsruhe, Germany, May 2013.

[Wohlkinger 11] Walter Wohlkinger and Markus Vincze. “Ensemble of shape functions for 3D object classification.” In: *ROBIO*. IEEE, 2011, pp. 2987–2992.

[Wu 13] Changchang Wu. “Towards Linear-Time Incremental Structure from Motion”. In: *Proceedings of the 2013 International Conference on 3D Vision*. 3DV ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 127–134. ISBN: 978-0-7695-5067-1.

[Wuest 05] Harald Wuest, Florent Vial, and Didier Stricker. “Adaptive line tracking with multiple hypotheses for augmented reality”. In: *Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR’05)* (2005), pp. 62–69.

[Wurm 11] Kai M. Wurm, Daniel Hennes, Dirk Holz, Radu Bogdan Rusu, Cyrill Stachniss, Kurt Konolige, and Wolfram Burgard. “Hierarchies of octrees for efficient 3D mapping”. In: *IROS*. IEEE, 2011, pp. 4249–4255.

[Xiao 13] Junhao Xiao, Benjamin Adler, Jianwei Zhang, and Houxiang Zhang. “Planar Segment Based Three-dimensional Point Cloud Registration in Outdoor Environments”. In: *Journal of Field Robotics* 30.4 (2013), pp. 552–582. ISSN: 1556-4967.

[Yang 13] Jiaolong Yang, Hongdong Li, and Yunde Jia. “Go-ICP: Solving 3D Registration Efficiently and Globally Optimally”. In: *2013 IEEE International Conference on Computer Vision (ICCV)*. 2013, pp. 1457–1464.

[Yang 17] Tsun-Yi Yang, Jo-Han Hsu, Yen-Yu Lin, and Yung-Yu Chuang. “DeepCD: Learning Deep Complementary Descriptors for Patch Representations”. In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[Yin 17a] Huan Yin, Xiaqing Ding, Li Tang, Yue Wang, and Rong Xiong. “Efficient 3D LIDAR based loop closing using deep neural network”. In: *Proceedings of the 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Dec. 2017, pp. 481–486.

[Yin 17b] Huan Yin, Yue Wang, Li Tang, Xiaqing Ding, and Rong Xiong. “LocNet: Global localization in 3D point clouds for mobile robots”. In: *Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV)*. Changshu, China, June 2017, pp. 26–30.

[Zach 07] Christopher Zach, Thomas Pock, and Horst Bischof. “A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration.” In: *ICCV*. IEEE Computer Society, 2007, pp. 1–8. ISBN: 978-1-4244-1631-8.

[Zaganidis 17] Anestis Zaganidis, Martin Magnusson, Tom Duckett, and Grzegorz Cielniak. “Semantic-assisted 3D Normal Distributions Transform for scan registration in environments with limited structure”. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 4064–4069.

[Zagoruyko 15] Sergey Zagoruyko and Nikos Komodakis. “Learning to Compare Image Patches via Convolutional Neural Networks”. In: *CoRR* abs/1504.03641 (2015).

[Zeng 17a] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. "3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions". In: *CVPR*. 2017.

[Zeng 17b] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas A. Funkhouser. "3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. 2017, pp. 199–208.

[Zhang 14] Ji Zhang and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time". In: *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*. 2014.

[Zhang 15] Ji Zhang and Sanjiv Singh. "Visual-lidar Odometry and Mapping: Low-drift, Robust, and Fast". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)* (May 2015).

[Zhong 09] Y. Zhong. "Intrinsic shape signatures: A shape descriptor for 3D object recognition". In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. Sept. 2009, pp. 689–696.

[Zhu 18] Zulun Zhu, Shaowu Yang, Huadong Dai, and Fu Li. "Loop Detection and Correction of 3D Laser-Based SLAM with Visual Information". In: *Proceedings of the 31st International Conference on Computer Animation and Social Agents*. CASA 2018. Beijing, China: ACM, 2018, pp. 53–58. ISBN: 978-1-4503-6376-1.

[Zlot 12] Robert Zlot and Michael Bosse. "Efficient Large-Scale 3D Mobile Mapping and Surface Reconstruction of an Underground Mine." In: *FSR*. Vol. 92. Springer Tracts in Advanced Robotics. Springer, 2012, pp. 479–493.