Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

Myklebust, Åsmund
Sæther, Edvard Ekrem
Nesseth, Nikolai Astad

Objektdeteksjon i punktskyer

Testrammeverk og analyse

Bacheloroppgave i BIAIS
Veileder: Kleppe, Adam Leon
Medveileder: Schaatun, Hans Georg
Mai 2024

NTNU
Kunnskap for en bedre verden

Myklebust, Åsmund
Sæther, Edvard Ekrem
Nesseth, Nikolai Astad

Objektdeteksjon i punktskyer

Testrammeverk og analyse

Bacheloroppgave i BIAIS
Veileder: Kleppe, Adam Leon
Medveileder: Schaatun, Hans Georg
Mai 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

NTNU
Kunnskap for en bedre verden

Institutt for IKT og realfag

AIS2900 - Bacheloroppgave ingeniørfag

Objektdeteksjon i punktskyer

Testrammeverk og analyse

Skrevet av:
Myklebust, Åsmund
Sæther, Edvard Ekrem
Nesseth, Nikolai Astad

Dato: 21. mai 2024

Forord

Denne bacheloroppgaven er gjennomført ved Institutt for IKT og realfag ved NTNU i Ålesund og markerer avslutningen på vår bachelorgrad i Informasjonsteknologi. Vi har jobbet med denne oppgaven gjennom hele vårsemesteret, fra januar til mai 2024.

Gjennom dette prosjektet har vi fått dykke ned i spennende utfordringer knyttet til objektdeteksjon i punktskyer. Målet vårt har vært å utvikle et testrammeverk og vurdere eksisterende algoritmer og metoder for å kunne hjelpe til med å forbedre nøyaktigheten og effektiviteten i transformasjonsestimering og objektdeteksjon i 3D-modellerte miljøer. Denne oppgaven har ikke bare vært en faglig utfordring, men også en mulighet til å bruke den teoretiske kunnskapen vår i praksis. Dette har gitt oss verdifulle erfaringer som vi tar med oss inn i våre fremtidige karrierer innen teknologi.

Vi vil gjerne takke våre veiledere, Adam Leon Kleppe og Hans Georg Schaatun, for deres verdifulle veiledning, støtte og engasjement gjennom hele prosjektet. Deres ekspertise og innsikt har vært avgjørende for dybden og kvaliteten på arbeidet vårt.

Ålesund, 21. mai 2024

Åsmund Myklebust
Edvard Ekrem Sæther
Nikolai Astad Nesseth

Sammendrag

Denne rapporten bygger på utfordringen med å registrere 3D-modeller i punktskyer fra et 3D-kamera, med fokus på hvordan man nøyaktig kan identifisere og plassere disse modellene. Prosjektet går ut på å utvikle et testrammeverk og vurdere eksisterende metoder for å se hvor nøyaktige og effektive de er, samt hva begrensningene deres er når det gjelder objektdeteksjon og transformasjonsestimering. Arbeidet er motivert av potensielle bruksområder innen robotikk og automatisert inspeksjon, hvor presis registrering er avgjørende.

Rapporten beskriver hvordan vi implementerte et testrammeverk for systematisk å evaluere og sammenligne ulike algoritmer under kontrollerte forhold. Dette inkluderer forberedelse av punktskydata ved bruk av både virkelige data fra et 3D-kamera og simulerte data fra Blender. Viktige evalueringskriterier er nøyaktighet, effektivitet og hvordan algoritmene håndterer støy og okklusjon.

Resultatene fra testene våre er dokumentert og analysert, og de gir innsikt i hvordan hver algoritme presterer under forskjellige forhold. Funnene peker også på områder som kan forbedres.

Abstract

This report builds upon the challenge of registering 3D models in point clouds obtained from a 3D camera, focusing on accurately identifying and positioning these models. The project involves developing a test framework and evaluating existing methods to determine their accuracy, efficiency, and limitations in object detection and transformation estimation. This work is motivated by potential applications in robotics and automated inspection, where precise registration is crucial.

The report describes how we implemented a test framework to systematically evaluate and compare various algorithms under controlled conditions. This includes preparing point cloud data using both real-world data from a 3D camera and simulated data from Blender. Key evaluation criteria are accuracy, efficiency, and the algorithms' ability to handle noise and occlusion.

Our test results are documented and analyzed, providing insights into each algorithm's performance under different conditions. The findings also highlight areas for improvement.

Innhold

Figurer

Kodesnutter

Tabeller

Terminologi

For å sikre klarhet i rapporten er noen viktige begreper og terminologi definert under:

3D-/dybdekamera En enhet som kan fange opp dybdeinformasjon, skape en punktsky som representerer tredimensjonale objekter.

Punktsky (eng: *Point cloud*) En samling av datapunkter i et tredimensjonalt rom som representerer objekters overflater.

Transformasjonsestimering Prosessen med å estimere et objekts orientering og posisjon i rommet, inkludert rotasjon og translasjon.

Iterative Closest Point (ICP) En algoritme for å minimere forskjellen mellom to punktskyer ved å iterativt forbedre en initiell transformasjonsestimat.

Random Sample Consensus (RANSAC) En metode for å estimere parametere av en modell ved å identifisere og utelukke outliers.

Evolusjonær algoritme En søkealgoritme som gjerne baserer seg på å etterligne genetikk eller biologiske prosesser, og anvendes for optimaliseringsproblemer.

Trekkbaserte metoder (eng. *Feature-based methods*) Teknikker som anvender distinkte egenskaper i data for å lette prosessen med objektdeteksjon eller mønstergjenkjenning.

Normalvektor En vektor som er ortogonal (vinkelrett) mot tangentplanet til et punkt på en overflate.

Uteliggere (eng. *Outliers*) Dataelementer som avviker markant fra de øvrige dataene, ofte som resultat av målefeil eller støy.

Point Cloud Library (PCL) Et stort åpen kildekode-bibliotek spesialisert for behandling av 2D/3D-bilde- og punktskydata.

Vokselnettbasert nedsampling (eng. *Voxel Grid Downsampling*) En teknikk for å redusere antallet datapunkter i en punktsky ved å dele rommet inn i en fast gitterstruktur av voxel og deretter konsolidere datapunkter innenfor hver voxel.

Egenverdier og egenvektor Matematiske konsepter brukt i analysen av 3D-objekter for å bestemme egenskapene som retninger hvor data varierer mest.

Segmentering Prosessen med å dele en punktsky inn i flere segmenter basert på kriterier som nærhet eller geometriske egenskaper, ofte brukt for å identifisere individuelle objekter.

Planoverflate (eng. *Mesh*) En samling av vertekser, kanter og flater som definerer formen til et 3D-objekt.

Okklusjon (eng. *Occlusion*) Fenomenet der deler av et objekt eller scene er blokkert fra synspunktet til et sensor- eller kamera, noe som gjør punkter usynlige i den resulterende punktskyen.

Registrering av punktskyer (eng. *Point Cloud Registration*) Prosessen med å alignere to eller flere punktskyer inn i ett koordinatsystem, ofte nødvendig i scenarier hvor flere observasjoner av samme scene er gjort fra forskjellige synspunkter.

Nedsampling av punktsky Teknikken for å redusere antall punkter i en punktsky, for å minske beregningskravene eller fjerne redundans.

Trekkutvinning (eng. *Feature Extraction*) Identifisering og utvelgelse av nyttige informasjonstrekk fra rådata, som kan brukes for videre analyse eller maskinlæringsmodeller.

Punkttetthet Mål av antall datapunkter per volumenhet innenfor en punktsky, viktig for analysekvalitet og overflatereskonstruksjon.

Støyreduksjon Teknikker brukt for å fjerne eller redusere støy i måledata, avgjørende for forbedring av nøyaktighet i punktskybehandling.

Kalibrering Prosessen med å justere målinger fra et måleinstrument for å sikre nøyaktighet. Dette gjøres ved å sammenligne instrumentets målinger med en kjent standard og justere instrumentet for å eliminere avvik, slik at det gir korrekte måleresultater.

Geometriske primitive former Enkle former som sfærer, sylinder, kuber og kjegler, som ofte brukes som grunnlag i geometrisk modellering og gjenkjenning.

Fotometri Måling av elektromagnetisk strålingsenergi.

Peer reviews Praksis hvor kode gjennomgås av gruppemedlemmer før den integreres i hovedprosjektet.

1 Introduksjon

Dette bachelorprosjektet tar for seg et uløst problem innen maskinsyn – nemlig registreringsproblemet. Vi undersøker hvordan man kan identifisere og nøyaktig plassere en 3D-modell i en punktsky, basert på data samlet inn fra fysiske og virtuelle 3D-kamera. Med økende interesse i industrien for teknologisk anvendelse som robotikk, virtuell virkelighet og automatisert inspeksjon [44][43][35], er målet vårt å bidra til feltet ved å utvikle et testrammeverk og teste eksisterende metoder, samt identifisere deres begrensninger. Hovedmålet er å utvikle et rammeverk som kan evaluere og sammenligne forskjellige algoritmer for transformasjonsestimering under kontrollerte forhold. Vi vil se på hvordan disse metodene presterer med hensyn til nøyaktighet og behandlingstid. Det er viktig å merke seg at vi ikke utvikler nye algoritmer fra bunnen av, men fokuserer på å enklere kunne evaluere eksisterende eller nye metoder under like forhold med samme testkriterier.

Registreringsproblemet er en utfordring innen datasyntese og maskinsyn [42]. Det handler om å finne den optimale endringen av transformasjonen av to eller flere punktskyer slik at de kan sammenstilles til en enkelt punktsky. Dette er spesielt relevant når data fra forskjellige synsvinkler eller sensorer skal kombineres for å skape et helhetlig bilde av et objekt eller miljø [27].

Å løse registreringsproblemet innebærer å finne korrespondanser mellom punkter i forskjellige datasett og deretter optimalisere overgangen mellom disse punktene for å minimere feil. Dette kan deles inn i to hovedkomponenter:

- **Deteksjon av korrespondanse:** Identifisere hvilke punkter i en punktsky som tilsvarer punkter i en annen. Dette kan være utfordrende på grunn av støy, varierende punktetthet og okklusjon.
- **Transformasjonsoptimering:** Finne den beste matematiske transformasjonen, som rotasjon og translasjon, som minimerer avstanden mellom tilsvarende punkter i de ulike punktskyene. Dette innebærer ofte komplekse, ikke-lineære minimeringsproblemer.

Evnen til å gjenkjenne objekter i rommet fra punktskybilder er viktig for mange teknologiske anvendelser. For eksempel kan det hjelpe roboter med å navigere og håndtere objekter i sitt arbeidsmiljø[33], forbedre brukeropplevelsen i VR ved å sikre troverdig interaksjon mellom virtuelle og fysiske objekter[59], og effektivisere automatiserte inspeksjonsprosesser ved å identifisere feil eller defekter, noe som kan øke både produktiviteten og sikkerheten[12].

Den største utfordringen vi utforsker er hvordan datamaskiner kan bli bedre til å identifisere og nøyaktig posisjonere 3D-modeller basert på data fra 3D-kameraer. Dette er vanskelig på grunn av flere faktorer:

- **Datakompleksitet:** Punktskyer kan inneholde millioner av punkter, noe som gjør databehandlingen krevende.
- **Støy og unøyaktigheter:** 3D-kameraer kan introdusere støy, som forvrenger punktene og gjør det vanskelig å identifisere korrespondanser.
- **Okklusjon i miljøet:** Objekter kan være delvis skjulte, eller miljøet kan endre seg over tid, noe som kompliserer registreringsprosessen.

Registreringsproblemet har vært gjenstand for mye forskning med mange tilnærminger[33][59][57]. Metoden ICP er en av de mest brukte[60], og flere studier har utforsket hvordan man kan forbedre denne metoden for å øke hastigheten og nøyaktigheten, som arbeidene til Besl & McKay [8] og Chen & Medioni [10].

1.1 Oppsett til prosjektrapporten

Denne rapporten er strukturert for å gi en grundig og systematisk oversikt over arbeidet med objektdeteksjon i punktskyer. Rapporten er delt inn i flere seksjoner, som dekker alt fra innledende planlegging til konklusjon. Her er en oversikt over hva leseren kan forvente i de ulike seksjonene:

- **Introduksjon 1:** Gir en oversikt over prosjektets bakgrunn, mål og problemstillinger. Her beskrives også prosjektets relevans og potensielle anvendelser innen ulike teknologiske områder som robotikk og automatisert inspeksjon.
- **Prosjektplanlegging:** Beskriver metodene og strategiene vi har brukt for å planlegge og gjennomføre prosjektet. Dette inkluderer ansvarsfordeling, tidsplanlegging, ressursstyring, risikohåndtering og kvalitetssikring.
- **Materialer:** Denne delen beskriver materialene og teknologiene vi har brukt i prosjektet. Vi benyttet ulike 3D-kameraer og datainnsamlingsutstyr for å samle inn punktskydata. Videre brukte vi spesifikke programvareverktøy og biblioteker for å utvikle og evaluere algoritmene våre.
- **Teori:** Presenterer relevant teori og metoder for forståelse av objektdeteksjon og transformasjonsestimering i 3D-punktskyer.
- **Metode:** Denne delen beskriver de metodiske tilnærmingene vi har brukt i prosjektet. Vi utviklet et testrammeverk for å systematisk evaluere ulike objektdeteksjons- og transformasjonsestimeringsalgoritmer. Datainnsamlingsprosessen involverte både fysiske og virtuelle 3D-kameraer, som ga oss et omfattende datasett å jobbe med.

- **Resultat:** Denne seksjonen er delt inn i 5 deler. Det består av punktskygenerering, testspesifisering og testgenerering, testrammeverk, testkjøring, og registreringsmetoder.
- **Diskusjon:** I denne delen diskuteres etikk, miljø og bærekraft, manglende og uferdige funksjoner, samt videre arbeid.
- **Konklusjon:** Oppsummerer hovedfunnene fra prosjektet, vurderer om vi har nådd forskningsmålene, og diskuterer betydningen av arbeidet. Seksjonen gir også forslag til fremtidig forskning og personlige refleksjoner fra prosjektet.
- **Referanser:** Inneholder en liste over alle referansene som vist til i rapporten.

Denne strukturen sikrer at leseren får en helhetlig forståelse av prosjektet, fra konseptutvikling til praktisk implementering og evaluering av resultatene. Dette gjør rapporten lett å følge og dekker alle viktige aspekter på en grundig måte.

1.2 Prosjektplanlegging

Vi har laget en plan for å styre og gjennomføre alle de nødvendige aktivitetene for å nå målene våre. Denne planen tar hensyn til tidsfrister, ressursbruk og risikohåndtering for å sikre at vi gjør fremskritt gjennom hele prosjektet.

1.2.1 Ansvarsfordeling

Gruppen har identifisert og fordelt hovedansvar blant medlemmene for å få en balansert arbeidsbelastning og optimal utnyttelse av hver enkelts ferdigheter:

- **Åsmund Myklebust:** Gruppeleder, ansvarlig for daglig loggføring, kommunikasjon med veileder og oppdragsgiver, samt 3D-design og produksjon av testobjekter.
- **Edvard Sæther:** Ansvarlig for utvikling og vedlikehold av testrammeverket for evaluering av estimeringsmetoder.
- **Nikolai Nesseth:** Fokuserer på innsamling av punktskydata, generering av testsett og dokumentasjon av møter og prosessfremdrift som Scrum Master.

1.2.2 Tidsplanlegging og milepæler

Prosjektet er strukturert med klare milepæler for å måle fremgang og eventuelt justere planer etter behov. Viktige datoer og frister inkluderer:

- **Oppstart og konfigurasjon av testplattform:** Dette skal være ferdig i løpet av de to første ukene
- **Testing av eksisterende algoritmer:** Midtveisgjennomgang etter en måned.

1.2.3 Risikohåndtering

For å unngå problemer underveis, har det blitt identifisert potensielle risikoer tidlig og laget strategier for å håndtere dem:

- **Teknisk risiko:** Feil i programvare eller maskinvare kan forsinke prosjektet. Forebyggende vedlikehold og tidlig testing vil anvendes for å håndtere denne risikoen. Det blir benyttet feilhåndtering i C++ med bruk av `exception`, `expected` og minneområder fordelt i egne prosesser, sammen med enhetstester for å validere at feilhånteringen fungerer som den skal.
- **Operasjonell risiko:** Gruppemedlemmers utilgjengelighet eller ressursmangel. Regelmessige møter og statusrapporter sikrer tidlig identifisering og adressering av slike problemer.

1.2.4 Kvalitetssikring

For å sikre høy kvalitet på arbeidet og resultatene, har gruppen:

- Brukt kodegjennomganger og "peer reviews" for å opprettholde høy kodekvalitet [5].
- Implementert kontinuerlige tester ved kompilering for å oppdage og rette feil tidlig i utviklingsfasen.

2 Teori

Dette kapittelet presenterer det teoretiske grunnlaget for objektdeteksjon og transformasjonsestimering i 3D-punktskyer. Teorien er viktig for å kunne forstå og utvikle effektive løsninger.

2.1 Smidige metoder

Smidige metoder er en gruppe utviklingsmetoder basert på iterative og inkrementelle prosesser[6]. Smidige metoder legger vekt på fleksibilitet, samarbeid og kundenes behov. Hovedprinsippene i smidige metoder inkluderer kontinuerlig levering av programvare med høy kvalitet, tilpasning til endrede krav, og tett samarbeid mellom utviklingsteamet og kunden.

Smidige metoder fremmer en smidig prosjektfremgang ved å dele opp prosjektet i små, håndterbare deler, og gjennomføre regelmessige evalueringer og tilbakemeldinger. Dette gjør det mulig for teamet å justere kursen etter behov, forbedre prosesser underveis, og sikre at prosjektet leverer verdi til kunden gjennom hele utviklingssyklusen [16]. Et eksempel på en slik arbeidsmetodikk er *Scrum*, som også har blitt benyttet i dette prosjektet.

2.2 Relevante teorier og metoder

Innen maskinsyn og 3D-bildebehandling, er flere teorier og metoder viktige for objektdeteksjon og transformasjonsestimering:

Iterative closest point (ICP): ICP [8] [10] [4] er en algoritme brukt for å minimere forskjellen mellom to punktskyer ved iterativt å forbedre transformasjonen som best beskriver dem. ICP er spesielt nyttig for finjustering av objektposisjonering i 3D-rommet, men kan være følsom for initiell posisjonering og lokale minimum. Se Figur 1 for en illustrasjon.

Random sample consensus (RANSAC): RANSAC [21] tilbyr en robust tilnærming til estimatproblemer innen datamodellering, spesielt nyttig for å håndtere data med signifikante andeler outliers. RANSAC er ofte brukt for grovestimering av objektets transformasjon før finjustering. Se Figur 2 for illustrasjon.

Evolusjonære algoritmer: Disse algoritmene simulerer biologiske prosesser som reproduksjon og mutasjon for å løse optimaliseringsproblemer [19]. I konteksten av maskinsyn kan evolusjonære algoritmer utforske et bredt søkeområde for å finne grensene for gyldigheten til en registrerings-

metode.

Trekkbaserte metoder: Disse metodene fokuserer på å identifisere unike trekk (features) i bildedata som hjørner, kanter, eller teksturer, som kan brukes for å finne korrespondanser mellom ulike sett av data. Feature-baserte metoder er viktige for å kunne initiere objektdeteksjon og transformasjonsestimering i punktskyer.

- **RSD** [37] [38] bruker en spesifikk radius rundt et referansepunkt på en overflate for å identifisere geometriske egenskaper. Ved å analysere forholdet mellom referansepunktet og dets nabopunkter, kan denne metoden karakterisere overflateegenskaper lokalt, se Figur 4. Dette er spesielt nyttig for applikasjoner som krever detaljert overflatedeteksjon eller teksturgjenkjenning [36].

- **3DSC** [23]er en utvidelse av den to-dimensjonale metoden Shape-Context til tre dimensjoner. Den bruker en sfære sentrert rundt punktet hvor beskrivelsen skal beregnes, og deler denne sfæren inn i volumer basert på asimut, høyde og en logaritmisk fordeling i den radielle dimensjonen for å håndtere skala-sensitivitet. Se Figur 7 for illustrasjon. Denne metoden er spesielt egnet for å fange opp geometriske strukturer i 3D-data og behandle romlig informasjon effektivt [22].

- **CVFH** [3] beregner et VFH-histogram (Viewpoint Feature Histogram) for hver region i stedet for et enkelt histogram for hele klyngen. Objektet deles inn i stabile, glatte områder ved hjelp av regionvoksende segmentering, som tar hensyn til avstandene og forskjellene i normalene til punktene i hver region. Se Figur 6 for illustrasjon. Dette gjør det mulig å finne et objekt i en scene så lenge minst en av regionene er synlig [2].

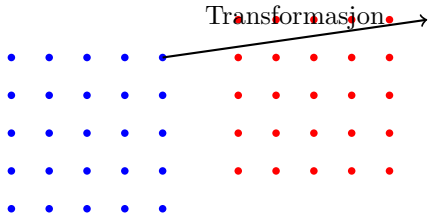
- **FPFH** [49] [51] forenkler beregningen av geometriske egenskaper ved å fokusere på umiddelbare naboer til hvert punkt i punktskyen. Først beregnes normalvektorer for hvert punkt, og deretter lages et forenklet histogram som tar hensyn til de direkte naboene. Se Figur 3 for illustrasjon. Dette innebærer å beregne vinkler mellom normalene til punktet og dets naboer samt avstandene mellom dem [48].

- **SHOT** [56] [54] beskriver lokal geometri rundt et punkt ved å analysere orienteringene til flere punkt innenfor et sfærisk nabolag. Hvert nøkkelpunkt får en lokal referanseramme basert på punktets normal og geometriske fordeling av nabolagspunktene, noe som gjør beskrivelsen rotasjonsinvariant. Nabolaget deles inn i sektorer, og for hvert volum beregnes et histogram basert på vinkelen mellom normalen ved nøkkelpunktet og vektoren fra nøkkelpunktet til nabolagspunktene. Se Figur 5 for illustrasjon. Disse histogrammene kombineres til en signatur som beskriver den lokale 3D-strukturen [53].

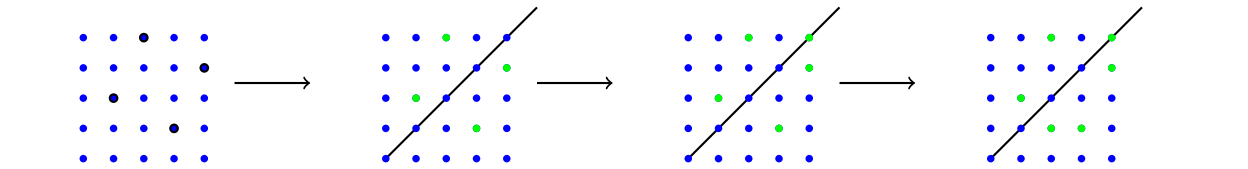
- **Zhao** [61] fungerer i to steg. I det første steget hentes det ut plan fra begge punktskyene. Dette gjøres ved å først pikselere punktskyene. Pikselering vil i dette tilfelle i praksis bety at punktskyen blir gjort om til et 2d-bilde. Bildet har verdier av dybde og hulrom. To pikselerte punktskyer blir videre brukt til å hente ut plan punktene danner i de. Disse planene blir brukt videre i selve registreringen. I det andre steget sammenlignes beskrivelsene av planene som ble funnet for å finne grupper med plan som stemmer overens med hverandre. Videre blir transformasjonen mellom disse to gruppene funnet for å til slutt finne transformasjon mellom de originale punktskyene. Se Figur 8 for illustrasjon.

2.2.1 Algoritmer

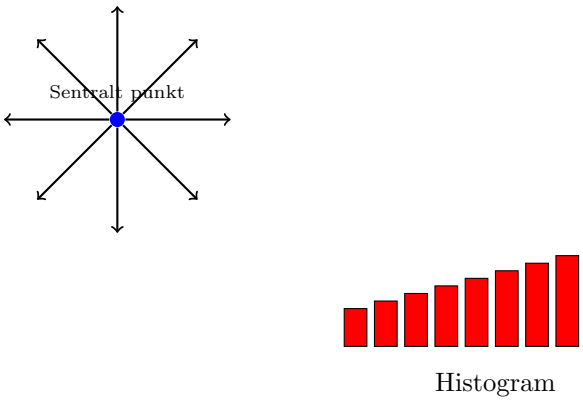
I dette underkapittelet kan man se illustrasjoner av metodene som har blitt forklart i Seksjon 2.2.



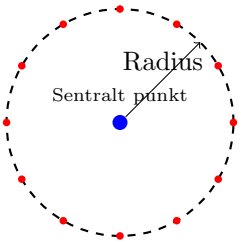
Figur 1: Illustrasjon av ICP-metoden hvor den blå punktskyen transformeres mot den røde for optimal registrering.



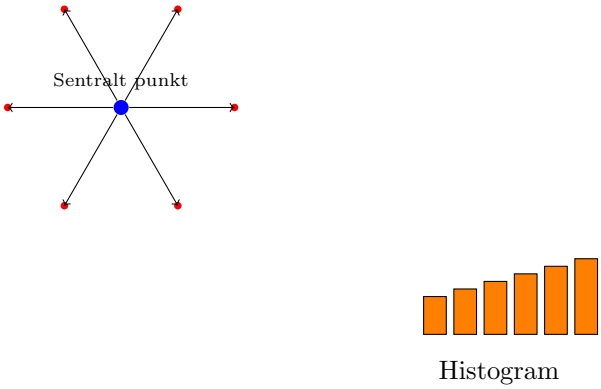
Figur 2: Illustrasjon av RANSAC-metoden



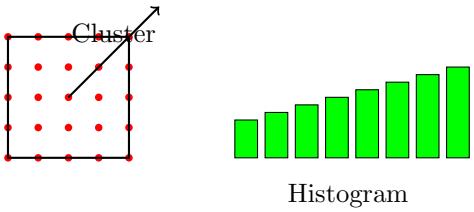
Figur 3: Visualisering av FPFH-beregning hvor lokale histogrammer beregnes basert på orienteringer og avstander fra et sentralt punkt



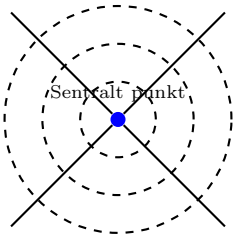
Figur 4: Illustrasjon av Radius-based Surface Descriptor (RSD) hvor lokal overflategeometri innen en fast radius blir analysert.



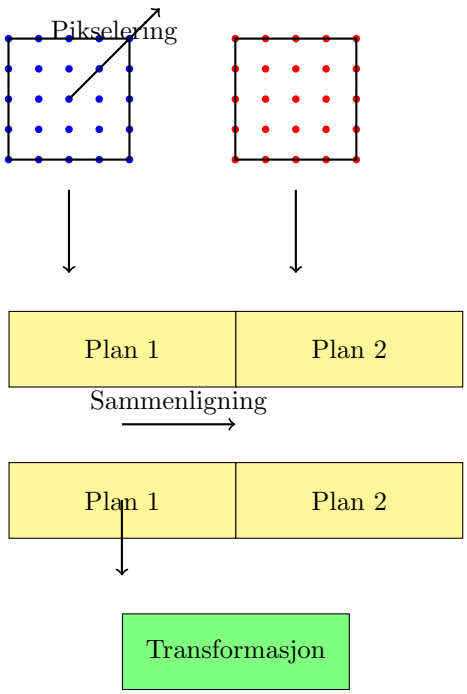
Figur 5: Visualisering av SHOT hvor lokale orienteringer og avstander rundt et sentralt punkt brukes til å generere et histogram.



Figur 6: Illustrasjon av Clustered Viewpoint Feature Histogram (CVFH) hvor punktskyen er segmentert i klynger og histogrammer genereres for hver.



Figur 7: Illustrasjon av 3D Shape Context (3DSC) hvor punktet er omgitt av et tredimensjonalt romlig histogram.



Figur 8: Illustrasjon av Zhao-metoden. Første steg: Pikselering av punktskyene og utvinning av plan. Andre steg: Sammenligning av planbeskrivelser og finne transformasjonen mellom dem.

2.3 Kritikk av eksisterende løsninger

Selv om de nevnte metodene tilbyr nyttige verktøy for objektdeteksjon og transformasjonsestimering, har hver av dem begrensninger. For eksempel så har ICP en følsomhet for initialisering av punktskyer [11]. Dette kan da føre til konvergens mot lokale minimum istedenfor globale minimum. Da kan det resultere i en lite gunstig løsning hvor startkonfigurasjonen ikke er nær den ideelle. RANSAC sin ytelse minsker med økende antall iterasjoner. RANSAC kan mislykkes i å få det beste resultatet eller finne den beste modellen viss dataen som er tilgjengelig ikke er god nok [20]. Genetiske algoritmer, selv om de er kraftige for globale søk, krever de mye beregningstid og ressurser siden de trenger mange iterasjoner. Dette gjør den lite egnet for scenarioer hvor det er nødvendig med hurtig beslutningstid. Genetiske algoritmer kan også stagnere ved lokale optimum og trenger da også en godt tilpasset fitness-funksjon, samt paramter-innstillinger som mutasjonsrate og populasjon [25]. Feature-baserte metoder er avhengige av kvaliteten og distinktheten av de identifiserte trekkene, som kan variere betydelig mellom ulike datasett. Så trekk-baserte metoder vil da fungere dårlig i forhold hvor det er lav belysning eller mye støy [34].

Ved å adressere disse begrensningene, kan prosjektet vårt utforske nye eller forbedrede metoder som kombinerer styrkene til eksisterende algoritmer mens de minimerer deres svakheter.

2.4 Transformasjonsestimering og punktsky-registrering

Transformasjonsestimering er en prosess for å posisjonere og orientere objekter korrekt i en punktsky [27]. I dette prosjektet fokuserer vi spesielt på å utvikle og evaluere algoritmer for nøyaktig og effektiv transformasjonsestimering.

Vår tilnærming evaluerer metodenes effektivitet og pålitelighet under varierte forhold, inkludert forskjellige nivåer av støy og kompleksitet i objektenes geometri. Dette arbeidet legger grunnlaget for videre forskning innen objektdeteksjon og kan potensielt forbedre teknologiske applikasjoner innen robotikk, automatisert inspeksjon og virtuell virkelighet.

2.5 3D-printing

I dette prosjektet har vi dratt nytte av 3D-printing, som lar oss lage fysiske objekter fra digitale modeller. Dette er spesielt nyttig for eksperimentene våre. Teknologien gjør det mulig å raskt lage og teste komplekse former og strukturer som ellers ville vært vanskelig eller dyrt å lage med tradisjonelle metoder. Vi har brukt 3D-printing for å lage testobjekter i ulike materialer og farger. Disse objektene har hjulpet oss med å teste og justere algoritmene vi bruker for objektdeteksjon i 3D-punktskyene våre. Vi har også sett på hvordan forskjellige materialer påvirker nøyaktigheten i målingene våre.

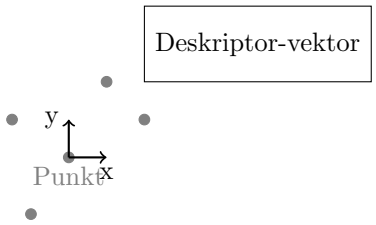
2.6 3D-kamera

For å få nøyaktige målinger og detaljerte punktskyer har vi brukt 3D-kameraer, spesielt Intel RealSense stereokamera. Dette kameraet fungerer bra til å fange opp dybdeinformasjon og gir oss en nøyaktig 3D-representasjon av objektene vi studerer. De fungerer ved å projisere infrarødt lys og måle refleksjonen for å bestemme avstanden til ulike punkter. Dette gir oss svært nøyaktige data, noe som er avgjørende for arbeidet vårt med å registrere og estimere transformasjoner i 3D-punktskyer. Vi brukte Intels egen programvare, RealSense Viewer, for å teste kameraet. Med dette programmet kunne vi se objektene i sanntid og justere parametere for å få de beste resultatene.

Vi brukte også Blender, et gratis og åpen kildekode-program for 3D-modellering. Blender er allsidig og blir brukt til alt fra å lage animasjoner og visuelle effekter til 3D-modeller og videospill. Programmet har mange funksjoner som hjelper oss med alle trinn i 3D-produksjonen, inkludert teksturering, grafikkredigering, rigging og rendering.

2.7 Deskriptor

En deskriptor er en vektor av tall som fanger opp viktig informasjon om et punkt og dets nabolag i en punktsky. Denne informasjonen kan inkludere form, størrelse, tekstur, og orientering av overflaten nær punktet. Ved å konvertere lokale data til en mer håndterbar form, muliggjør deskriptorer effektiv sammenligning av punkter mellom forskjellige punktskyer [27] [26].

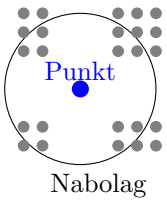


Figur 9: Illustrasjon av en deskriptor for et punkt i en punktsky.

I punktskyer er en deskriptor et verktøy som hjelper til med å identifisere og skille punkter eller områder basert på deres geometriske og fotometriske egenskaper. Deskriptorer er spesielt viktige i behandlingen av punktskydata fordi de tilbyr en informativ representasjon av lokale egenskaper rundt hvert punkt i punktskyen [27] [26].

2.8 Punkt i punktsky

Når vi snakker om punktskyer, mener vi alle de små datapunktene som til sammen danner en 3D-modell av et objekt eller et miljø. Hvert punkt viser en del av overflaten til objektet og er plassert i et tredimensjonalt rom. I prosjektet vårt bruker vi punktskyer som vi lager med 3D-kameraer. Disse punktskyene kan inneholde tusenvis, eller til og med millioner, av punkter. Å behandle disse punktene er en utfordring, særlig når det gjelder å redusere støy, filtrere og segmentere dataene. Dette må gjøres for å få bedre datakvalitet før vi kan analysere dem videre [15].



Figur 10: Visning av et punkt og dets nabolag i en punktsky.

3 Materiale

Dette kapittelet gir en oversikt over utstyret, materialene og teknologiene som er anvendt i prosjektet for å oppnå de definerte målene. Det inkluderer informasjon om programvarer, kodebibliotek, samt hvilke datamaskiner som har blitt benyttet i prosessen.

3.1 Programvare

I dette delkapittelet presenteres programvarene og utviklingsverktøyene som ble brukt i prosjektet. Disse verktøyene var avgjørende for utviklingen av testrammeverket og den generelle håndteringen av kode og prosjektsamarbeid. Programvaren som er benyttet er listet i Tabell 1. Kodebibliotekene som er benyttet direkte er listet i Tabell 2. Merk at bibliotekene som er listet opp kan kreve andre biblioteker for å fungere, men disse er ikke brukt direkte. Til slutt lister Tabell 3 biblioteker som enten ble brukt på et tidspunkt i løpet av utviklingen, eller biblioteker som kan være nyttige for å forbedre programmet.

Programvare	**Formål**	**Bruksområde**
CLion	Integrert utviklingsmiljø (IDE) for C++	Brukt for all hovedkoding og testing av C++ kode
Blender	3D-modelleringsprogram	Anvendt for visualisering og testing av 3D-modeller og punktskyer
Blainder [46]	Tilleggsprogram for Blender	Brukt til å generere punktskyer fra simulerte dybdesensor-kamera
Intel RealSense Viewer	Programvare for kjøring av Intel RealSense dybdekameraer	3D-visualisering i sanntid for kamera og eksportering av genererte punktskyer
GitHub	Versjonskontrollsystem	Brukt for kodesamarbeid, versjonskontroll og kildekodestyring
GNU GCC	Kompiler for C og C++	Brukt for å kompilere prosjektet på Linux-baserte systemer
MSVC	Microsoft Visual C++ Kompiler	Brukt for å kompilere og kjøre prosjektet på Windows
Python3	Programmeringsspråk	Skripting og automatisering av diverse oppgaver
C++(23)	Programmeringsspråk	Hovedspråk for testrammeverket
Prusaslicer	3d printing	Gjøre om .stl-filer til g-kode for å kunne 3d printe objekter
Overleaf	Rapportskriving	Skrive rapporten og loggføre testing, samt informasjon
Discord	Kommunikasjonsplattform	Kommunisere med hverandre og dele artikler, koding og relevant informasjon.
Confluence	Dokumentasjon	Dokumentere møtereferat, samt kontroll på issues i JIRA
JIRA	Dokumentasjon & planlegging	Planlegge og dokumentere arbeidet på prosjektet

Tabell 1: Oversikt over programvare som er brukt

Programvare	Formål	Bruksområde
PCL (Point Cloud Library) [50]	Bibliotek for behandling av punktskyer	Brukt for å behandle og analysere 3D-punktskydata
Eigen [30]	Bibliotek for lineær algebra	Håndtering av matematiske operasjoner innen transformasjon og manipulering av data
Boost.DLL [47]	C++ bibliotek for å behandle dynamiske bibliotek	Brukt for å laste inn registreringsmetoder fra bibliotekfiler
Boost.Interprocess [24]	C++ bibliotek for kommunikasjon mellom prosesser	Brukt for å sette opp delte minneområder
POSIX [28]	C bibliotek for blant annet prosessbehandling på POSIX-systemer	Brukt for å starte, overvåke og terminere arbeidsprosesser på POSIX-systemer
Win32 processthreadsapi [39]	C bibliotek for prosessbehandling på Windows-systemer	Brukt for å starte, overvåke og terminere arbeidsprosesser på POSIX systemer
Glaze [7]	C++ bibliotek for lesing av JSON-filer	Lese filer for inndata til testrammeverket
libharu [18]	C bibliotek for skriving av PDF-filer	Lage rapporter av testresultat

Tabell 2: Oversikt over brukte kodebiblioteker

Programvare	Formål	Bruksområde
Boost.QVM [17]	Bibliotek for kvaternioner, vektorer og matriser	Kan bli brukt istedenfor Eigen og egne funksjoner
Boost.Process [40]	Bibliotek for kryssplattform prosessbehandling	Hadde ikke funksjonene som var ønsket. Ble byttet ut med POSIX og Win32
Boost.Lockfree [9]	Bibliotek for datastrukturer for bruk av flere tråder samtidig	Hadde ikke funksjonene som var ønsket

Tabell 3: Oversikt over tidligere og fremtidige kodebiblioteker som kan benyttes

3.2 Datamaskiner

I denne seksjonen kan man se alle de ulike private datamaskinene gruppen har benyttet gjennom utførelsen av bachelor-oppgaven. Her ser man både spesifikasjon og om hvorvidt det er en bærbar eller stasjonær maskin. Det ble benyttet flere datamaskiner med ulike spesifikasjoner for å se hvordan testingen av algoritmene ble påvirket. På denne måten fikk vi se når prosesseringstiden var enten dårligst eller best og eventuelt om testing kræsjet på noen av datamaskinene. Spesifikasjonene er gitt i Tabell 4, 5, 6, 7 og 8.

3.3 Utstyr

I dette delkapittelet beskrives utstyret som ble brukt for datagenerering og eksperimentering for prosjektet i Tabell 9.

Komponent	Verdi
Type	Laptop, Asus VivoBook S14
Hovedprosessor	Intel Core i5-1135G7 (4 fysiske kjerner, 8 logiske kjerner, 4,2 GHz)
Grafikkprosessor	Intel iRISxe
Arbeidsminne	16 GB, 3200 MHz
Operativsystem	Windows 11 Home, versjon 23H2

Tabell 4: Spesifikasjoner til datamaskin 1

Komponent	Verdi
Type	Laptop, Lenovo IdeaPad Pro 5 14IRH8
Hovedprosessor	Intel i5 13500H (12 fysiske kjerner, 16 logiske kjerner, 4,7 GHz)
Grafikkprosessor	Intel Iris Xe
Arbeidsminne	16 GB, 5200 MHz
Operativsystem	Windows 11 Home, versjon 23H2

Tabell 5: Spesifikasjoner til datamaskin 2

Komponent	Verdi
Type	Stasjonær
Hovedprosessor	Intel i7 6700K (4 fysiske kjerner, 8 logiske kjerner, 4,2 GHz)
Grafikkprosessor	NVIDIA GeForce GTX 1080
Arbeidsminne	48 GB, 2400 MHz
Operativsystem	Arch Linux 6.8.4

Tabell 6: Spesifikasjoner til datamaskin 3

Komponent	Verdi
Type	Laptop, Lenovo Legion 5
Hovedprosessor	AMD Ryzen 7 5800H (8 kjerner, 4,4 GHz)
Grafikkprosessor	NVIDIA GeForce RTX 3070
Arbeidsminne	16 GB, 3200 MHz
Operativsystem	Windows 11 Home, versjon 23H2

Tabell 7: Spesifikasjoner til datamaskin 4

Komponent	Verdi
Type	Stasjonær
Hovedprosessor	Intel i7 12700K (12 fysiske kjerner, 20 logiske kjerner, 5,0 GHz)
Grafikkprosessor	AMD Radeon RX 6800 XT
Arbeidsminne	32 GB, 5600 MHz
Operativsystem	Windows 11 Education, versjon 23H2

Tabell 8: Spesifikasjoner til datamaskin 5

Utstyr	Beskrivelse
Intel RealSense D435f	Stereo dybdekamera brukt til å fange 3D-punkskyer av fysisker objekter og scener.
Kamerastativ	Gir flere muligheter til posisjonering av kameraet og sikrer stabilisering under datainnsamling.
PETG	Type av plast brukt for 3D-utskrifter som er del av eksperimentelle oppsett.
PLA	En annen type biologisk nedbrytbar plast brukt til 3D-printing for å skape modeller og deler til testing.
Polariseringsfilter	Brukt på kameraet for å redusere refleksjoner og forbedre bildekvaliteten under visse forhold.
3D printer, prusa MK3S+	Brukt for å 3D printe objekter

Tabell 9: Oversikt over utstyr

4 Metode

Dette kapittelet gir en oversikt over hvordan vi planla og gjennomførte arbeidsprosessene og utviklingsstrategiene våre.

4.1 Scrum

I vårt prosjekt benyttet vi *Scrum* som vår arbeidsmetodikk. Scrum er et rammeverk for å implemetere smidige arbeidsmetoder, og består av faste iterasjoner kalt *Sprinter*, samt en strukturert tilnærming til samarbeid og kommunikasjon i teamet. Vi satte opp hver sprint til å vare i to uker, hvor vi skulle gjennomføre et sett med oppgaver vi hadde sett for oss. Dette ga os en jevn rytme for planlegging, gjennomføring og evaluering av arbeidet. For å organisere arbeidet brukte vi Jira til å administrere sprints og spore oppgaver, og Confluence til å ta møtenotater og holde oversikt over prosjektet generelt.

Et av gruppemedlemmene tok rollen som *Scrum Master* hvor ansvarene inkluderte å sette opp Scrum-møter og sikre at prosessen ble fulgt. Denne rollen fungerte som en sikkerhet for gruppens fremdrift og bidro til å optimalisere prosjektutviklingen. Siden oppdragsgiver for prosjektet har vært NTNU, tok veilederene rollen som kunde. I tillegg sørget de for å kontinuerlig gi konkrete tilbakemeldinger og veiledning, noe som hjalp oss med å justere arbeidet i henhold til behovene.

I starten av hver sprint diskuterte gruppen hvilke oppgaver som skulle prioriteres de neste to ukene i et sprint planleggingsmøte. Dette møtet hjalp oss med å etablere klare mål og sikre at alle teammedlemmene hadde samme forståelse for arbeidet som skulle gjøres. Vi valgte å velge bort daglige standups, men opprettholdt i stedet lav terskel for samarbeid og diskusjon rundt oppgavene gjennom hele sprinten. I slutten av hver to-ukers sprint, holdt vi en *Sprint Review*. Dette møtet var todelt: først diskuterte gruppen intert hva som hadde blitt gjort og hva som ikke hadde blitt oppnådd. Deretter møtte vi veilederne for å presentere fremdriften og eventuelle spørsmål vi trengte svar på. Her mottok vi tilbakemelding på arbeidet vårt og råd om hvordan vi kunne løse eventuelle problemer som hadde oppstått. Etter Sprint Review gjennomførte vi en *Sprint Retrospective*, hvor vi reflekterte over prosessene våre, identifiserte forbedringsområder og implementerte tiltak for å forbedre fremtidige sprints. Dette hjalp oss med kontinuerlig forbedring av teamets arbeidsmåter.

4.2 Utvikling

Testrammeverket ble designet for å kunne laste inn og kjøre tester på ulike registreringsmetoder. Testene ble spesifisert i egne filer, slik at programmet ikke trengte å rekompileres hver gang. Vi sørget for at både programmet og metodene kunne kjøres på forskjellige plattformer og operativ-

systemer.

4.2.1 Lisens

Det var viktig for oss å overholde alle lisenskravene for programvarekomponentene vi brukte. Testrammeverket vårt benytter flere åpen kildekodebiblioteker og verktøy, hver med sine spesifikke lisensbetingelser. Hovedbibliotekene, som PCL og Boost, er distribuert under BSD-lisensen og Boost Software License, som gir oss stor frihet til å modifisere og distribuere koden, så lenge vi beholder opphavsrettsinformasjonen og lisensvilkårene.

4.2.2 Strategi

Vi passet på å unngå lisenskonflikter mellom de forskjellige bibliotekene ved å sjekke om lisensene var kompatible med hverandre. Dette gjør det enkelt å distribuere testrammeverket uten problemer.

Vi fokuserte på lisenskompatibilitet for å være sikre på at vi kunne kombinere og dele ulike programvarebiblioteker uten juridiske utfordringer. Mange åpne kildekodebibliotek har forskjellige lisenskrav. Noen lisenser tillater fri bruk og modifikasjon, mens andre kan ha restriksjoner som påvirker hvordan koden kan distribueres eller kombineres med annen programvare.

For å unngå slike konflikter, gikk vi gjennom lisensene til alle bibliotekene vi ønsket å bruke. Vi leste betingelsene og sammenlignet dem for å sikre at de kunne fungere sammen. Vi sørget for at alle lisensene tillot fri bruk, modifikasjon og distribusjon i tråd med våre prosjektmål.

Ved å gjøre dette unngikk vi problemer som kunne hindret oss i å distribuere testrammeverket. Det sikret også at vi kunne dele arbeidet med andre utviklere og forskere uten bekymringer om lisensproblemer.

4.2.3 Samlede datasett

I prosjektet er det satt opp ett datasett for hvert objekt som er benyttet. Hvert datasett inneholder en mengde punktskyer hvor det gjeldende objektet er transformert med ulike kombinasjoner av translasjon og rotasjon. I tillegg har alle datasett en undermappe som inneholder punktsky-filen som skal benyttes som referanse hvor hver metode. For de syntetiske punktskyene er dette hvor objektet er plassert i $(0, 0, -1)$ uten rotasjon.

`c_tx0y5e-2z-1rx-15y0z0.ply`

Figur 11: Eksempel på navngiving av en punktsky-fil

Vi brukte en jevn fordeling av rotasjoner og translasjoner for å sikre variasjon og representativitet i datasettene, og alle punktskyene er lagret i *.ply*-format. Vi utviklet en spesifikk navngivningsstruktur for å inkludere all nødvendig informasjon om objektets translasjon og rotasjon i filnavnet til punktskyene. For å unngå desimaltegn i filnavnet, ble desimaltall skrevet som tierpotens. I eksempelet vist i Figur 11 representerer c et kubisk objekt, med translasjon $(0, 0.05, -1)$cm og rotasjon $(-15, 0, 0)^\circ$. I dette prosjektet har det blitt benyttet fem ulike objekter, som vist i Figur 20 og 14. Hver av disse har følgende bokstavreferanser:

c - Kube

f - Sylinder

s - Kule

t - Torus

m - 3D-modell

4.3 Metode for evaluering

Dette delkapitlet beskriver hvordan vi evaluerte metodene for objektdeteksjon og transformasjonsestimering. Vi forklarer hvilke kriterier vi brukte for å måle suksess og hvordan vi sammenlignet de forskjellige algoritmene.

4.3.1 Evaluering av ytelse

For å vurdere hvor godt algoritmene presterte, brukte vi følgende kriterier:

- **Nøyaktighet:** Vi målte hvor godt algoritmene klarte å identifisere og plassere objekter i punktskyer ved å sammenligne de estimerte posisjonene med kjente startverdier, posisjoner, rotasjoner og translasjoner.
- **Effektivitet:** Vi så på hvor lang tid det tok for hver algoritme å prosessere data og levere resultater, inkludert både oppsettstid og kjøretid under ulike forhold.
- **Robusthet mot støy:** Vi testet hvordan algoritmene håndterte punktskyer med ulike nivåer av støy, og hvor mye dette påvirket nøyaktigheten deres.

- **Håndtering av okklusjon og variabel belysning:** Vi undersøkte hvordan algoritmene presterte når objekter delvis var skjult eller under forskjellige lysforhold.

Vi kvantifiserte ytelsen til hver algoritme gjennom eksperimentelle tester under kontrollerte forhold og dokumenterte resultatene for å kunne sammenligne dem.

4.3.2 Statistiske metoder

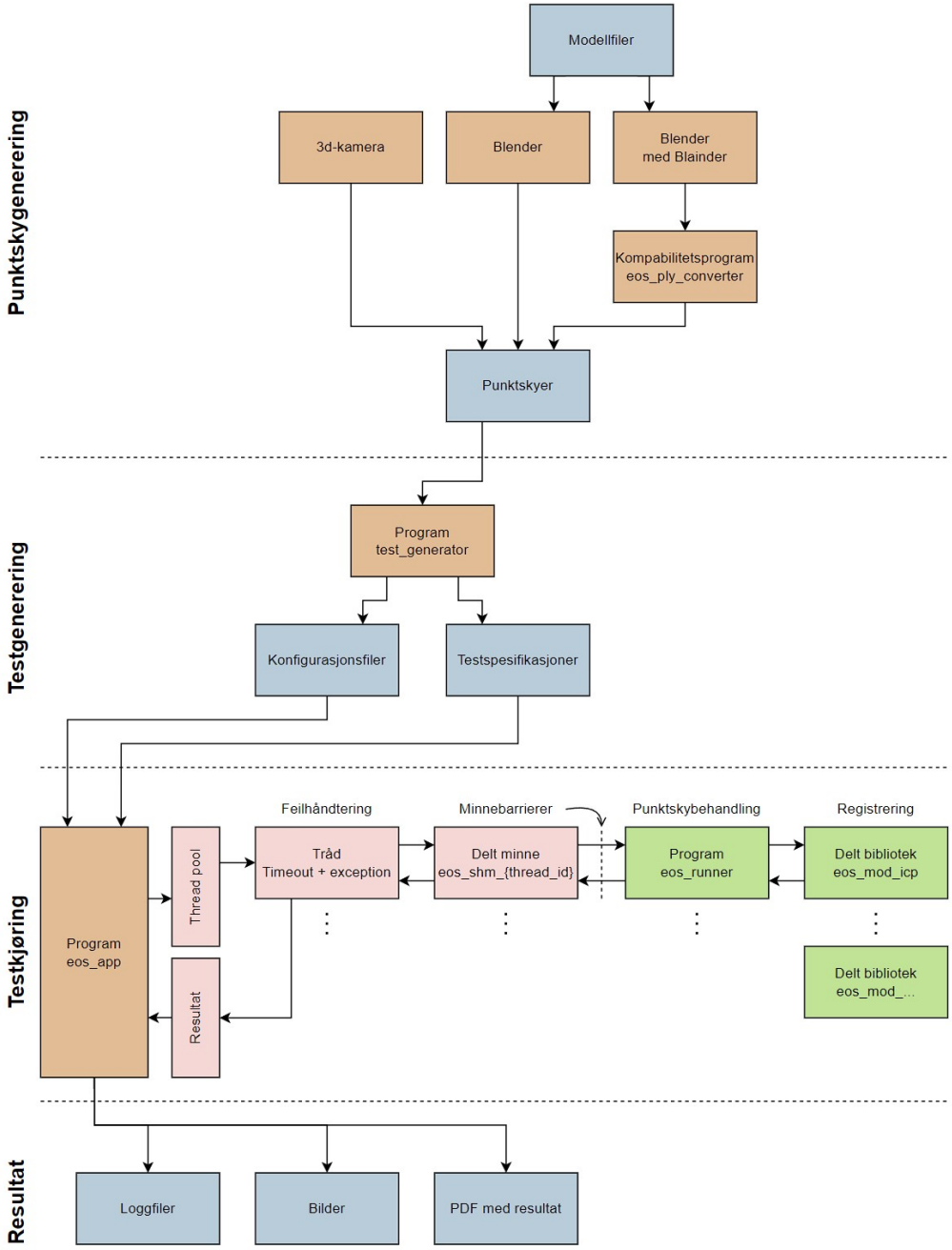
For å sikre at resultatene våre var pålitelige, brukte vi flere statistiske metoder:

- **Gjentatte eksperimenter:** Vi testet hver algoritme flere ganger under de samme forholdene for å måle hvor konsistente ytelsene var.
- **Prosentandeler:** Vi beregnet prosentandeler for hver modul og bibliotek i hver algoritme for å få en indikasjon på hvor nøyaktige de var i forskjellige bruksscenarioer.

Disse metodene hjalp oss med å validere resultatene og ga oss en klar forståelse av hver algoritmes styrker og svakheter.

4.4 Oppdeling

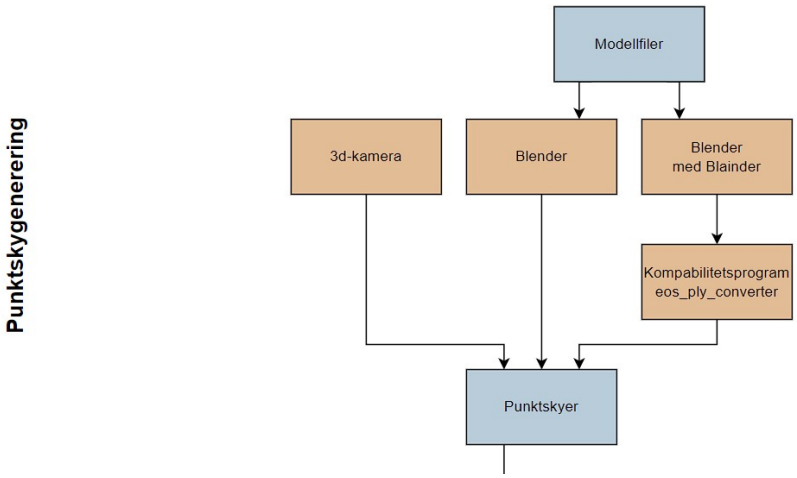
Som nevnt i introduksjonen kan prosjektet deles opp i fire deler. Disse delene og forholdet mellom de er vist i Figur 12. De fem neste kapitlene vil ta for seg disse delene hver for seg, hvor *Testkjøring* deles i to kapitler.



Figur 12: Komplett illustrasjon av prosessen og testrammeverket

5 Punktskygenerering

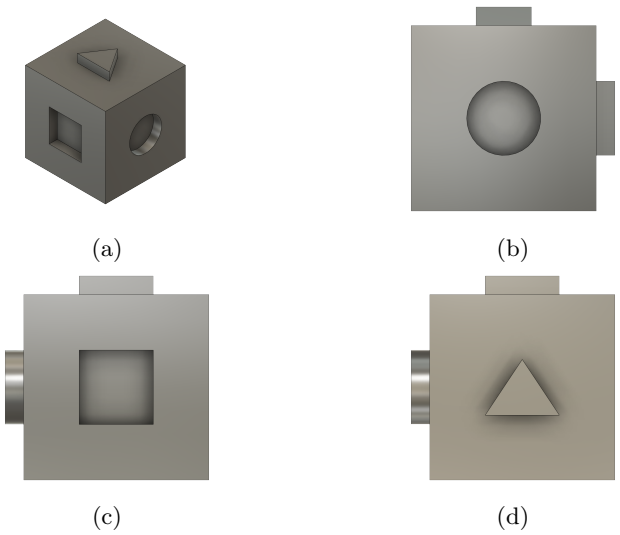
Dette kapitlet tar for seg punktskygenerering, og innebærer flere deler. Første del er valg av overflatemodeller som skal bli gjort om til punktskyer. Videre blir det vist hvordan modellene ble 3D-printet, og kamera brukt for å lage punktskyer av de. Senere er det vist hvordan Blender ble brukt for å generere punktskyer. Til slutt er et kompabilitetsprogram beskrevet for å kunne bruke noen av punktskyene. En oversikt over denne delen av prosjektet er vist i Figur 13. Denne figuren inkluderer ikke modelleringen og 3d-printingen som et steg.



Figur 13: Illustrasjon av dataflyten i testrammeverket – punktskygenerering

5.1 Design

Første design av objektet vårt som man kan se i Figur 14 ble 3d-printet i plastmaterialet PLA på manulab. I regi av skolens labfasiliteter, har NTNU et svært begrenset antall ruller og fargevarianter/nyanser i PLA. Dette gjorde at gruppen printet kuben i fargene som var tilgjengelig, altså svart og rød PLA. Resultatet av den første printen viste seg å passe dårlig til oppgaven. Vi så øyeblikkelig at fargen på kuben hadde for mye gjenskinn, som gjorde at kameraet ikke hadde mulighet til å produsere en god nok punktsky av objektet. Etter å ha testet objektene en stund, gikk vi så til materiallabben i kjelleren på lanterna for å prøve å spraymale kubene med primer og matt sort farge. Vi lot kubene ligge på materiallabben og tørke over natten. Neste dag plukket vi dem med oss og testet så objektene på nytt for å se om resultatet endret seg. Vi kunne se overflatene og de distinkte geometriske formene noe bedre, men allikevel ikke ett optimalt utgangspunkt.



Figur 14: Illustrasjon av første iterasjon av 3D-objektet i Fusion 360

Videre kom vi til konklusjonen om at objektet burde printes ut i typen matt-finish av lyse fargetyper, som for eksempel lys blåfarge og oransje. I tillegg tenkte gruppen at det kunne vært lurt å testet med en annen type plast materiale, i tilfelle det kunne ha noe å si for refleksjonen i overflatene på objektet. Så da bestilte vi inn en rull av typen matt PLA i blå, og en rull av typen matt PETG i oransje.

Før gruppen printet kubene i de nye fargene og materialene ble det tatt en avgjørelse om å redesigne objektene. Avgjørelsen ble tatt etter grundig testing av hvordan punktskyene så ut i programvaren til intel, altså intel realsense viewer. Denne programvaren ble benyttet for å se 3d visualisering av punktskyene i sanntid. Når vi testet kameraet og programvaren på kuben som vi hadde printet i starten, så vi at kantene på kuben ble automatisk avrundet og på grunn av refleksjon, ble flatene bølgete. Da tok vi en beslutning på å selv avrunde alle kanter og hjørner på kubene og på geometrien på selve kubene. Slik at vi kunne se bort i fra den feilkilden når vi så på punktskyer.

Etter å ha re-designet kuben i Fusion 360, ble de printet i de nye fargene og materialene som ble bestilt inn. Når printen var ferdig, så vi at den oransje fargen i material PETG var like blank og hadde mye glans i overflatene. Dermed var den like dårlig som de første printene vi hadde. Den andre kuben som ble printet i matt blåfarge i PLA var mye bedre. Denne ble mer matt i fargen enn vanlige fargetyper og ble dermed mer egnet til vårt formål.

I tabellene under kan man se hvilke parameter som ble valgt for printingen av objektene:

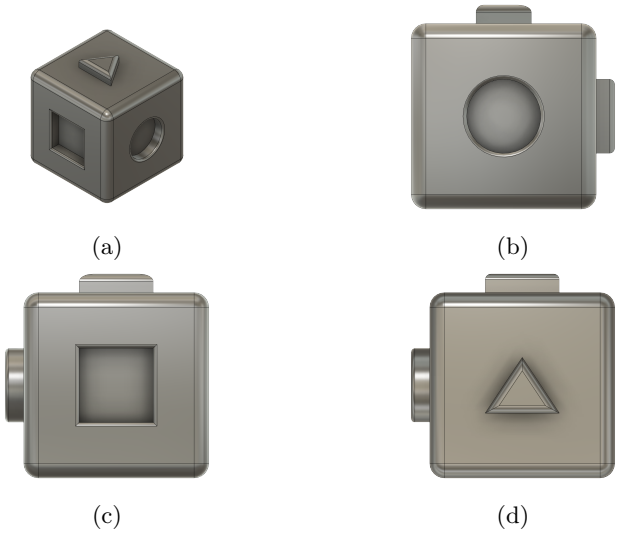
3D-printer innstillinger for matt PLA:

Parameter	Verdi
Bed temp	55
Nozzle temp	200
Printing speed	50mm/s
Infill %	5
Support	Organisk

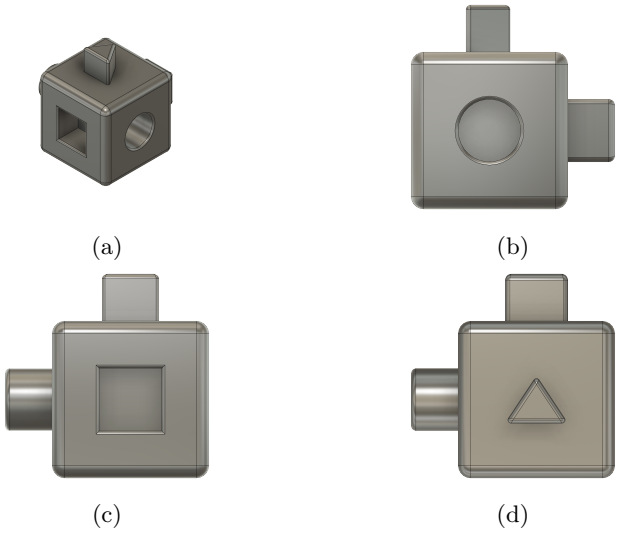
3d-printer innstillinger for matt PETG:

Parameter	Verdi
Bed temp	75
Nozzle temp	240
Printing speed	40mm/s
Infill %	5
Support	Orgnaisk

Her er bilder som illustrerer videre iterasjoner av det første objektet som ble 3D-printet:



Figur 15: Illustrasjon av neste iterasjon av 3D-objektet i Fusion 360

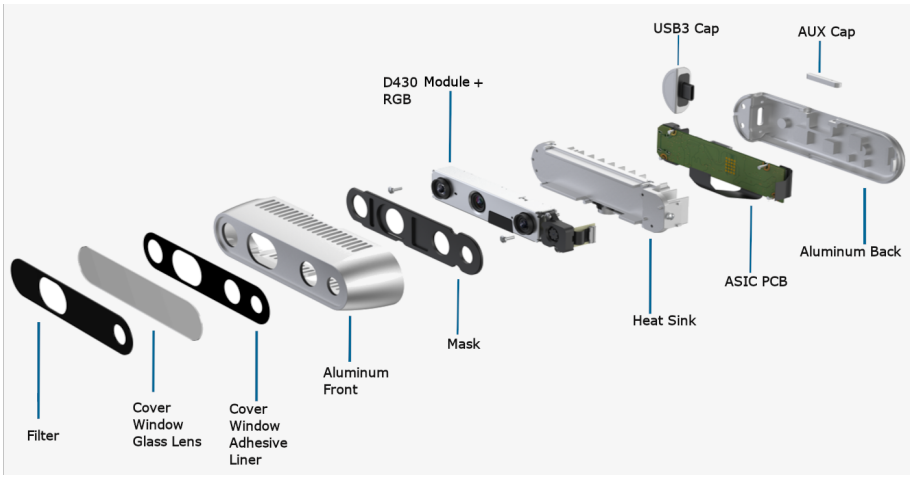


(a) (b) (c) (d)

Figur 16: Illustrasjon av nyeste iterasjon av 3D-objektet i Fusion 360

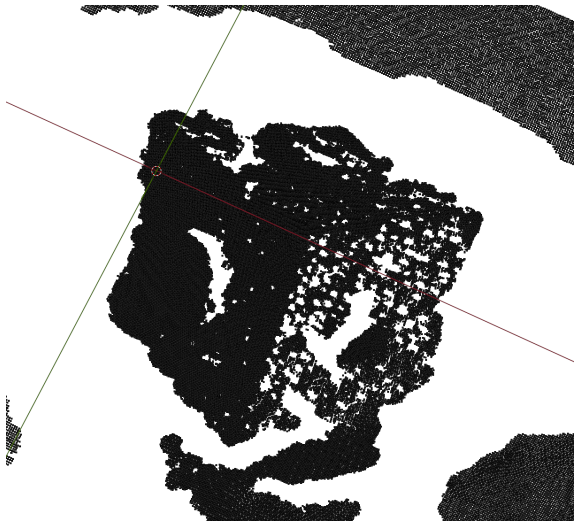
5.2 Bruk av 3D-kamera

I prosjektet har det blitt benyttet et stereo-kamera av typen Intel RealSense D435f for datainnsamlingen. Dette kameraet ble valgt for dets evne til å generere nøyaktige punktskyer av objekter i ulike miljøer.



Figur 17: Intel RealSense D435f [45]

I den innledende fasen av prosjektet ble Intel RealSense Viewer-programvaren benyttet til å få sanntidsvisualisering av punktskyen som ble generert av kameraet. Dette ga en forståelse av stereo-kameraets funksjonalitet og var essensielt for å bygge opp kunnskap om innhenting av 3D-punktskyer og for å danne en forståelse for hvordan objekter og overflater blir fanget opp. Denne innsikten var nyttig for å forme våre tilnærminger og metodikk for videre eksperimenter.



Figur 18: Punktsky av modell vist i figur 14 generert fra RealSense D435f kamera

Senere i prosessen ble kameraet brukt til å generere mer komplekse punktskyer av virkelige scener. Dette ga muligheten til å utføre grundige tester av objektdeteksjonsalgoritmene under realistiske forhold. Disse testene bidro til å validere ytelsen og påliteligheten til rammeverket for objektdeteksjon i 3D-punktskyer.

Spesifikasjon	**Verdi**
Modell	Intel RealSense D435f
Synsfelt (FOV)	87°x58° for dybde, 69°x42° for farge
Dybdeoppløsning	1280 x 720
Fargeoppløsning	1920 x 1080
Maksimum bildetakt	90 fps for dybde, 30 fps for farge
Infrarødt (IR) projektor	Ja
Dybdeteknologi	Stereoskopisk, global lukker
Dybdenøyaktighet	<2% ved 2m
Tilkoblinger	USB 3.1 Type-C
Strømforbruk	5V, via USB
Dimensjoner	90 mm x 25,8 mm x 25 mm

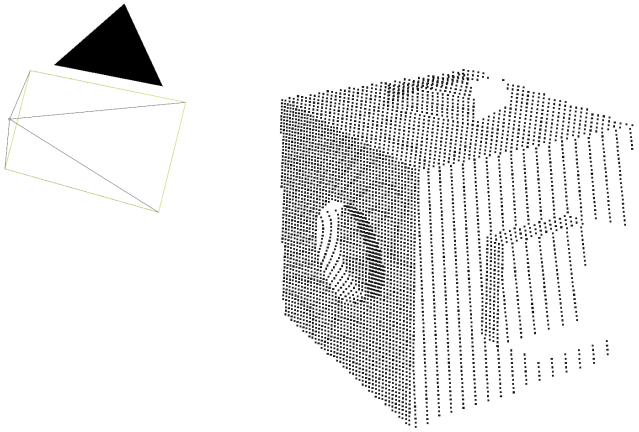
Tabell 10: Tekniske spesifikasjoner for Intel RealSense D435f [45]

5.3 Punktskygenerering i Blender

Punktskygenerering er en sentral del av vårt prosjekt, som involverer å lage detaljerte 3D-representasjoner av objekter for å teste og evaluere algoritmer for objektdeteksjon. Ved å bruke Blender, en kraftig åpen kildekode-programvare for 3D-modellering, har vi vært i stand til å generere både syntetiske og virkelighetsnære punktskyer. Denne prosessen har inkludert både bruk av kameraperspektiv for å simulere virkelige scener og generering av punktskyer uten kameraperspektiv for spesifikke testformål. Ved å variere kompleksiteten og egenskapene til de genererte punktskyene, sikrer vi at rammeverket kan teste objektdeteksjonsmetodene under ulike forhold og utfordringer.

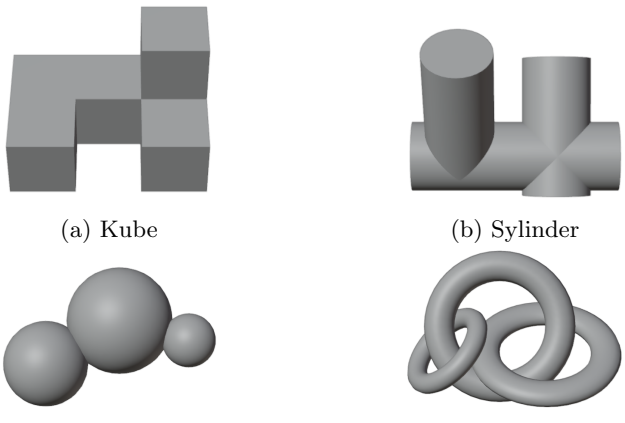
5.3.1 Med kameraperspektiv

Blender har for det meste blitt benyttet til å generere punktskyer i kontrollerte simulerte miljøer med varierende kompleksitet. Dette ble oppnådd ved å konstruere scenen, og deretter fange den opp fra et virtuelt dybdekamera ved hjelp av tilleggsprogramvaren *Blainder* [46], som vist i Figur 19.



Figur 19: Punktsky av modell vist i Figur 14 generert fra et virtuelt 3D-kamera i Blender vha. Blainder

Innledningsvis ble fire enkle virtuelle figurer med grunnleggende egenskaper benyttet til å danne et datasett med punkskyer. Disse figurene ble modellert i Blender og ble valgt for å representere forskjellige geometriske former som er relevante for testing av egenskapene til algoritmene for objektdeteksjon i rammeverket.



(a) Kube (b) Sylinder

(c) Kule (d) Torus

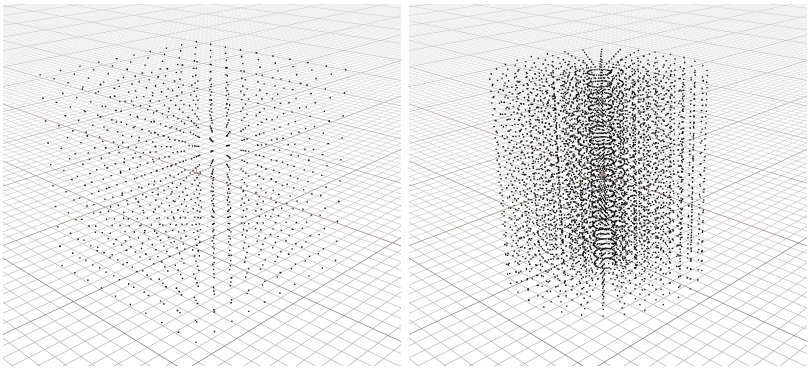
Figur 20: Figurer med grunnleggende geometriske egenskaper

Ved å presentere figurene med forskjellige geometriske egenskaper, eller en kombinasjon av dem, ble det mulig å analysere hvordan algoritmene utførte objektdeteksjon i hvert scenario. Denne til-

nærmingen gjorde det mulig å identifisere og vurdere algoritmene i deres evne til å håndtere ulike typer objekter og former i 3D-punkskyene. I tillegg er de designet slik at de ikke har flere identiske rotasjoner for å unngå falske negative tester.

5.3.2 Uten kameraperspektiv

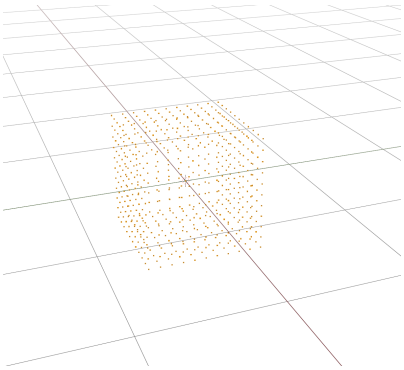
Det er laget noen enkle punktskyer uten tanke på kameraperspektiv. Disse punktskyene er i hovedsak brukt til selvtestingen. Figur 21a viser en kube-formet punktsky med volum, mens Figur 21b viser en sylinder-formet punktsky med volum. Det er ingen spesifikk grunn til hvorfor akkurat disse formene ble valgt fremfor andre, men kuben er spesielt tenkt for å teste posisjonsuavhengig støy mens sylinderen er spesielt tenkt for å teste posisjonsavhengig støy. Noen av fordelene med å bruke disse formene er at det er enkelt å se spredningen ved å se med ortografisk kameravisning fra koordinataksene. En annen form som kan være aktuell å bruke er en sfære, men det ble ikke gjort. Grunnen til at det ikke ble testet å visualisere støy med en sfære er at vi synes det var vanskelig nok å vurdere støyen med sylinderen. Sylinderen har kun to akser hvor punktene kurver seg langs med, mens en sfære vil ha tre akser.



(a) 10x10x10 rutenett av punkt (b) 9 ringer med 32 punkt, 19 lag

Figur 21: Illustrasjon av punktskyer for å teste støyfunksjonene

En annen punktsky som ble laget i Blender uten tanke på kameraperspektiv er en annen testkube. Denne kuben er vist i Figur 22. Punktskyen består av punkter på overflaten til en kube. Punktene er likt fordelt på hver side. En side av kuben er gjort om til et 11 gange 11 rutenett av punkter. Totalt er det 602 punkter. Denne formel ble valgt siden den inneholder flate overflater og skarpe vinkler. Ulempen med denne punktskyen er at den kan roteres på mange måter og fremdeles være en like god løsning ved bruk i registrering. Bruk av kuben i registrering var bruksområdet til den. Den inneholder relativt få punkter, og registreringsmetodene kan fort finne en løsning, selv om den kanskje ikke er rett. Kuben ble også brukt underveis i utviklingen av testrammeverket ellers som en midlertidig punktsky mens de andre som faktisk skal brukes til testing ble laget.



Figur 22: Enkel punktsky av overflaten til en kube

5.4 Kompabilitetsprogram

Punktskyfilene som ble laget med bruk av Blainder kan ikke brukes direkte av PCL. PLY-leseren til PCL støtter kun x-, y- og z-koordinater i formatet `float` [52]. Blainder produserer PLY-filer med `double` som format. Et eksempel på toppteksten før punktdataen er vist i Kodesnutt 1. For å løse dette problemet ble et eget kompabilitetsprogram laget. Dette programmet laster inn alle filene i en mappe og gjør de om til det støttede formatet. Etter konvertering er toppteksten endret til slik det er vist i Kodesnutt 2. Kommentaren på linje 3 er gjort tre bokstaver lengre siden forskjellen på ordlengden mellom *double* og *float* er én bokstav og det er tre ord som byttes. Dette gjør at punktdataen starter på den samme plassen i filen som den originalt gjorde.

Punktdataen blir lest inn sekvensielt fra starten og rundet av til en `float`-verdi før den blir skrevet tilbake til filen. Siden en `float` kun tar opp halve mengden data som en `double`, må i tillegg to pekere brukes i konverteringen. Den første peker på hvor den skal lese neste verdi, mens den andre peker på hvor den skal skrive resultatet etter konvertering. Ved å bruke to pekere trenger man ikke å lese inn hele filen inn i arbeidsminne før den skrives tilbake igjen. Til slutt reduseres filstørrelsen slik den ender hvor skrivepekeren var når lesepekeren nådde enden av filen.

Dette programmet fungerer bare vellykket på Linux. På Windows halverer ikke punktdataen seg. Det er mulig at Windows håndterer filpekere på en annen måte enn Linux gjør. Det kan være at Windows har egne buffere for lesing og skriving, og at dette tukler med logikken. Nøyaktig hvorfor programmet ikke fungerer på Windows er ikke funnet ut, men det kan være det kan gjøres små endringer som gjør at det fungerer.

```
ply
format binary_little_endian 1.0
comment Created by Open3D
element vertex 25342
property double x
property double y
property double z
end_header
```

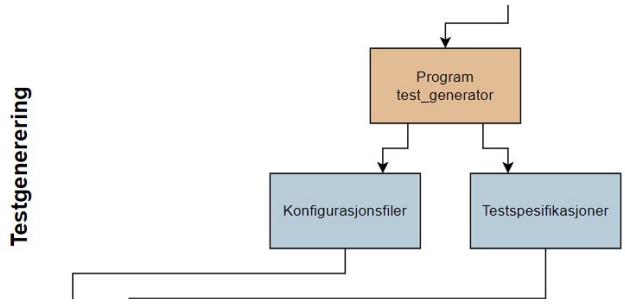
Kodesnutt 1: Eksempel på toppteksten i en punktskyfil før konvertering

```
ply
format binary_little_endian 1.0
comment Created for eos_app!
element vertex 25342
property float x
property float y
property float z
end_header
```

Kodesnutt 2: Eksempel på toppteksten i en punktskyfil etter konvertering

6 Testspesifisering og testgenerering

Denne seksjonen beskriver hvordan testene spesifiseres og hvordan de kan genereres. Figur 23 viser en grov oversikt.



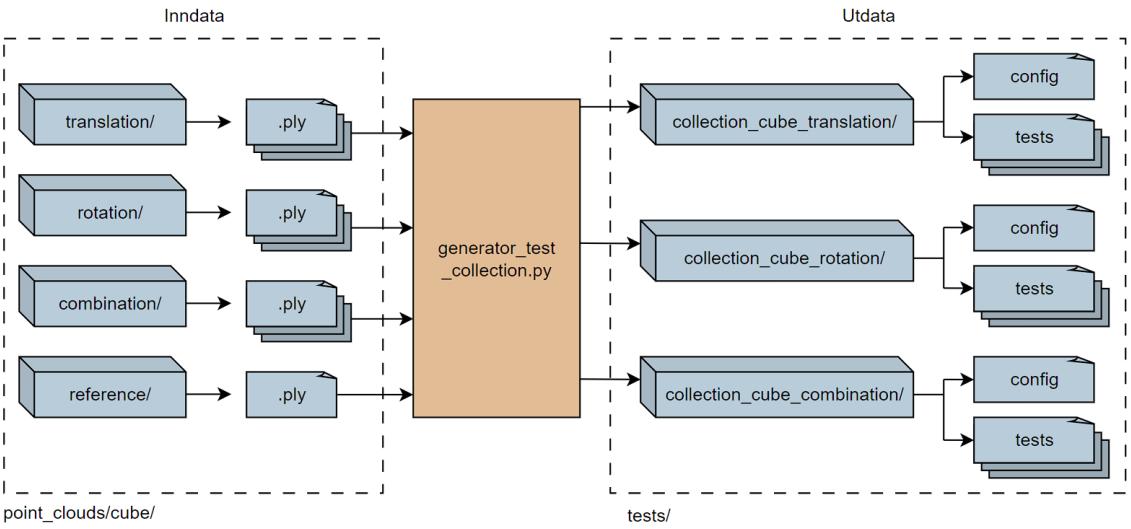
Figur 23: Illustrasjon av dataflyten i testrammeverket – testgenerering

6.1 Funksjoner

All inndata for hovedprogrammet ligger i en mappe ved siden av programmet, hvor undermapper for tester er kalt kolleksjoner. En kolleksjon kan inneholde en eller flere tester, definert i testfiler med valgfritt navn som ligger sammen med en konfigurasjonsfil. Testfilene må være i et forhåndsdefinert JSON-format, som er fleksibelt nok til å inneholde flere tester mellom to spesifikke punktskyer. Formatet for testfilene er vist i Kodesnutt 3. Feltet `world_transforms` er en liste som kan inneholde flere elementer med lignende struktur. Mange av feltene har standardverdier og trenger ikke spesifiseres med mindre det er spesifikke krav. For eksempel, om feltet `scale` mangler, vil en standardverdi på $(1, 1, 1)$ brukes.

6.2 Virkemåte

Som nevnt i delkapittel 4.2.3, inneholder prosjektet en mengde datasett, ett for hvert objekt som har blitt benyttet. Datasettene er delt inn i undermapper basert på hvilke transformasjoner som er påført objektet i punktskyen. I rammeverket er det utviklet en Python-kode kalt `generator_test_collection.py` som genererer testkolleksjoner fra disse datasettene. Hver av mappene i datasettene føres inn i testgenereringen, som danner en testfil for hver punktsky og samler dem i testkolleksjoner, som vist i Figur 24. Dette gjentas for hvert datasett.



Figur 24: Eksempel på hvordan kolleksjoner av testsett dannes fra punktskyene i et datasett

Formålet med en testfil er å instruere testrammeverket om hvordan metodene for objektdeteksjon skal testes, og hvilke resultater i transformasjon som forventes. Hver testfil er strukturert som vist i Kodesnutt 3. Den refererer først til punktskyen som benyttes som referanse (`world_file`), som angir hva metodene skal se etter i testen. Deretter defineres `target_file`, som er punktskyen hvor metodene testes for deres evne til å detektere objektet i en ny transformasjon. Både `world_file` og `target_file` inneholder informasjon om translasjon, rotasjon og skalering av objektet i de respektive punktskyene, hvor `target_file` spesifiserer dette under `expect`. Disse opplysningene hentes fra filnavnene deres, som beskrevet i delkapittel 4.2.3.

I `target_file` finnes det også muligheter til å utvide en test ved å definere variablene `steps` og `step_size`, som angir hvor mange tester som skal gjennomføres per testfil og hvor stor forflytning det skal være mellom hver av dem. For å teste algoritmenes robusthet, er det mulig å legge til ekstra støy i punktskyene. Ulike typer støy som er implementert i dette prosjektet beskrives i delkapittel 7.2.3. Hver testkolleksjon inneholder også en konfigurasjonsfil, `config.json`. Den informerer testrammeverket om hvilke metoder for objektdeteksjon som skal benyttes for den aktuelle kolleksjonen, og hvor testfilene er lokalisert i mappestrukturen.

6.3 Diskusjon

En utfordring vi møtte under utviklingen av testrammeverket var å finne passende navn for feltene `world` og `target`, slik at de på en enkel måte gir innsikt i deres betydning. Vi vurderte flere alternative navn, og kom frem til at static"og dynamickunne være bedre alternativer. Navnet static"indikerer referansepunktet eller den opprinnelige tilstanden til objektet, mens dynamicrrepresenterer objektet i den transformerte tilstanden. Disse alternative navnene kan gjøre det lettere

```
{
  "test_name": "Example of a test file",
  "world": {
    "file": "world_file.ply",
    "transform": {
      "affine": {
        "translation": { "x": 0.0, "y": 0.0, "z": 0.0 },
        "rotation": { "alpha": 0.0, "axis": { "x": 1.0, "y": 0.0, "z": 0.0 } },
        "scale": { "x": 1.0, "y": 1.0, "z": 1.0 }
      }
    }
  },
  "target": { "file": "target_file.ply" },
  "world_transforms": [
    {
      "affine": {
        "translation": {
          "mirrored": false, "steps": 3, "step_size": 0.1,
          "origin": { "x": 0, "y": 0, "z": 0 },
          "direction": {  "x": 1, "y": 1, "z": 1 }
        },
        "rotation": {
          "mirrored": true, "steps": 180, "step_size": 1,
          "alpha": 0, "axis": { "x": 0, "y": 1, "z": 0 }
        },
        "scale": {
          "steps": 10, "step_size": 0.01,
          "origin": { "x": 1, "y": 1, "z": 1 },
          "direction": { "x": 0, "y": 1, "z": 0 }
        }
      },
      "noise": { "seed": 12345, "type": "normal", "param1": 0.001 },
      "projective_noise": {
        "noise_z": { "seed": 23456, "type": "normal", "param1": 0.00005 },
        "noise_x": { "seed": 34567, "type": "normal", "param1": 0.00002 },
        "noise_y": { "seed": 45678, "type": "normal", "param1": 0.00001 }
      }
    }
  ],
  "expect": {
    "relation": "equal-world-transform",
    "transform": {
      "affine": {
        "translation": { "x": 1.0, "y": 0.0, "z": 0.0 },
        "rotation": { "alpha": 5.0, "axis": { "x": 0.8, "y": 0.4, "z": 0.0 } },
        "scale": { "x": 1.0, "y": 1.0, "z": 1.0 }
      }
    }
  }
}
```

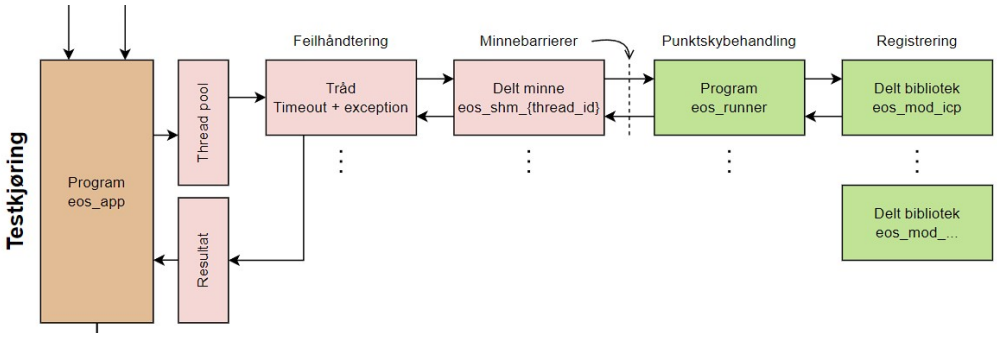
Kodesnutt 3: Format til en testfil

for brukerne av rammeverket å forstå testfilene uten behov for ytterligere forklaringer.

Et forbedringsområde i vårt rammeverk er håndteringen av støy. Foreløpig er det ikke mulig å sette inn støy med steg slik som det er ved translasjoner. Dette skyldes at ønsket økning av støy ofte ikke følger en lineær funksjon, slik tilfellet er med andre variasjoner. For å håndtere dette anbefaler vi å implementere egne steglengder i generator-scriptet `generator_test_collection.py`. Ved å gjøre dette kan brukeren spesifisere hvordan støyen skal økes på en måte som er tilpasset deres spesifikke behov og testscenarier. Dette vil gjøre rammeverket mer fleksibelt og bedre tilpasset til å simulere virkelige forhold.

7 Testrammeverk og testkjøring

Testrammeverket som er utviklet i dette prosjektet, er designet for å systematisk evaluere og sammenligne ytelsen til forskjellige registreringsmetoder. Dette gir en plattform for automatisert testing hvor flere algoritmer kan lastes inn, kjøres, og evalueres under kontrollerte og lignende forhold.



Figur 25: Illustrasjon av dataflyten i testrammeverket – testkjøring

7.1 Funksjoner

En første funksjon i testrammeverket er enkelheten av å teste nye registreringsmetoder. Registreringsmetoder kan lastes inn i testrammeverket som et delt bibliotek. Det eneste en implementasjon trenger å gjøre er å utføre ønskede operasjoner i en forhåndsdefinert funksjon. Denne funksjoner tar inn to punktskyer og en liste med valgfrie argumenter. De valgfrie elementene kan for eksempel brukes til å definere en søkeradius, om enkelte funksjoner skal brukes eller andre egenskaper en registreringsmetode kan ta nytte av. Navnet på det delte biblioteket sammen med et sett av valgfrie argumenter er gitt i filer med navn `config.json`. Denne konfigurasjonsfilen må ligge i en undermappe i mappen testrammeverket kjører testene fra. Konfigurasjonsfilen kan også definere flere biblioteker som skal brukes i mappen filen befinner seg i, eller flere ulike sett av parametere en metode skal kjøres med.

En mulighet en testfil har er å endre på transformasjonen mellom punktskyene før registrering blir utført mellom de. Den første muligheten for en transformasjonen er en affin transformasjon. Testrammeverket støtter å kunne endre på translasjonen, rotasjonen og skaleringen til en av filene. Alle feltene har valgfrie felt for en steglengde og antall steg. Disse feltene er kalt `steps` og `step_size`. Translasjon- og rotasjonsfeltet har også et valg for speiling med `mirrored` feltet. Om dette feltet har en verdi `true`, vil steglengen virke med både positivt og negativt fortegn. Antall steg vil da bli: $2 \cdot \texttt{steps} - 1$. For translasjon virker steglengden på `direction`-feltet, og endringen virker på `origin`-feltet. Det samme gjelder for skalering. For rotasjon virker steglengden på feltet `alpha`, som er rotasjonsvinkelen i grader rundt aksen spesifisert av feltet `axis`. Operasjonene skjer i rekkefølgen skalering, rotering og til slutt translasjon.

Vi har valgt å implementere to måter å legge til støy. Den første metoden er vanlig posisjonsuavhengig støy. Det er denne typen støy det blir mest testet i mot i artikler som beskriver nye registreringsmetoder [55] [58], med bruk av en normalfordeling. Denne typen støy legger forflytter hvert punkt i punktskyen med en felles fordeling og parametere til denne fordelingen. Fordelingene som støttes er uniformfordelingen, eksponensialfordelingen, gammafordelingen, weibullfordelingen, normalfordelingen, log-normalfordelingen og cauchy-fordelingen. Programmet vil gi en feilmelding dersom ugyldige parametere blir gitt til en fordeling. Normalfordelingen kan for eksempel ikke ha et negativt standardavvik. Posisjonsuavhengig støy vil bruke den angitte fordelingen på x-, y- og z-aksen.

Når testene kjøres kan de parallelliseres. Hvordan de kan parallelliseres er bestemt av hver enkelt konfigurasjonsfil i en kolleksjon. Tester kan ikke parallelliseres på tvers av kolleksjoner. Hovedbruken av feltet `execution_cost` i en konfigurasjonsfil er å indikere at registreringsmetoden det gjelder vil selv starte en eller flere tråder. Standard verdi er 1, som vil si at metoden bruker kun tråden testrammeverket håndterer. Under kjøringen vises fremgang i konsollen. Konsollen viser også rådata fra testene sammen med eventuelle meldinger fra registreringsmetodene som kjøres.

Etter kjøringen av testene er utført vil resultatene bearbeides. De ulike dataene vil bli skrevet til 4 ulike filer. Den første filen `log.txt` inneholder innholdet til alle filene. Den andre filen er `log_error.txt`, som inneholder meldinger fra feil som er håndtert. Til slutt inneholder `log_info.txt` rådataen fra å kjøre registreringsmetodene og `log_summary.txt` inneholder den resultatene aggregert per testfil. Loggfilene starter hver linje med dato og klokkeslett. Loggfilene ligger samlet i en mappe merket med dato og klokkeslett for når programmet ble startet, sammen med programnavnet. Disse mappene blir lagt inn i mappen `data/out/results/`.

Implementeringen av den automatiske rapportgenereringen har vist seg å være en betydelig forbedring for effektiviteten og nøyaktigheten i dokumentasjonsprosessen. Under testing og evaluering av ulike algoritmer for objektdeteksjon og transformasjonsestimering i punktskyer, har det automatiserte systemet vært i stand til å produsere detaljerte og konsistente rapporter. Dette har ikke bare spart oss tid, men også sikret en standardisert presentasjon av resultater, noe som er viktig for sammenligning og analyse.

De genererte rapportene inneholder omfattende data inkludert visualiseringer av referanse- og mål-punktskyer, transformasjonsmatriser, og prosesseringstid for hver testet metode. Denne informasjonen er viktig for å evaluere ytelsen til de ulike algoritmene. Figur 26 viser et eksempel på en slik automatisk generert rapport, som tydelig illustrerer plasseringen av objekter i bilder, samt detaljerte beregninger av tidsforbruk og transformasjonsnøyaktighet.

Resultatene fra de automatiske rapportene har vist seg å være pålitelige og konsistente, noe som

bekrefter effektiviteten til rapportgenererings-systemet. Dette har redusert risikoen for menneskelige feil og sikret at alle viktige aspekter ved testen blir dokumentert nøyaktig og grundig. Denne tilnærmingen har dermed forbedret kvaliteten på vår analyse og dokumentasjon, og fremstår som en verdifull komponent i prosjektets testrammeverk.

7.2 Virkemåte

I vårt testrammeverk er algoritmer implementert som separate moduler. Dette gjør det mulig for oss å tilpasse og bytte ut algoritmer uten å måtte omstrukturere hele systemet. Hver modul kan utvikles, testes og vedlikeholdes uavhengig av de andre, noe som forbedrer modulariteten og fleksibiliteten til vårt testrammeverk.

Modulene lastes dynamisk inn i testrammeverket ved oppstart. Dette gjøres ved hjelp av et dynamisk oppsett som tillater oss å injisere eksterne klasser og metoder direkte inn i kjøretidsmiljøet. Det gir oss mulighet til å oppdatere og legge til nye algoritmer uten å restarte eller rekonfigurere hele systemet.

For hvert testsett defineres spesifikke scenarier og parametere i en konfigurasjonsfil. Testrammeverket leser denne informasjonen og genererer automatiserte tester basert på de oppgitte kriteriene. Dette inkluderer valg av datasett, algoritmer som skal testes, og spesifikke parametere som påvirker algoritmeoppførselen.

Transformasjonsestimering er nøkkelen i vår prosess og moduler tillater justeringer av transformasjonsparametere som rotasjon og translasjon basert på algoritmenes tilbakemelding. Dette trinnet er vitalt for å simulere realistiske scenarioer der objekter ikke alltid er perfekt orientert i forhold til kameraet.

Testrammeverket introduserer også støy i punktskyene for å simulere virkelige observasjonsforhold. Vi bruker posisjonsuavhengig støy for å tilfeldig forskyve punkter, og sfærisk posisjonsavhengig støy for å skape variasjoner i punktdensiteten, noe som tester robustheten til hver algoritme under ulike støyforhold.

Testene kjøres i en thread pool for å optimalisere ressursbruken og effektivisere kjøretiden. Ved å distribuere testene over flere tråder, kan vi utføre flere oppgaver parallelt, noe som drastisk reduserer den totale testtiden.

Detaljert logging av hver testkjøring utføres for å spore algoritmenes ytelse og for å diagnostisere eventuelle feil.

Det ble utviklet et testrammeverk for å teste registreringsmetoder. Testrammeverket er delt inn i fem hoveddeler: konfigurasjon og spesifikasjonslesing, preprosessering, modulinnlasting, testkjøring, og resultatbehandling. Videre inneholder testrammeverket en rekke punktskyer som er brukt til testingen.

7.2.1 Konfigurering- og testspesifikasjons-filer

Testrammeverket baserer seg på brukerangitte filer i *json*-formatet. Det ble valgt å bruke kodebiblioteket *glaze* for å lese *.json* filer. Dette biblioteket ble valgt av flere grunner. Den første er at biblioteket bruker moderne funksjoner i C++. Denne viktigste funksjonen er kalt *reflection*/refleksjon, og er en del av foreslått teknisk spesifikasjon som senere kan bli inkludert i standarden for C++ [32] [29]. De fleste kompilatorer støtter allerede deler av virkemåten av denne tekniske spesifikasjonen, men uten et felles grensesnitt. *glaze* bruker du ulike grensesnittene internt for å kunne automatisk korrelere innholdet i et json objekt til en `struct` i C++. Dette gjør det svært enkelt å lage ønskede hierarkier med ulike datatyper. Andre grunner til at *glaze* ble valgt var tidligere kjennskap til og bruk av biblioteket og den gode lesehastigheten den har i forhold til andre biblioteker [7].

For å kunne laste inn en modul må filnavnet være kjent. Dette navnet er gitt i en konfigurasjonsfil. Kjøresettet er delt inn i kolleksjoner. Hver kolleksjon er identifisert av en konfigurasjonsfil. Denne konfigurasjonsfilen må hete `config.json`. Konfigurasjonsfilen kan angi at en eller flere moduler skal benyttes i kolleksjonen. Ulike sett med parametere til moduler kan også angis. Parametere av typene booleansk (`bool`), heltal (`std::int64_t`), flyttall (`double`) og tekst (`std::string`) kan angis. Konfigurasjonsfilen angir også mappen testene skal lete etter punktskyer.

Spesifikasjonslesingen i testrammeverket vårt er ansvarlig for å tolke og anvende konfigurasjonsfiler, som bestemmer parametre og innstillinger for hver test. Systemet benytter JSON-formaterte filer for å definere testscenarier, inkludert hvilke algoritmer som skal testes, deres konfigurasjonsparametere, og hvilke datasett som skal brukes. Dette trinnet sikrer at hver del av testen er korrekt initialisert etter spesifikasjonene, og gjør det mulig for oss å endre testbetingelser dynamisk uten å rekode eller rekompilere applikasjonen. En typisk spesifikasjonsfil inneholder informasjon om transformasjonsparametre, støyinnstillinger og detaljene om punktskydataene som skal brukes.

```
// (Noen tester)
for (std::int32_t i = 0; i < t_translation.steps; ++i)
{
    t_translations.emplace( t_translation.origin.x
        + i * t_translation.direction->x * *t_translation.step_size,
         // Lignende for y
        // Lignende for z
        );
    if (t_translation.mirrored && *t_translation.mirrored && i != 0)
    {
        t_translations.emplace( t_translation.origin.x
            - i * t_translation.direction->x * *t_translation.step_size,
            // Lignende for y
            // Lignende for z
        );
    }
}
```

Kodesnutt 4: Spesifikasjonslesing – translasjon

```
// (Noen tester)
for (std::uint32_t i = 0; i < t_rotation.steps; ++i)
{
    t_rotations.emplace(
        normalize_axis({t_rotation.alpha + i * *t_rotation.step_size,
                        t_rotation.axis}));
    if (t_rotation.mirrored && *t_rotation.mirrored && i != 0)
    {
        t_rotations.emplace(
            normalize_axis({t_rotation.alpha - i * *t_rotation.step_size,
                            t_rotation.axis}));
    }
}
```

Kodesnutt 5: Spesifikasjonslesing – rotasjon

```
std::vector<std::vector<test_spec>> test_specs;
// (Gjenta resterende kode for alle testsett i nåværende kolleksjon)
test_specs.emplace_back();
// (Gjenta resterende kode for alle tester i nåværende testsett)
std::set<vec3> translations;
std::set<unit_axis_angle> rotations;
std::set<vec3> scales;
std::set<processing::noise_type> pi_noise;
std::set<processing::noise_type_3d> spd_noise;

// Hent ut translasjoner, eller bruk en verdi (0, 0, 0) om ingen er gitt
// Hent ut rotasjoner, eller bruk en verdi (0, (0, 0, 1)) om ingen er gitt
// Hent ut skaleringer, eller bruk en verdi (1, 1, 1) om ingen er gitt
// Hent ut posisjonsuavhengig støy, eller bruk en verdi "none" om ingen er gitt
// Hent ut sfærisk posisjonsavhengig støy, eller bruk en verdi
//  ("none", "none", "none") om ingen er gitt

for (const auto& combination: std::views::cartesian_product(
    translations, rotations, scales, pi_noise, spd_noise))
{
    test_specs.back().emplace_back(
        // Gjeldene kombinasjon med noe tilleggsinformasjon
        );
}
```

Kodesnutt 6: Finne transformasjoner fra spesifikasjonslesing

7.2.2 Moduler

Delen for modulinnlasting er den mest selvstendige delen. Denne delen har ansvaret for å laste inn moduler slik testrammeverket kan bruke den. En modulimplementerer minst en registeringsmetode. Modulene er laget og lagret som delte biblioteker for et gitt operativsystem. Dette betyr at på Windows-plattformer vil modulene være .dll filer, på Linux-plattformer vil de være .so filer og på MacOS-plattformer vil de være .dylib filer. Det er ikke testet om testrammeverket funger på MacOS. Lasting av filene skjer med bruk av kodebiblioteket Boost.DLL. Dette biblioteket laster inn og tilgjengeliggjør blant annet variabler og datatyper som er definert i delte biblioteker. For å laste inn en modul brukes koden gitt i Kodesnutt 7. For at en modul skal være gyldig må

klassen deklareres slik den arver fra en forhåndsbestemt baseklasse. Denne baseklassen inneholder funksjonspekere, virtuelle funksjoner, som skal peke til en aktuell implementasjon av en registreringsmetode. Et utdrag av klassen er vist i Kodesnutt 8, hvor det er `run`-funksjonen som skal starte den aktuelle registreringsmetoden. Forventet resultat fra en implementering er enten: en 4x4 transformasjonsmatrise; rotasjon i XYZ-Euler-format i grader, translasjon og en uniform skalering; eller en feilmelding.

```
auto eos::load_module(const std::filesystem::path &t_path) noexcept
    -> std::expected<std::shared_ptr<eos_module>, std::string> {
    try {
        auto module =  std::make_shared<boost::shared_ptr<eos_module>>(
                boost::dll::import_symbol<eos_module>(
                    t_path.string(), "module",
                    boost::dll::load_mode::append_decorations));
        if (auto ptr = std::shared_ptr<eos_module>{module, module->get()}) {
            return ptr;
        }
        return std::unexpected{"Pointer to module was null. There might be a "
          "version difference between the module interface of this program and"
          " the module. Try to recompile at least one of them."};
    }
    // (Etterfulgt av ulike 'catch' blokker som returnerer feilmeldinger)
}
```

Kodesnutt 7: Modullasting

```
class BOOST_SYMBOL_VISIBLE eos_module
{
public:
    // ...
    [[nodiscard]] virtual auto
    name() const -> std::string = 0;

    [[nodiscard]] virtual auto
    run(std::mutex &t_log_mutex,
        const pcl::PointCloud<pcl::PointXYZ>::Ptr &t_world,
        const eos::property_set &t_props,
        const pcl::PointCloud<pcl::PointXYZ>::Ptr &t_target)
    noexcept
    -> std::expected<std::variant<eos::raw_mat_transform, eos::transform>,
                     std::string> = 0;
};
```

Kodesnutt 8: Grensesnitter for en modul

7.2.3 Preprosessering

Preprosessering er et valgfritt steg i en testkjøring. Preprosessering inkluderer både affine transformasjoner og støytillegg. En affin transformasjon kan inkludere translasjon, rotasjon og skalering. Rekkefølgen transformasjonene er tilføyd er med skalering først, etterfulgt av rotasjon og translasjon til slutt. Translasjon og skalering er angitt med verdier for x-, y- og z-akse. Translasjonen skjer i globale koordinater, ikke lokale etter rotasjon. Rotasjonen er angitt med akse-vinkel konvensjonen. Aksen blir internt normalisert, som forenkler hvordan aksen kan skrives inn. Vinkelen er angitt i grader.

Støy kan også legges til. Posisjonsuavhengig støy og sfærisk posisjonsavhengig støy er støttet. Noen plasser i koden er disse referert til som henholdsvis volumetrisk støy og projektiv- eller perspektivstøy. Posisjonsuavhengig støy vil si støy som blir lagt til langs euler-aksene hver for seg. Dette er den typen støy de fleste artikler om registreringsmetoder tester for [55] [58]. Sfærisk projektivt støy har to moduser internt. Den ene andrer kun på dybden til et punkt, mens den andre kan endre på dybden i tillegg til forflyttelse over en sfærisk overflate. Flere typer distribusjoner kan brukes for å forflytte et punkt langs en spesifikk akse. Distribusjoner som er støttet er: uniform fordeling, normalfordeling, gammafordeling, log-normal-fordeling. Det er også lagt til en fordeling som ikke

forflytter punktet.

```
eos::processing::noise noise_x(t_noise_type_x.stddev, t_noise_type_x.shape);
noise_x.seed(t_noise_type_x.seed);
// (Lik for _y og _z)
const auto itr_x = get_types().find(t_noise_type_x.name);
// (Noen tester), (Lik for _y of _z)

for (auto &point: t_cloud.points)
{
    const auto dr = static_cast<float>(itr_z->second.first(noise_z));
    const auto dtheta = static_cast<float>(itr_x->second.first(noise_x));
    const auto dphi = static_cast<float>(itr_y->second.first(noise_y));

    const auto r = std::sqrt(point.x * point.x + point.y * point.y
                             + point.z * point.z);
    const auto theta = std::acos(point.z / r);
    const auto phi = std::atan2(point.y, point.x);

    point.x = (r + dr) * std::sin(theta + dtheta) * std::cos(phi + dphi);
    point.y = (r + dr) * std::sin(theta + dtheta) * std::sin(phi + dphi);
    point.z = (r + dr) * std::cos(theta + dtheta);
}
```

Kodesnutt 9: Kode for å legge til sfærisk posisjonsavhengig støy

```
eos::processing::noise noise(t_noise_type.stddev, t_noise_type.shape);
eos::processing::noise.seed(t_noise_type.seed);
const auto itr = noise::get_types().find(t_noise_type.name);
// (Noen tester)

for (auto &point: t_cloud.points)
{
    point.x += static_cast<float>(itr->second.first(noise));
    point.y += static_cast<float>(itr->second.first(noise));
    point.z += static_cast<float>(itr->second.first(noise));
}
```

Kodesnutt 10: Kode for å legge til posisjonsuavhengig støy

7.2.4 Kjøring

Det ble originalt laget et eget *thread pool* for å kunne kjøre testene i. Denne trådsamlingen ble laget for å kunne tilordne en kostnad til en oppgave. En vanlig variant av en trådsamling er en hvor oppgaver kan ha ulik prioritet [13]. Denne løsningen implementerer ikke dette men heller at en oppgave kan ta plassen til flere oppgaver. Dette er for at ulike registreringsmetoder kan i seg selv benytte flere tråder. For at det ikke skal blir konkurranse om prosessorkraften når flere tester kjøres i parallell trenger rammeverket å vite hvor mye prosessorkraft, antall tråder, en metode kan bruke.

Det viste seg at det ikke holt med bare å kjøre registreringsmetodene i egne tråder. Enkelte metoder under testing produserte enten segmenteringsfeil, feilmeldinger, `assert` i *debug*-modus eller de brukte relativt lang tid. Merk at med segmenteringsfeil menes det her feilen man for blant annet ved å prøve å lese fra en ugyldig minneadresse, ikke feil med segmentering av punktskyene. For å løse disse problemene ble det valgt å la hovedprogrammet starte andre små programmer som igjen starter registreringen. Det nye programmet blir startet og vedlikeholdt av det samme *thread pool*-et som før. Ved å kjøre metodene i egne program vil de ha sine egne minneområdet håndtert av operativsystemet. Det betyr at operativsystemet kan håndtere eventuelle feil ved en metode uten at hele programmet krasjer. Tråden som startet prosessen kan da overvåke vanlig resultat samt feil operativsystemet har merket. Videre kan et forsøk på registrering kanselleres ved å drepe prosessen når en vis tid har passert. Denne tiden er satt til 60 sekunder slik programmet er nå.

Denne funksjonen ble først prøvd å bli laget med bruk av Boost.Process [40]. Det viste seg at biblioteket ikke støttet å vente en vis mengde for den gir et signal. Dette var noe som var ønsket for å kunne begrense kjøretiden. Det ble istedenfor brukt *SPAWN*-funksjonssettet fra POSIX [28] på Linux og Win32 [39] på Windows. Vår implementasjon ved å kommunisere med operativsystemet direkte fungerte vellykket. Eneste som er noe ugunstig er at på Windows tar det 5 sekunder fra en segmenteringsfeil til programmet får varsel om det. Det er trolig lagt inn denne tidsforsinkelsen for at man skal kunne se popup-boksen som varsel om en programfeil. Når hovedprogrammet mottar feilsignalet lukkes både programmet og popup-boksen.

For å sende data til og fra hovedprogrammet og registreringsprogrammene settes det opp delte minneområdet. Hver prosess får sitt eget minneområde, og områdene er kontrollert av hovedprogrammet. For å sette opp og håndtere de delte minneområdene ble Boost.Interprocess [24] brukt.

Funksjonene som legger til støy fungerer på forskjellige måter. De projektive variantene krever en omgjøring til sfæriske koordinater, utføre endringen og omgjøring tilbake til kartesiske koordinater.

Testkjøringen i vårt rammeverk automatiserer prosessen med å anvende forskjellige registreringsmetoder på inngitt punktskydata. Hver test laster først relevante data og konfigurasjoner definert under spesifikasjonslesingen. Deretter kjøres registreringsalgoritmene sekvensielt eller parallelt, avhengig av testkravene. For å maksimere effektiviteten og utnytte tilgjengelig maskinvare, støtter rammeverket kjøring med flere tråder. Dette gjør det mulig å kjøre flere algoritmer eller datasett samtidig, noe som reduserer total kjøretid og gir raskere tilbakemelding på systemets ytelse under forskjellige forhold.

Etter at testene er fullført, trer resultatbehandlingen i kraft for å samle og analysere dataene produsert av testkjøringen. Resultatene presenteres i en strukturert rapport , som inkluderer tabeller og grafiske fremstillinger av algoritmenes ytelse. Rapporten blir automatisk generert for hver

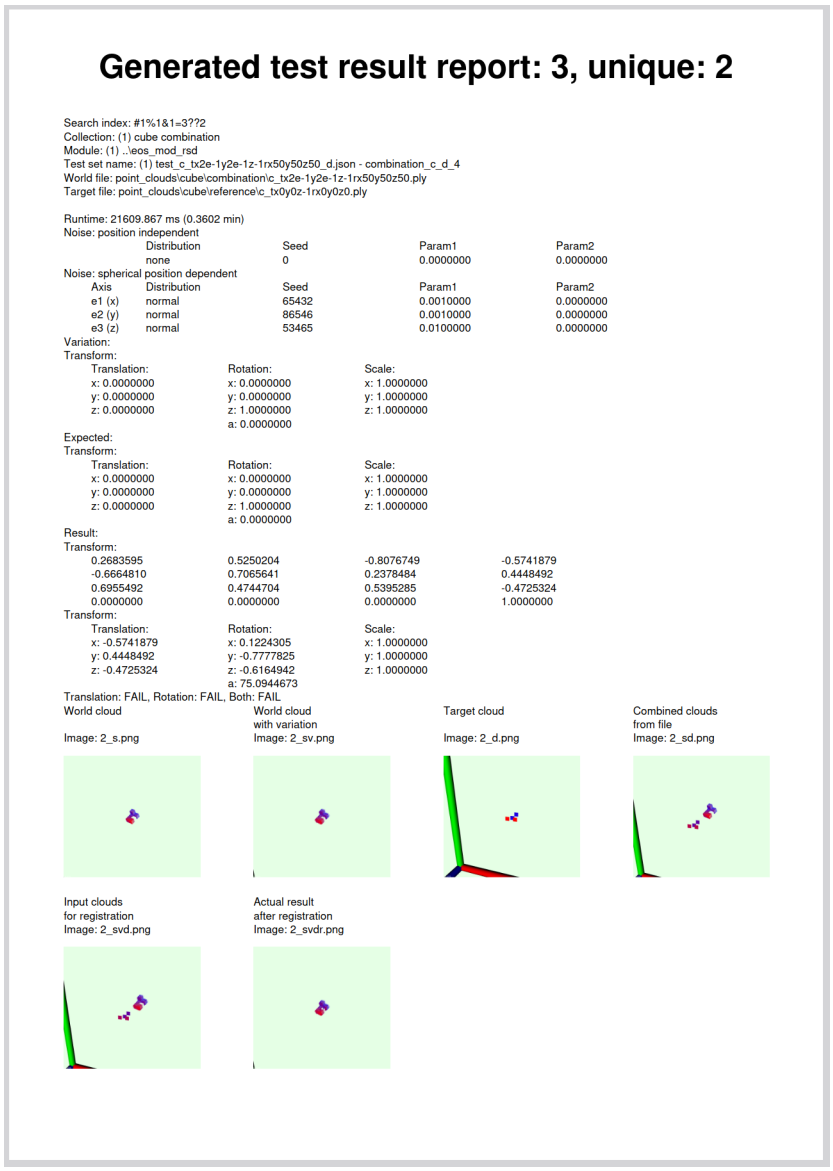
testkjøring.

7.2.5 Rapportgenerering

Etter tester på registreringsmetodene er kjørt blir det generert rapporter. Det genereres flere rapporter som inneholder enten alt eller kun deler av resultatene. En første inndeling er mellom fulle rapporter og sammendragsrapporter. Forskjellen er at fulle rapporter har med sider for hver enkelt test utført, mens sammendragsrapportene unnlater disse sidene. Disse rapport-parene blir laget på flere nivåer. Det øverste nivået er rapportene som inneholder alt, videre er det egne rapporter per kolleksjon, modul, testsett og test. På nivået for en enkelt test lages det ikke en sammendragsrapport. Et eksempel på en rapportside for en individuell test er vist i Figur 26. Denne siden inneholder følgende:

- Informasjon om hvilken kolleksjon, modul og testsett som kjøres, sammen med en unik id for testen.
- Filnavnene til punktskyene som brukes i testen.
- Tiden det tok for å kjøre testen, i millisekund og i minutt.
- Variasjonstransformasjonen som er tilføyd før registrering.
- Forventet transformasjon.
- Faktisk transformasjon fra å kjøre registrering med de angitte punktskyene og variasjonstransformasjonen. Dersom registreringsmetoden produserte en 4x4 matrise vises den i tillegg til translasjon, rotasjon som akse-vinkel og skalering som vises så lenge metoden ikke feilet.
- Status av sammenligning mot gitte toleranser. Toleransen som er satt er 1 cm i hver akse i translasjon, og 10 grader i vinkel med 0.1 i forskjell i en akse for rotasjon.
- Eventuell feilmelding
- Bilder av punktskyene som lastet fra fil, sammen i samme bilde, den statiske punktskyen etter variasjonstransformasjonen, sistnevnte sammensatt med punktskyen som skal flytte seg og eventuelt sammensatt resultat etter registrering.

Sidene for et testsett inneholder antall tester, og om de var vellykkede, hadde feil resultat eller feilet. Videre er det vist statistikk for kjøretiden for hver av inndelingene. Tester som feilet er delt opp i myke og harde feil i denne listen. Myke feil er feil metoden selv rapporterte om, men harde feil er feil den ikke merket. Eksempel på harde feil er tidsavbrudd, segmenteringsfeil, signal og feilmeldinger som ikke ble fanget. Tidsavbrudd er tatt med i telleren for harde feil, mer er ikke tatt



Generated test result report: 3, unique: 2

Search index: #1%1&1=3??2
Collection: (1) cube combination
Module: (1) ..\eos_mod_rsd
Test set name: (1) test_c_tx2e-1y2e-1z-1rx50y50z50_d.json - combination_c_d_4
World file: point_clouds\cube\combination\c_tx2e-1y2e-1z-1rx50y50z50.ply
Target file: point_clouds\cube\reference\c_tx0y0z-1rx0y0z0.ply

Runtime: 21609.867 ms (0.3602 min)
Noise: position independent

Distribution	Seed	Param1	Param2
none	0	0.0000000	0.0000000

Noise: spherical position dependent

Axis	Distribution	Seed	Param1	Param2
e1 (x)	normal	65432	0.0010000	0.0000000
e2 (y)	normal	86546	0.0010000	0.0000000
e3 (z)	normal	53465	0.0100000	0.0000000

Variation:
Transform:

Translation:	Rotation:	Scale:
x: 0.0000000	x: 0.0000000	x: 1.0000000
y: 0.0000000	y: 0.0000000	y: 1.0000000
z: 0.0000000	z: 1.0000000	z: 1.0000000
	a: 0.0000000	

Expected:
Transform:

Translation:	Rotation:	Scale:
x: 0.0000000	x: 0.0000000	x: 1.0000000
y: 0.0000000	y: 0.0000000	y: 1.0000000
z: 0.0000000	z: 1.0000000	z: 1.0000000
	a: 0.0000000	

Result:
Transform:

0.2683595	0.5250204	-0.8076749	-0.5741879
-0.6664810	0.7065641	0.2378484	0.4448492
0.6955492	0.4744704	0.5395285	-0.4725324
0.0000000	0.0000000	0.0000000	1.0000000

Transform:

Translation:	Rotation:	Scale:
x: -0.5741879	x: 0.1224305	x: 1.0000000
y: 0.4448492	y: -0.7777825	y: 1.0000000
z: -0.4725324	z: -0.6164942	z: 1.0000000
	a: 75.0944673	

Translation: FAIL, Rotation: FAIL, Both: FAIL

World cloud
Image: 2_s.png

World cloud with variation
Image: 2_sv.png

Target cloud
Image: 2_d.png

Combined clouds from file
Image: 2_sd.png

Input clouds for registration
Image: 2_svd.png

Actual result after registration
Image: 2_svdr.png

Figur 26: Eksempel på en rapportside for en individuell test.

med i tidsstatistikken. Tidsstatistikken viser flere verdier både i millisekund og i minutt. Verdiene er korteste kjøretid, lengste kjøretid, gjennomsnittlig kjøretid og medianen av kjøretiden. Total kjøretid per inndeling vises også. Videre listes det opp antall feilmeldinger av hver type sammen med teksten til feilmeldingen.

På forsiden, per kolleksjon og per modul vises lignende informasjon som for et testsett. Informasjonen som mangler er korteste, lengste, gjennomsnittlige kjøretiden sammen med median-verdidene per inndeling. De andre verdiene er addisjonen av de samme tallene til undersidene. For eksempel er tallet for totalt antall tester i en kolleksjon lik summen av det samme feltet for modulene som er i kolleksjonen.

7.3 Selvtesting

Selvtestingrammeverket er laget for å teste de ulike funksjonene brukt i hovedprogrammet. Det ble i hovedsak laget for å teste og tilføyingen av ulike støytyper fungerte som det skulle. Det ble valgt å lage et eget minimalt testsystem fremfor å bruke noe eksisterende. Eksempel på eksisterende testsystem er Catch2, Boost.Test og GTest. Grunnen til at disse ikke ble valgt var at vi ønsket at testene skulle ta inn og lagre punktskyer for analyse i eksterne program som Blender. Ved å bruke et eksisterene system måtte testene ha mye felles for å håndtere innlasting, sjekking, feilhåndtering og lagring til fil. Et eksempel på en test for å legge til støy med ugyldige parametere er vist i Kodesnutt 11. Tester som er vellykket lagrer punktskyen til en fil.

```
eos::testing::test test{argv[0]};


test.expect_fail("param vol uniform 0 0.0 -1.0",
    "Expected error for invalid parameters for a uniform distribution",
    "noise/volume_cube_10x10x10.ply", [](eos::io::point_cloud &t_cloud) {
        return  eos::processing::add_volume_noise(
         t_cloud, {std::rand(), "uniform",
         0.0, -1.0});
});


// Flere test.expect_fail(...) eller test.expect_pass(...)
```

Kodesnutt 11: Eksempel på en selvtest

I selv-testingrammeverket kan man forvente både positive og negative resultat fra det som testes. Resultatet fra selv-testingrammeverket er en liste over sanne-positive, sanne-negative, falske-positive og falske-negative resultater. Denne forvirringstabellen skrives til konsollen og til fil på en lignende måte som annen loggføring beskrevet tidligere. Eventuelle feil og falske resultater er det som nå loggføres i filen for feilmeldinger. Selv-testingrammeverket er brukt til å teste at de ulike rotasjons-operasjonene fungerer som forventet.

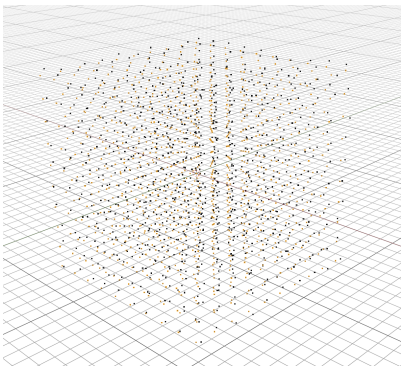
Selv-testingrammeverket er også utformet med tanke på operasjoner med punktskyer. En test tar av seg automatisk lesing av en punktsky-fil og lagrer vellykkede operasjoner på en punktsky som en egen punktsky-fil. Andre tester kan videre bruke resultatet fra en tidligere kjørt test. Dette gjøres ved å starte filnavnet med *$*. Denne prefiksen sier at programmet skal lete etter filen i mappen for hvor den lagrer resultatet fra tidligere kjørte tester. Denne delen av selv-testingrammeverket brukes for å teste at lesing og lagring av punktsky-filer fungerer som forventet. Videre testes feilhåndtering

av ugyldige parametere for å legge til de ulike typer støy. For å teste at støy legges til på antatt måte kan resultatfilene analyseres i et ekstern program som Blender. Et eksempel på en selvtest er vist i Kodesnutt 12.

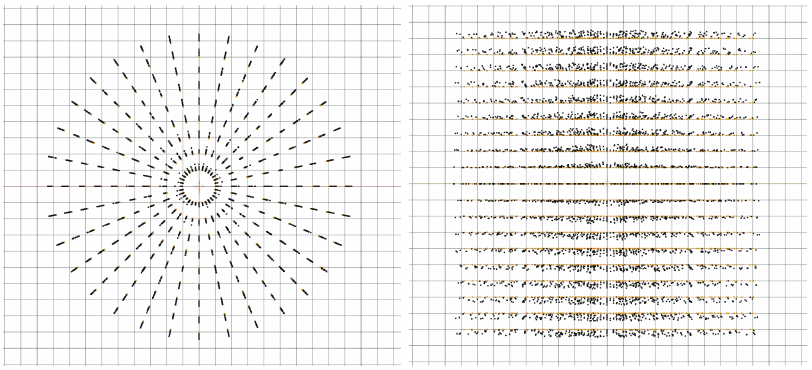
```
eos::testing::test test{argv[0]};

test.expect_fail("param vol failes on nonexistent distribution names",
    "Expected failure saying 'nonexistent_' followed by a number is not a"
    " valid distribution",
    "noise/volume_cube_10x10x10.ply",
    [](eos::io::point_cloud &t_cloud) {
        return eos::processing::add_volume_noise(t_cloud,
                {0, std::format("nonexistent_{}", std::rand()), -0.1, 0.1}
            );
    });
```

Kodesnutt 12: Eksempel på en selvtest med en punktsky

Punktskyene som er brukt for å teste tilføyingen av støy er vist i Figur 21. I Figur 27 vises en punktsky med tilføyd støy i svart, mens de originale punktskyene er vist i farge. I Figur 28 vises to punktskyer fra forskjellige perspektiv. I den svarte punktskyen er det lagt til sfærisk posisjonsavhengig støy i dybden med en uniformfordeling som kun forskyver punktene bort fra origo. Disse testene sammen med mange flere er med på å validere at støy-funksjonene er rett implementert.



Figur 27: Illustrasjon av punktskyen vist i Figur 21a med tilføyd uniformt posisjonsuavhengig-støy



(a) Visning fra toppen (b) Visning fra siden

Figur 28: Illustrasjon av punktskyen vist i Figur 21b med tilføyd posisjonsavhengig-støy i dybden

Selv-testrammeverket ble også utvidet for å teste matematiske operasjoner. Dette ble gjort for å sikre at alle de matematiske operasjonene i testrammeverket fungerer som de skal. Det ble utført flere tester. Disse testene fokuserte på å verifisere at konverteringen og sammenligningen mellom ulike typer representasjoner for transformasjoner var rett. Dette innebærer rotasjon, translasjon og skalering, men også spesialtilfeller med blant annet rotasjon uten en akse. Testen er utført med å bruke kjente referansedata og forventede transformasjoner. Resultatene viser at de beregnede transformasjonene samsvarte godt med de forventede verdiene. Noen av de ulike metodene for å regne ut det samme ga nære men ikke identiske svar. Dette er trolig på grunn av at feil i flyttalls-matematikken bygger seg opp. Et eksempel på et slikt tilfelle er en test hvor det var en forventet translasjon på $\Delta x = 0.1, \Delta y = 0.1, \Delta z = 0.1$, men viste et resultat på $\Delta x = 0.0999, \Delta y = 0.1001, \Delta z = 0.0998$. Tester på rotasjon ga resultater med lignende feilmargin.

7.4 Diskusjon

Den største funksjonen som ikke er implementert som ønsket er rapportgenereringen. Nåværende skjer testkjøringen, statistikkutregningen og rapportgenereringen serielt etter hverandre. Dette er ikke et problem i seg selv, men dersom det totalt er mange tester som kjøres i en omgang kan det oppstå problemer. Problemet er at PDF-er må normalt komprimeres. Uten komprimering påskrudd bruker en PDF på 200 sider og totalt 1100 bilder med en oppløsning på 1024x1024 piksler en størrelse på 3,5 GB. Med å redusere bildeoppløsningen til 128x128 bruker PDF-en med ellers samme innhold 1,5 GB. Med denne oppløsningen er det normalt ikke mulig å vurdere resultatet i det. Med å skru på komprimering, blir derimot PDF-en med bilder og en oppløsning på 1024x1024 piksler bare 18 MB. Dette er en betydelig reduksjon i filstørrelsen, men det tar også betraktelig mer tid å lagre filen på grunn av komprimeringen som må skje. Det kreves også mer arbeidsminne for å behandle større filer. Med rundt 1000 tester kreves det omtrent 11 GB med arbeidminne. Utgangspunktet før genereringen av rapporter er på rundt 20 MB for applikasjonen. De fleste andre applikasjoner var da lukket, i tillegg var `Sanntidsbeskyttelse` i Windows midlertidig deaktivert

mens programmet kjørte. Dette betyr at det ikke anbefales å kjøre programmet på en datamaskin med mindre enn 16 GB arbeidsminne.

Grunnen til minneproblemene er at *libharu* arbeider med filer bare i arbeidsminnet. Dette er trolig blant annet på grunn av funksjonen den har til å sette inn sider hvor det er ønskelig, og ikke bare på slutten av dokumentet. Dette gjør at det enten ikke er mulig eller upraktisk fra deres side å skrive side for side til disk. Det kan også være mulig at PDF-er ikke kan lagres side for side uten på grunn av plassering av meta-data eller av andre grunner. Mulighetene og begrensningene til PDF har vi ikke utforsket videre.

Et alternativ som kan brukes dersom det er ønskelig å kjøre store testsett om gangen er å gjøre noen endringer slik hver side av den ønskede PDF-en skrives til fil hver for seg, og til slutt enten i programmet i seg selv, eller i et eksternt program kan disse enkeltsidene settes sammen til én stor PDF. En annen løsning kan være å ikke skrive til PDF direkte. Testrammeverket kan i steden for produsere HTML filer med det samme innholdet og oppsettet. HTML-filen kan videre gjøres om til en PDF via en nettleser [1] [41] dersom det er ønskelig å ha resultatet i spesifikt PDF-form og ikke bare en enkel lesbar form. På denne måten kan programmet skrive direkte til en fil uten å måtte ha den i arbeidsminne for å komprimere den.

Den største endringen vi hadde gjort om skulle starte på nytt har med bruk av egne matematiske funksjoner og klasser. I starten av prosjektet var det bare et behov for et få antall matematiske funksjoner og klasser. I hovedsak var det lagring av vinkler i vinkel-akse-form. I steden for å bruke et stort ekstern bibliotek som Boost.QVM eller Eigen ble de heller implementert selv. Videre i utviklingen ble det et større behov for andre matematiske funksjoner men disse ble i hovedsak laget selv. Unntaket var å tilføye variasjonstransformasjonen til en punktsky. Dette ble gjort med bruk av Eigen. Senere ble også Eigen brukt for å regne ut den forventede transformasjonen etter en vellykket registrering. Det kan være hensiktsmessig å bytte ut våre egne matematiske funksjoner med Eigen sine tilsvarende funksjoner, siden biblioteket nå uansett er benyttet. Ved å bruke et eksisterende bibliotek fra starten av hadde vi heller ikke trengt å lage enhetstester for våre matematiske funksjoner. Fordelen med at vi startet selvlagde funksjoner er at vi har fått en dypere forståelse for transformasjonsmatriser, rotasjonsmatriser, akse-vinkel, kvaternioner og versorer (enhets-kvaternioner), samt sammenhengen mellom de.

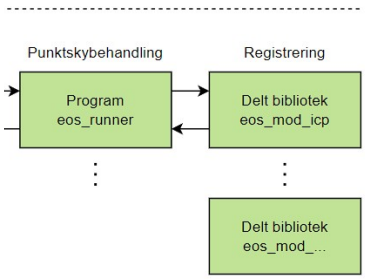
Det ble valgt å bruke delte biblioteker for å representere modulene av flere grunner. En første grunn er at det tillater å legge til nye registreringsmetoder uten å måtte rekompilere hele testrammeverket. På denne måten vil det ta mindre tid å legge inn nye metoder. En annen grunn til hvorfor det ble valgt var på grunn av det var tenkt delte biblioteker tilbydde isolering mot en rekke minnefeil. Dette viste seg til å ikke stemme. Et eksempel på dette er at implementasjonen av Zhao-metoden ikke var skrevet på en trygg måte for vilkårlige hyperparameter. Med mange kombinasjoner av

hyperparameter, spesielt de som angår blokkstørrelsene for søking, vil ulike feilmelinger oppstå. I hovedsak var det segmenteringsfeil (SIGSEGV) som oppsto. Når slike feil oppsto krasjet hele testrammeverket. Dette er ikke ønskelig, spesielt om flere tester kjøres samtidig, og ikke bare en test for seg selv. Løsningen ble da å utvide rammeverket slik modulene ble lastet inn av egne prosesser.

Det er et kjent tilfelle hvor testrammeverket vil krasje. Programmet vil krasje dersom det blir sagt at en modul vil trenge 0 tråder å kjøre, eller et negativt tall. Programmet vil da prøve å starte den aktuelle modulen med en test og krasje. Grunnet til dette er at den prosessen fikk ikke tildelt en eller flere id-er, siden ved å spør om mindre enn én tråd vil gi den ingen. Dette kunne ha blitt håndtert når konfigurasjonsfilene blir lest inn, og invalidert konfigurasjonen dersom verdien ikke er over elle lik 1.

8 Registreringsmetoder

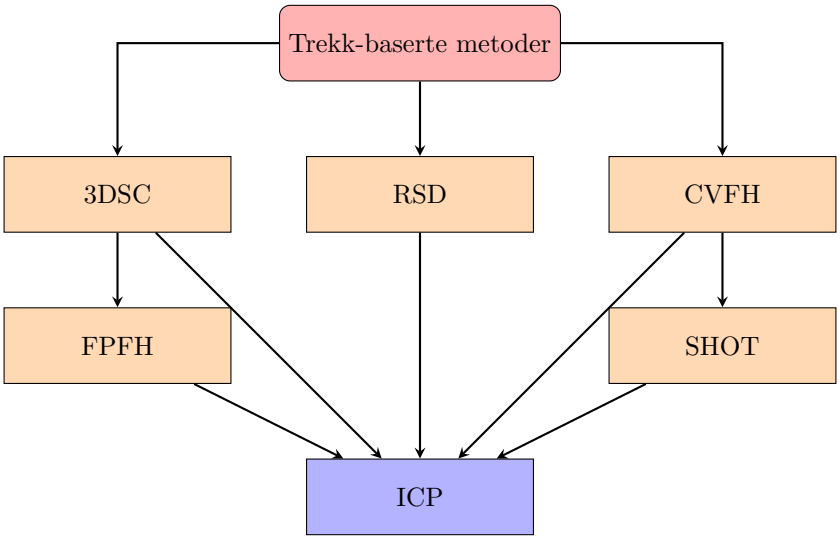
Denne seksjonen beskriver hvordan regitreringsmetodene er implementert. Figur 29 viser delen av programmet dette omfanger.



Figur 29: Illustrasjon av dataflyten i testrammeverket – registrering

8.1 Algoritmer

Diagrammet under illustrerer de trekk-baserte algoritmene som kjøres parallelt i hver sin modul. Hver av de navngitte metodene kjører ICP etter deteksjon av distinktive trekk for ett mer nøyaktig resultat:



Valget om å implementere de trekk-baserte metodene i parallell før bruken av ICP, er på grunn av ønsket om å optimalisere effektiviteten og nøyaktigheten av registreringen. Ved å kjøre hver metode som RSD, 3DSC, CVFH, FPFH og SHOT i separate moduler, oppnås en parallell prosessering som kan redusere total beregningstid. Dette gjør det mulig å oppnå en fleksibel og skalerbar systemarkitektur der modulene enkelt kan legges til eller oppdateres.

De trekk-baserte metodene er først og fremst anvendt for å identifisere distinktive trekk i punktskyene, som danner grove initialkorrespondanser før finjustering utført av ICP. Denne typen tilnærming reduserer mengden med data som ICP må behandle, og kan dermed fokusere på områder med etablerte korrespondanser. Dette vil da føre til raskere og mer nøyaktig registrering. ICP minimerer da avstanden mellom korrespondernde punkt og korrigerer små avvik.

8.2 Implementering

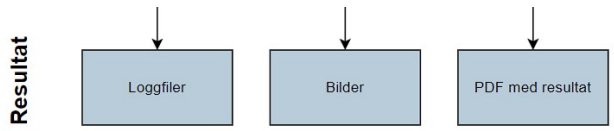
De fleste metodene baserer seg på implementasjonen tilgjengelig i PCL. Dette gjelder ICP, RSD, 3DSC, FPFH, CVFH og SHOT. Det ble valgt å bruke PCL sin implementasjon siden PCL er et av de største punktsky-bibliotekene, og at deres implementasjon trolig er gode.

Metoden vi har testet som ikke brukte PCL som base er den av Yuan Zhao mfl. [61], heretter referert til som *Zhao*. Denne metoden ble valgt for å kunne se hvor mye arbeid som måtte til for å bruke en tilfeldig valgt ny metode. Zhao-metoden har en implementasjon på GitHub med en MIT lisens. Den måtte delevis skrives om til å kunne brukes på en automatisk måte over flere plattformer. Metoden er implementert i C, og den håndterer ikke minnet den allokerer. Metoden kunne også bare lese PCD-filer i ASCII-format samt CSV-filer. Det første som ble gjort var å endre koden slik den aksepterte `pcl::PointCloud<pcl::PointXYZ>` for punktskyer. Videre ble ulike *stack*-mønster og lister byttet ut med tilsvarende C++ datastrukturer som hånderer minnet. Dette ble implementert før det ble bestemt at metodene skulle kjøre i egne programmer. Med den nåværende løsningen hadde det ikke vært nødvendig å gjøre metoden minnetrygg, siden operativsystemet vil håndtere det så fort en registrering er ferdig. Noe som gjenstår for å kunne benytte Zhao-metoden er å tolke resultatet fra den rett. Rotasjon og translasjon tydes ikke på rett måte, og visualiseringen av resultatet fra denne metoden er dermed feil. Flere ulike rekkefølger og metoder for å kombinere rotasjon og translasjon på ble testet, men ingen med et mer forståelig resultat enn andre. Metoden har derimot en innebygd test for å se om den selv tror registreringen var vellykket. Feilmeldinger om at den ikke var vellykket blir videresendt i steden for et resultat dersom en den merker en feil.

Metoden foreslott av Zhao fungerer på en spesiell måte. Den takler ikke at punktskyene er rotert i det virtuelle rommet. Det vil si at den forventer at midten av kamerabildet er i Z- aksen, altså i $[0, 0, -|z|]$. Dette gjør at metoden kun greier å detektere forskjellen i en pyramideform, med spissen av pyramiden, kameraposisjonen og origo i samme punkt. Dette kan også tydes fra at metoden krever å fylle inn blant annet horisontal og vertikal oppløsning sammen med andre kameraegenskaper. Denne oppløsningen skal være lik kameraoppløsningen eller mindre. Ved en mindre oppløsning blir ikke alle datapunktene brukt. Det er det punktet som er nærmest kameraet som blir bevart av to nærliggende punkter.

9 Resultat

Denne seksjonen beskriver resultatet fra å evaluere registreringsmetodene. Resultatene er basert på delene vist i Figur 30.



Figur 30: Illustrasjon av dataflyten i testrammeverket – resultat

9.1 ICP

Metoden ICP har vist seg å både være hurtig og god til formålet vårt som var finjustering. Den brukes også som referanse til de videre resultatene.

9.2 Zhao

Det er vanskelig å evaluere resultatene ettersom vi ikke fant ut hvordan rotasjonen og translasjonen den gir skal tolkes. Den er derimot rask på medium store punktskyer (1000 - 30000 punkt), men den feiler med små punktskyer.

9.3 RSD

Under testingen av RSD-metoden ble det utført flere tester for å evaluere nøyaktigheten av translasjonsestimatene i forskjellige retninger og størrelser. Hver test viste varierende grad av nøyaktighet, og ga viktig innsikt i metodens styrker og svakheter. Testingen ble utført på en kunstig generert punktsky som ble designet i programvaren blender.

Test 1, Mindre translasjoner: Den første testen involverte en forventet translasjon på $x = 0.1, y = 0.1, z = 0.1$. Resultatet viste en translasjon på $x = -0.08161895, y = 0.115491994, z = 0.10000012$, som er ganske nær det forventede for z-koordinaten, men med små avvik i x- og y-koordinater. Dette antyder en liten mismatch i forventet versus faktisk translasjonsretning, men viser at metoden er i stand til å håndtere mindre translasjoner med relativt god nøyaktighet.

Test 2, Moderat translasjoner: I den andre testen, hvor forventet translasjon var $x = 0.2, y = 0.2, z = 0.2$, var resultatene $x = -0.16323738, y = 0.23098399, z = -0.20000035$. Disse resultatene indikerer betydelige avvik fra forventet translasjon, spesielt for x- og z-koordinatene, noe som tyder på utfordringer med metodens håndtering av større translasjoner.

Test 3 & 4, Inverterte translasjoner: Testene 3 og 4 viste også at translasjonene for x og y ofte var omvendte fra det som var forventet, og ikke helt nøyaktige for z. For eksempel i test 3 med en forventet translasjon på $x = -0.2, y = -0.2, z = -0.2$, viste resultatene $x = 0.19806822, y = 0.22652619, z = -0.1716261$. Dette mønsteret av inkonsistens i å fange opp riktig retning på translasjonene gjentok seg gjennom flere tester.

Test 5, Null translasjon: I den femte testen, hvor det ikke var noen forventet translasjon, viste resultatene seg å være ekstremt nær null for x- og y-aksene, men med en z-translasjon på $z = 0.06777443$, noe som indikerer en feil i håndteringen av null-translasjon.

Generelle observasjoner Fra testene observerte vi at RSD-metoden konsekvent viser problemer med retningen på translasjonene. Dette kan skyldes metodens måte å tolke overflatekrumninger på, som ikke nødvendigvis korrelerer med nøyaktige posisjonsestimater. Det var også en tendens til unøyaktigheter i størrelsen på de forventede translasjonene, spesielt for større verdier.

9.4 3DSC

1. **Test 6:**
    - **Forventet translasjon:** $x = 0, y = 0, z = 0$
    - **Resultat translasjon:** $x = 0, y = 0, z = 0$
    - **Analyse:** Resultatet er perfekt i tråd med de forventede verdiene, noe som indikerer nøyaktig deteksjon der det ikke er noen forskyvning.
2. **Test 7:**
    - **Forventet translasjon:** $x = 0.1, y = 0.1, z = 0.1$
    - **Resultat translasjon:** $x = -0.030189019, y = -0.03202916, z = -0.034124054$
    - **Analyse:** Resultatene viser en misvisning i deteksjonen av translasjon, med en feil i retning og størrelse som indikerer en potensiell kalibreringsfeil.
3. **Test 8:**

- **Forventet translasjon:** $x = 0.2, y = 0.2, z = 0.2$
- **Resultat translasjon:** $x = -0.12964714, y = -0.13199848, z = -0.13469318$
- **Analyse:** Som i forrige test, resultatene er feilaktige og viser store avvik fra forventet translasjon, noe som tyder på utfordringer med metoden under større translasjoner.

4. **Test 9:**
   - **Forventet translasjon:** $x = -0.1, y = -0.1, z = -0.1$
   - **Resultat translasjon:** $x = 0.032029297, y = 0.034124073, z = 0.030189073$
   - **Analyse:** Translasjonene er igjen feilaktige og motsatte av de forventede verdiene, noe som bekrefter en konsistent utfordring med metoden.
5. **Test 10:**
   - **Forventet translasjon:** $x = -0.2, y = -0.2, z = -0.2$
   - **Resultat translasjon:** $x = 0.13199894, y = 0.12964731, z = 0.1346936$
   - **Analyse:** Resultatet er motsatt av det som var forventet, og viser at metoden har vanskeligheter med å nøyaktig detektere og tolke translasjonsretninger og størrelser.

Generelle observasjoner

- **Deteksjon av translasjon:** 3DSC-metoden har vist seg å være uforutsigbar i å oppdage korrekt translasjon i disse testene, noe som antyder at metodens følsomhet og parameterinnstillinger må revurderes.
- **Bruksområder:** Selv om 3DSC kan være effektiv i tettere punktskyer for detaljert form- og overflateanalyse, viser disse testene at den kanskje ikke er ideell for nøyaktig posisjonsdeteksjon, spesielt når nøyaktig lokalisering er kritisk.

9.5 CVFH

1. **Test 11:**
   - **Forventet translasjon:** $x = 0, y = 0, z = 0$
   - **Resultat translasjon:** $x = 0, y = 0, z = 0$
   - **Analyse:** Resultatene viser ingen translasjon, som samsvarer med de forventede verdiene. Dette indikerer at CVFH-metoden ikke identifiserte noen translasjoner i testen.
2. **Test 12:**
   - **Forventet translasjon:** $x = -0.1, y = -0.1, z = -0.1$

- **Resultat translasjon:** $x = 0.032029297, y = 0.034124073, z = 0.030189073$
- **Analyse:** Den faktiske translasjonen er ikke i tråd med de forventede verdiene. Det er små avvik i translasjonsverdiene.

3. **Test 13:**
   - **Forventet translasjon:** $x = 0.2, y = 0.2, z = 0.2$
   - **Resultat translasjon:** $x = -0.12964714, y = -0.13199848, z = -0.13469318$
   - **Analyse:** Det er betydelige avvik i translasjonsverdiene fra de forventede verdiene. CVFH-metoden viser utfordringer med å nøyaktig identifisere større translasjoner.
4. **Test 14:**
   - **Forventet translasjon:** $x = 0.1, y = 0.1, z = 0.1$
   - **Resultat translasjon:** $x = -0.030189019, y = -0.03202916, z = -0.034124054$
   - **Analyse:** Translasjonsverdiene viser avvik fra de forventede verdiene, men det er noen likheter. Metoden viser imidlertid fortsatt begrensninger i nøyaktig identifisering av translasjon.
5. **Test 15:**
   - **Forventet translasjon:** $x = -0.2, y = -0.2, z = -0.2$
   - **Resultat translasjon:** $x = 0.13199894, y = 0.12964731, z = 0.1346936$
   - **Analyse:** Resultatene viser betydelige avvik fra de forventede translasjonsverdiene, og indikerer begrensninger i CVFH-metodens evne til å identifisere store translasjoner nøyaktig.

9.5.1 Generelle observasjoner

- **Nøyaktighet i translasjon:** CVFH-metoden viser begrensninger i nøyaktigheten til å identifisere translasjoner, spesielt for større verdier.
- **Manglende samsvar med forventningene:** Det er en generell manglende samsvar mellom de forventede og faktiske translasjonsverdiene, noe som indikerer utfordringer med metoden.

9.6 FPFH

1. **Test 16:**
   - **Forventet translasjon:** $x = 0, y = 0, z = 0$
   - **Resultat translasjon:** $x = 0, y = 0, z = 0$

- **Analyse:** Resultatet er perfekt i tråd med de forventede verdiene, noe som indikerer nøyaktig deteksjon uten noen forskyvning.

2. **Test 17:**
   - **Forventet translasjon:** $x = -0.1, y = -0.1, z = -0.1$
   - **Resultat translasjon:** $x = 0.032029297, y = 0.034124073, z = 0.030189073$
   - **Analyse:** Resultatene viser ett lite avvik i translasjon, men det er generelt nær de forventede verdiene, noe som indikerer tilfredsstillende nøyaktighet i deteksjonen.
3. **Test 18:**
   - **Forventet translasjon:** $x = -0.2, y = -0.2, z = -0.2$
   - **Resultat translasjon:** $x = 0.13199894, y = 0.12964731, z = 0.1346936$
   - **Analyse:** Det er en liten feil i translasjon, men retningen og størrelsen er generelt i nærheten av de forventede verdiene, noe som indikerer akseptabel nøyaktighet.
4. **Test 19:**
   - **Forventet translasjon:** $x = 0.2, y = 0.2, z = 0.2$
   - **Resultat translasjon:** $x = -0.12964714, y = -0.13199848, z = -0.13469318$
   - **Analyse:** Som i tidligere tester, er det en liten feil i translasjon, men metoden har generelt oppdaget retningen og størrelsen på translasjonen, selv om det er avvik.
5. **Test 20:**
   - **Forventet translasjon:** $x = 0.1, y = 0.1, z = 0.1$
   - **Resultat translasjon:** $x = -0.030189019, y = -0.03202916, z = -0.034124054$
   - **Analyse:** Selv om det er en liten feil i translasjon, er resultatene relativt nær de forventede verdiene, og metoden har oppdaget retningen og størrelsen på translasjonen.

9.6.1 Generelle observasjoner

- **Nøyaktighet i translasjon:** FPFH-metoden viser seg å være relativt nøyaktig i deteksjonen av translasjoner, selv om det er mindre avvik i noen tilfeller.
- **Konsistens:** Metoden viser konsistens i å oppdage retningen og størrelsen på translasjonen, selv om det er små unøyaktigheter.

9.7 SHOT

1. **Test 21:**

- **Forventet translasjon:** $x = 0, y = 0, z = 0$
- **Resultat translasjon:** $x = 0, y = 0, z = 0$
- **Analyse:** Resultatet er nøyaktig i tråd med de forventede verdiene, som indikerer en perfekt deteksjon uten noen forskyvning.

2. **Test 22:**
   - **Forventet translasjon:** $x = -0.1, y = -0.1, z = -0.1$
   - **Resultat translasjon:** $x = -0.030189019, y = -0.03202916, z = -0.034124054$
   - **Analyse:** Resultatene viser ett lite avvik i translasjon, men det er generelt nær de forventede verdiene, noe som indikerer tilfredsstillende nøyaktighet i deteksjonen.
3. **Test 23:**
   - **Forventet translasjon:** $x = -0.2, y = -0.2, z = -0.2$
   - **Resultat translasjon:** $x = -0.12964714, y = -0.13199848, z = -0.13469318$
   - **Analyse:** Det er en liten feil i translasjon, men retningen og størrelsen er generelt i nærheten av de forventede verdiene, noe som indikerer akseptabel nøyaktighet.
4. **Test 24:**
   - **Forventet translasjon:** $x = 0.2, y = 0.2, z = 0.2$
   - **Resultat translasjon:** $x = 0.13199894, y = 0.12964731, z = 0.1346936$
   - **Analyse:** Som i tidligere tester, det er en liten feil i translasjon, men metoden har generelt oppdaget retningen og størrelsen på translasjonen, selv om det er avvik.
5. **Test 25:**
   - **Forventet translasjon:** $x = 0.1, y = 0.1, z = 0.1$
   - **Resultat translasjon:** $x = 0.032029297, y = 0.034124073, z = 0.030189073$
   - **Analyse:** Selv om det er en liten feil i translasjon, er resultatene relativt nær de forventede verdiene, og metoden har oppdaget retningen og størrelsen på translasjonen.

9.7.1 Generelle observasjoner

- **Nøyaktighet i translasjon:** SHOT-metoden viser seg å være relativt nøyaktig i deteksjonen av translasjoner, selv om det er mindre avvik i noen tilfeller.
- **Konsistens:** Metoden viser konsistens i å oppdage retningen og størrelsen på translasjonen, selv om det er noen unøyaktigheter.

10 Diskusjon

10.1 Manglende og uferdige funksjoner

Det er flere manglende og uferdige funksjoner. Den viktigste av disse er å regne ut forventet transformasjon med variasjonstransformasjon. Dette viste seg til å være vanskeligere å gjøre en først antatt. Det burde også være mulig å bestemme toleranser for sammenligning uten å måtte rekompilere programmet. Toleransene kan da enten være angitt i konfigurasjonsfilene eller i testsett-filene. Videre burde det være mulig å bestemme antall tråder uten å måtte rekompilere programmet. Dette kan håndteres som et argument til programmet. Nåværende kjører koden med 6 tråder. Til slutt kan det være nyttig å visualisere den forventede løsningen i de autogenererte-rapportene. Dette kan gjøre det enklere å sammenligne forventet og faktisk resultat, siden de har både en numerisk og visuell fremstilling.

10.2 Alternativ strategi

Testene må nå genereres eller manulet skrives i en rekke filer. En annen måte å ha definert testene på er å bruke en evolusjonær algoritme. På denne måten kan algoritmen prøve finne kantene til det flerdimensjonale volumet hvor en registreringsmetode fungerer med et gitt sett med hyperparametere. Dimensjonene blir 3 for translasjon, 3 for rotasjon og eventuelt 3 for skalering. Flere typer algoritmer kan være aktuelle, blant annet partikelsvermoptimalisering [31], hvor målet er å maksimere translasjonen. Hvordan rotasjon, skalering og støy skal behandles må vurderes. Et problem som kan oppstå er at det blir vanskelig å finne kantene dersom løsningen er et konkav sett av verdier [14]. Løsningen kan også være flere sett som ikke overlapper hverandre.

10.3 Lisens

Vi har valgt å lisensiere vårt testrammeverk og de tilhørende algoritmene under en åpen kildekode-lisens, nemlig MIT-lisensen. Denne lisensen er kjent for å være både fleksibel og brukervennlig, noe som betyr at du fritt kan bruke, kopiere, endre og distribuere programvaren vår, så lenge den opprinnelige opphavsrettsinformasjonen og lisensmerkingen beholdes.

Vi valgte MIT-lisensen for å fremme samarbeid og videre utvikling i forskningsmiljøet. Ved å gjøre koden vår åpen, håper vi å oppmuntre andre til å bidra til prosjektet, enten ved å forbedre eksisterende algoritmer, legge til nye funksjoner, eller bruke rammeverket i sine egne prosjekter. Dette samsvarer med vår visjon om å støtte åpen vitenskap og teknologisk utvikling.

En annen viktig grunn til at vi valgte MIT-lisensen, er at den er kompatibel med andre populære

åpne kildekode-lisenser. Dette gjør det enklere å integrere vår programvare med andre verktøy og biblioteker, som OpenCV og Point Cloud Library (PCL), som også er lisensiert under lignende betingelser.

Det er viktig å merke seg at selv om programvaren vår er fritt tilgjengelig, påtar vi oss ikke ansvar for eventuell skade som kan oppstå ved bruk av den. Brukerne må derfor benytte programvaren på eget ansvar. Dette er en vanlig klausul i de fleste åpne kildekode-lisenser, inkludert MIT-lisensen.

Ved å dele vår kode og våre metoder, ønsker vi å gjøre det lettere for andre å bygge videre på vårt arbeid og fremme kunnskapsdeling i både akademia og industrien.

10.4 Selvtesting

Det ble gjort et forsøk på å sette opp testing av selvrammeverket. Denne selv-testingen ble ikke utført i den grad det var ønsket. I forsøket skulle Blender bli brukt for å eksternt sammenligne likheten mellom to punktskyfiler. Disse punktskyfilene ble generert fra selvtestrammeverket, f.eks med å legge til støy til en eksisterende punktsky. Selve sammenligningen mellom punktskyene var vellykket i den grad det var testet. Problemet oppsto når informasjonen skulle sendes tilbake til testrammeverket. Blender er lisensiert under GPL. Dette betyr at python-koden som ble skrevet for å kjøres i blender med bruk av Blender-API-et *bpy* også må være lisensiert under GPL. Dette vil videre begrense at testrammeverket i seg selv også må være lisensiert under GPL ettersom de to programmene skal dele informasjon med hverandre.

Det var ikke ønskelig å sette seg inn i dybden til alle kravene GPL setter for å lisensiere testrammeverket under denne lisensen. Et forsøk på kommunikasjon ble utført før lisensieringsproblemet kom til tankene. I dette forsøket ble resultatet fra Blender skrevet til en miljøvariabel som testrammeverket leste etter å ha kjørt python-koden i Blender. Det ble da funnet ut at miljøvariabler i python fungerer på en snedig måte, i tillegg til at på de fleste operativsystem (Windows, MacOS, Linux) vil de miljøvariablene bli kopiert når en ny prosess starter. På denne måten kan miljøvariabler kun påvirke prosessen i seg selv og prosesser den selv starter. I det andre forsøket hvor lisensieringsproblemet kom til lys var med å bruke navngitt delt minne mellom prosessene. Python har dette innebygd i sine standardbiblioteker, mens på C++ siden var det tenkt å bruke biblioteket *Boost.Interprocess*.

10.5 Registreringsmetoder

I prosjektet vårt har vi nøye vurdert ulike algoritmer for objektdeteksjon og estimering av transformasjoner. Det var viktig å velge riktig algoritme fordi hver algoritme har sine egne styrker og

svakheter som påvirker hvor nøyaktig og effektivt vi kan registrere punktskyene. Den største utfordringen var å finne en balanse mellom beregningskostnad og nøyaktighet. Algoritmer som ICP er svært nøyaktige under ideelle forhold, men de kan være sårbare for startverdier og støy i dataene. På den annen side, metoder som FPFH er raskere til å finne sammenhenger mellom ulike punktskyer, men kan være mindre nøyaktige i komplekse eller tett befolkede punktskyer. Resultatene våre, som vi har diskutert i resultat-seksjonen, viser at noen algoritmer fungerer godt i visse situasjoner, men ikke i andre.

10.5.1 Begrensninger

FPFH (Fast Point Feature Histograms) FPFH er en metode som hjelper med å registrere punktskyer raskere. Den bruker lokale geometriske trekk for å finne samsvar mellom punktskyer, noe som gjør prosessen mer effektiv. I vårt prosjekt viste FPFH betydelig raskere kjøretider sammenlignet med ICP, og er derfor godt egnet for applikasjoner som krever rask databehandling. Likevel er FPFH sensitiv for varierende punkttettheter og støy, og den yter ofte dårligere enn forventet i scenarier med kompleks geometri eller dårligere datakvalitet.

SHOT (Signature of Histograms of Orientations) SHOT er en algoritme som effektivt gjenkjenner trekk i punktskyer. Den gir detaljerte lokale beskrivelser som tåler betydelig støy i dataene. Ulempen med SHOT er at den krever relativt høy beregningskraft, noe som kan begrense bruken i sanntidsapplikasjoner. Metoden er også følsom for varierende punkttettheter, noe som gjør at data må forhåndsbehandles for optimal ytelse.

CVFH (Clustered Viewpoint Feature Histogram) CVFH er spesielt nyttig for objektgjenkjenning når objektene vises fra forskjellige synsvinkler. Denne metoden er designet for å være mer tolerant overfor endringer i synsvinkel enn mange andre trekkdeskriptorer. Likevel kan CVFH være mindre effektiv i svært tette og komplekse scenarier, der flere objekter kan skape forvirrende cluster-formasjoner som er vanskelige å skille fra hverandre.

3DSC (Three-Dimensional Shape Context) 3DSC gir en god kontekstuell representasjon av punktskyer, spesielt i miljøer med komplekse former og strukturer. Imidlertid er 3DSC kostbar både i beregning og minnebruk, noe som er en begrensning i større eller mer detaljerte datasett.

RSD (Radius-based Surface Descriptor) RSD tilbyr en effektiv måte å estimere lokale overflateformer på, noe som er viktig for mange punktskybehandlingsoppgaver. Selv om RSD er relativt

lettvekt og rask, er ytelsen avhengig av jevn punktdistribusjon. Ujevnheter og hull i dataene kan føre til feilaktig formestimering, noe som kan kreve ytterligere validering.

Disse funnene viser et klart kompromiss mellom beregningshastighet og detaljnivå i trekk-ekstraksjonen som hver metode tilbyr. Resultatene peker også på behovet for videre forskning og utvikling for å forbedre algoritmenes pålitelighet og effektivitet, spesielt i komplekse og dynamiske miljøer. Videre kan en kombinasjon av disse metodene utforskes for å skape en mer omfattende og bedre løsning for punktskyregistrering og objektgjenkjenning.

10.6 Bærekraft

En handling og valg som ble gjort var å bruke fysiske målobjekter. Disse fysiske målobjektene ble 3d-printet av PLA og PETG. Bruk av plast var kanskje ikke det mest bærekraftige valget. Et annet alternativ hadde vært å laget objektene fra å maskinere metall eller treverk.

De ulike registreringsmetodene som ble testet brukte ulik utregningstid mellom seg. Når registrering skal brukes utenom testrammeverket er det fornuftig å velge en metode som ikke bruker lang utregningstid. Lengre utregningstid vil normalt bety et høyere energiforbruk. Faktoren som påvirker forholdet mellom utregningstid og energiforbruket mest er om metoden kjører serielt eller parallelt på en hovedprosessor eller på en grafikkprosessor. En annen faktor er om sammensatte operasjoner brukes (SIMD eller BLAS). I testingen utført i denne oppgaven ble det ikke prøvd å påvirke disse faktorene i noen stor grad.

10.7 Etiske tanker

Registrering har ikke så mange etiske problemstillinger knyttet til seg. Gode registreringsmetoder kan derimot være med på forbedre resultater og korte ned ventetid på medisinske bilder og avbildinger. Med tanke på mulig bruk i militære droner og lignende er trolig eksisterende løsninger basert på GPS og 2d-bilder i farge eller i andre lysspektrum bedre ettersom nøyaktig utforming på mulige mål er normalt ukjent. Et unntak kan være for å lage større dybdekart for ulike formål. Noen tenkte bruksområder kan da være utregning av refleksjon i vegger mot et mål og estimering av svake punkt i konstruksjoner.

11 Konklusjon

Denne bachelor-oppgaven har utviklet et testrammeverk for utforsket effektiviteten, nøyaktigheten og begrensningene av ulike algoritmer for objektdeteksjon i punktskyer. Våre eksperimenter og analyser har bidratt til dypere innsikt i hvordan ulike tilnærminger presterer under varierte forhold, og har belyst viktige aspekter ved objektdeteksjon.

11.1 Oppsummering av hovedresultater

Gjennom hele prosjektet har vi grundig evaluert forskjellige metoder for registrering og objektdeteksjon i 3D-punktskyer. Vi utviklet et testoppsett som gjorde det mulig for oss å analysere ytelsen til metodene våre i et kontrollert, men realistisk miljø. Resultatene har vist at noen algoritmer er svært nøyaktige under optimale forhold, men de kan være sårbare for støy og hvordan de blir satt opp i starten. Metoder som er basert på trekk har vist seg å være raskere, men ytelsen deres kan variere avhengig av hvor tett punktene er og hvor mye støy det er. Dette viser hvor viktig det er å velge riktig algoritme basert på de spesifikke kravene til applikasjonen.

11.2 Vurdering av forskningsmål

Våre forskningsmål om å forbedre nøyaktigheten og effektiviteten i objektdeteksjon og estimering av transformasjoner i 3D-modellerte miljøer har delvis blitt oppnådd. Vi har oppdaget flere viktige begrensninger i de nåværende algoritmene, noe som viser at det er behov for videre utvikling for å kunne håndtere varierte og komplekse miljøer mer effektivt.

11.3 Betydningen av arbeidet

Resultatene fra arbeidet vårt kan ha en betydelig innvirkning på hvordan fremtidige applikasjoner innen robotikk, automatisert inspeksjon og virtuell virkelighet blir utviklet, spesielt når det gjelder å inkludere mer presis og effektiv objektdeteksjon. Innsiktene vi har fått kan også være en verdifull ressurs for andre forskere som jobber med lignende utfordringer.

11.4 Begrensninger og utfordringer

Selv om vi har lært mye gjennom testene våre, er det noen begrensninger vi må ta hensyn til. Testene ble utført i kontrollerte miljøer, og disse kan ikke helt gjenspeile alle de uforutsigbare faktorene vi kan møte i virkelige situasjoner. I tillegg er kompleksiteten og variasjonen i dataene en utfordring som krever mer forskning.

11.5 Forslag til fremtidig forskning

Ut fra erfaringene våre foreslår vi at fremtidig forskning bør fokusere på å utvikle hybridmetoder som kombinerer ulike algoritmer for å kompensere for deres individuelle svakheter. Det kan også være veldig nyttig å undersøke hvordan kunstig intelligens kan brukes til å forutsi hvilke algoritmer som vil fungere best basert på egenskapene til de innkommende dataene.

11.6 Personlige refleksjoner og lærdommer

Dette prosjektet har gitt oss verdifull innsikt i både tekniske og metodiske aspekter av objektdeteksjon. Arbeidet har styrket vår evne til kritisk tenkning og problemløsning, særs innenfor maskinsyn.

Referanser

[1] Adobe. *How to Convert Chrome HTML to PDF*. URL: https://www.adobe.com/uk/acrobat/resources/chrome-to-pdf.html.

[2] Aitor Aldoma. *CVFH*. 2011. URL: https://robotica.unileon.es/index.php?title=PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)#CVFH.

[3] Aitor Aldoma mfl. «CAD-model recognition and 6DOF pose estimation using 3D cues». I: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, s. 585–592. DOI: 10.1109/ICCVW.2011.6130296.

[4] K. S. Arun, T. S. Huang og S. D. Blostein. «Least-Squares Fitting of Two 3-D Point Sets». I: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (1987), s. 698–700. DOI: 10.1109/TPAMI.1987.4767965.

[5] Atlassian. *Code Review Best Practices*. 2023. URL: https://www.atlassian.com/blog/add-ons/code-review-best-practices.

[6] Atlassian. *What is Agile?* 2023. URL: https://www.atlassian.com/agile.

[7] Stephen Berry. *glaze*. 2024. URL: https://github.com/stephenberry/glaze.

[8] P.J. Besl og Neil D. McKay. «A method for registration of 3-D shapes». I: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), s. 239–256. DOI: 10.1109/34.121791.

[9] Tim Blechmann. *Chapter 20. Boost.Lockfree*. boost, 2011. URL: https://www.boost.org/doc/libs/1_84_0/doc/html/lockfree.html.

[10] Yang Chen og Gérard Medioni. «Object modelling by registration of multiple range images». I: *Image and Vision Computing* 10.3 (1992). Range Image Understanding, s. 145–155. ISSN: 0262-8856. DOI: https://doi.org/10.1016/0262-8856(92)90066-C. URL: https://www.sciencedirect.com/science/article/pii/026288569290066C.

[11] Yifan Chen mfl. «Iterative Feedback Network for Unsupervised Point Cloud Registration». I: *arXiv preprint arXiv:2401.04357* (2021). URL: https://arxiv.org/abs/2401.04357.

[12] Comau. *Cognitive robotics for advanced automated inspection*. 2024. URL: https://www.comau.com/en/2024/02/26/comau-and-leonardo-leverage-cognitive-robotics-for-automated-inspection/.

[13] Wikipedia Contributors. *Thread pool*. 2024. URL: https://en.wikipedia.org/wiki/Thread_pool.

[14] Zhihua Cui, Jianchao Zeng og Yufeng Yin. «An Improved PSO with Time-Varying Accelerator Coefficients». I: *2008 Eighth International Conference on Intelligent Systems Design and Applications*. Bd. 2. 2008, s. 638–643. DOI: 10.1109/ISDA.2008.86.

[15] Weiye Deng, Xiaoping Chen og Jingwei Jiang. «A Staged Real-Time Ground Segmentation Algorithm of 3D LiDAR Point Cloud». I: *Electronics* 13.5 (2024), s. 841. DOI: 10.3390/electronics13050841. URL: https://www.mdpi.com/2079-9292/13/5/841.

[16] Prof. Torgeir Dingsøyr. *Smidige utviklingsmetoder*. 2024. URL: https://snl.no/smidige_utviklingsmetoder.

[17] Emil Dotchevski. *QVM*. boost, 2023. URL: https://www.boost.org/doc/libs/1_84_0/libs/qvm/doc/html/index.html.

[18] Antony Dovgal og Takeshi Kanno. *Haru Free PDF Library*. 2000. URL: https://github.com/libharu/libharu.

[19] Cândida Ferreira. «Gene Expression Programming: A New Adaptive Algorithm for Solving Problems». I: *Complex Syst.* 13 (2001). URL: https://api.semanticscholar.org/CorpusID:1647961.

[20] Martin A Fischler og Robert C Bolles. «Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography». I: *Communications of the ACM* 24.6 (1981), s. 381–395. URL: https://dl.acm.org/doi/pdf/10.1145/358669.358692.

[21] Martin A. Fischler og Robert C. Bolles. «Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography». I: *Readings in Computer Vision*. Red. av Martin A. Fischler og Oscar Firschein. San Francisco (CA): Morgan Kaufmann, 1987, s. 726–740. ISBN: 978-0-08-051581-6. DOI: https://doi.org/10.1016/B978-0-08-051581-6.50070-2. URL: https://www.sciencedirect.com/science/article/pii/B9780080515816500702.

[22] Andrea Frome. *3DSC*. 2004. URL: https://robotica.unileon.es/index.php?title=PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)#3DSC.

[23] Andrea Frome mfl. «Recognizing Objects in Range Data Using Regional Point Descriptors». I: *Computer Vision - ECCV 2004*. Red. av Tomáš Pajdla og Jiří Matas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, s. 224–237. ISBN: 978-3-540-24672-5.

[24] Ion Gaztanaga. *Chapter 16. Boost.Interprocess*. boost, 2015. URL: https://www.boost.org/doc/libs/1_84_0/doc/html/interprocess.html.

[25] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989. URL: http://www2.fiit.stuba.sk/~kvasnicka/Free%20books/Goldberg_Genetic_Algorithms_in_Search.pdf.

[26] Xian-Feng Han mfl. «3D Point Cloud Descriptors in Hand-crafted and Deep Learning Age: State-of-the-Art». I: *arXiv preprint arXiv:1802.02297* (2021). URL: https://ar5iv.labs.arxiv.org/html/1802.02297v1.

[27] Xiaoshui Huang mfl. «A comprehensive survey on point cloud registration». I: *arXiv preprint arXiv:2103.02690* (2021). URL: https://ar5iv.labs.arxiv.org/html/2103.02690.

[28] IEEE & The Open Group. *POSIX.1-2017*. 2018. URL: https://pubs.opengroup.org/onlinepubs/9699919799/.

[29] ISO/IEC mfl. *Reflection for C++26 – P2996R0*. Okt. 2023. URL: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2996r0.html.

[30] Benoît Jacob mfl. *Eigen - TuxFamily*. 2024. URL: https://eigen.tuxfamily.org/index.php.

[31] J. Kennedy og R. Eberhart. «Particle swarm optimization». I: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Bd. 4. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[32] Andreas Krug mfl. *C++ keyword: reflexpr (reflection TS)*. cppreference.com, mai 2023. URL: https://en.cppreference.com/mwiki/index.php?title=cpp/keyword/reflexpr&oldid=151633.

[33] Wei Li, Hongtai Cheng og Xiaohua Zhang. «Efficient 3D Object Recognition from Cluttered Point Cloud». I: *Sensors* 21.17 (2021), s. 5850. DOI: 10.3390/s21175850. URL: https://www.mdpi.com/1424-8220/21/17/5850.

[34] David G Lowe. «Object recognition from local scale-invariant features». I: *Proceedings of the International Conference on Computer Vision* (1999), s. 1150–1157. URL: https://inst.eecs.berkeley.edu//~cs294-6/fa06/papers/LoweD_Object%20recognition%20from%20local%20scale-invariant%20features.pdf.

[35] Zhanat Makhataeva og Huseyin Atakan Varol. «Augmented Reality for Robotics: A Review». I: *Robotics* 9.2 (2020), s. 21. DOI: 10.3390/robotics9020021. URL: https://www.mdpi.com/2218-6581/9/2/21.

[36] Zoltan-Csaba Marton. *RSD*. 2010. URL: https://robotica.unileon.es/index.php?title=PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)#RSD.

[37] Zoltan-Csaba Marton mfl. «Combined 2D–3D categorization and classification for multimodal perception systems». I: *The International Journal of Robotics Research* 30.11 (2011), s. 1378–1402. DOI: 10.1177/0278364911415897. eprint: https://doi.org/10.1177/0278364911415897. URL: https://doi.org/10.1177/0278364911415897.

[38] Zoltan-Csaba Marton mfl. «General 3D modelling of novel objects from a single view». I: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Okt. 2010, s. 3700–3705. DOI: 10.1109/IROS.2010.5650434.

[39] Microsoft. *processthreadsapi.h header*. Jan. 2013. URL: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/.

[40] Klemens David Morgenstern. *Chapter 28. Boost.Process*. boost, 2016. URL: https://www.boost.org/doc/libs/1_84_0/doc/html/process.html.

[41] Mozilla. *How to print web pages in Firefox*. URL: https://support.mozilla.org/en-US/kb/how-print-web-pages-firefox.

[42] OpenCV. *Your 2024 Guide to the Top 6 Computer Vision Problems*. Accessed: 2024-05-19. 2024. URL: https://opencv.org/blog/computer-vision-problems/.

[43] Tihomir Orehovački. «Intelligent Robotics—A Systematic Review of Emerging Technologies and Trends». I: *Electronics* 13.3 (2024), s. 542. DOI: 10.3390/electronics13030542. URL: https://www.mdpi.com/2079-9292/13/3/542.

[44] Ixchel G. Ramirez-Alpizar. «Integrating Virtual, Mixed, and Augmented Reality to Human–Robot Interaction Applications Using Game Engines: A Brief Review of Accessible Software Tools and Frameworks». I: *Applied Sciences* 13.3 (2023), s. 1292. DOI: 10.3390/app13031292. URL: https://www.mdpi.com/2076-3417/13/3/1292.

[45] Intel RealSense. *Intel RealSense Depth Camera D435f*. 2022. URL: https://www.intelrealsense.com/depth-camera-d435f/.

[46] Stefan Reitmann. *BLAINDER—A Blender AI Add-On for Generation of Semantically Labeled Depth-Sensing Data*. 2021. URL: https://www.mdpi.com/1424-8220/21/6/2144.

[47] Antony Polukhin Renato Forti. *Chapter 12. Boost.DLL*. boost, 2023. URL: https://www.boost.org/doc/libs/1_84_0/doc/html/boost_dll.html.

[48] Radu Bogdan Rusu. *FPFH*. 2009. URL: https://robotica.unileon.es/index.php?title=PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)#FPFH.

[49] Radu Bogdan Rusu, Nico Blodow og Michael Beetz. «Fast Point Feature Histograms (FPFH) for 3D registration». I: *2009 IEEE International Conference on Robotics and Automation*. 2009, s. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473.

[50] Radu Bogdan Rusu og Steve Cousins. «3D is here: Point Cloud Library (PCL)». I: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, mai 2011.

[51] Radu Bogdan Rusu mfl. «Fast geometric point labeling using conditional random fields». I: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, s. 7–12. DOI: 10.1109/IROS.2009.5354763.

[52] Nizar Sallem. *pcl::PLYReader Class Reference (1.14.1-dev)*. PointClouds.org, mai 2024. URL: http://pointclouds.org/documentation/classpcl_1_1_p_l_y_reader.html.

[53] Federico Tombari. *SHOT*. 2010. URL: https://robotica.unileon.es/index.php?title=PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)#SHOT.

[54] Federico Tombari, Samuele Salti og Luigi Di Stefano. «A combined texture-shape descriptor for enhanced 3D feature matching». I: *2011 18th IEEE International Conference on Image Processing*. 2011, s. 809–812. DOI: 10.1109/ICIP.2011.6116679.

[55] Federico Tombari, Samuele Salti og Luigi Di Stefano. «Unique shape context for 3D data description». I: (okt. 2010), s. 59. DOI: 10.1145/1877808.1877821.

[56] Federico Tombari, Samuele Salti og Luigi Di Stefano. «Unique Signatures of Histograms for Local Surface Description». I: *Computer Vision – ECCV 2010*. Red. av Kostas Daniilidis, Petros Maragos og Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, s. 356–369. ISBN: 978-3-642-15558-1.

[57] Author(s) unknown. «Global ray-casting range image registration». I: *IPSJ Transactions on Computer Vision and Applications* (2021). URL: https://ipsjcva.springeropen.com/articles/10.1186/s41074-017-0025-4.

[58] Ranjith Unnikrishnan. «Statistical Approaches to Multi-scale Point Cloud Processing». I: (mai 2008), s. 82. DOI: 10.1145/1877808.1877821.

[59] Guoqi Xie og Renfa Li. «LiDAR Point Cloud Recognition and Visualization with Deep Learning for Overhead Contact Inspection». I: *Sensors* 20.21 (2020), s. 6387. DOI: 10.3390/s20216387. URL: https://www.mdpi.com/1424-8220/20/21/6387.

[60] Juyong Zhang, Yuxin Yao og Bailin Deng. «Fast and Robust Iterative Closest Point». I: *arXiv preprint arXiv:2007.07627* (2022). J. Zhang and Y. Yao are with the School of Mathematical Sciences, University of Science and Technology of China. B. Deng is with the School of Computer Science and Informatics, Cardiff University. Corresponding author: DengB3@cardiff.ac.uk. URL: https://ar5iv.labs.arxiv.org/html/2007.07627.

[61] Yuan Zhao mfl. «A unified framework for automated registration of point clouds, mesh surfaces and 3D models by using planar surfaces». I: *The Photogrammetric Record* 37.180 (), s. 366–384. DOI: https://doi.org/10.1111/phor.12428. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/phor.12428.

12 Vedlegg A – Forprosjektrapport

Se vedlagt zip-mappe.

13 Vedlegg B – Statusrapporter og møtereferater

Se vedlagt zip-mappe.

14 Vedlegg C – Møteinnkallinger

Se vedlagt zip-mappe.

15 Vedlegg D – Timelister

Se vedlagt zip-mappe.

16 Vedlegg E – Eksempel på konfigurasjoner og testspesifikasjoner

Se vedlagt zip-mappe.

17 Vedlegg F – Eksempel på autogenererte rapporter

Se vedlagt zip-mappe.

18 Vedlegg G – Poster

Se vedlagt zip-mappe.

19 Vedlegg H – Pitch presentasjon

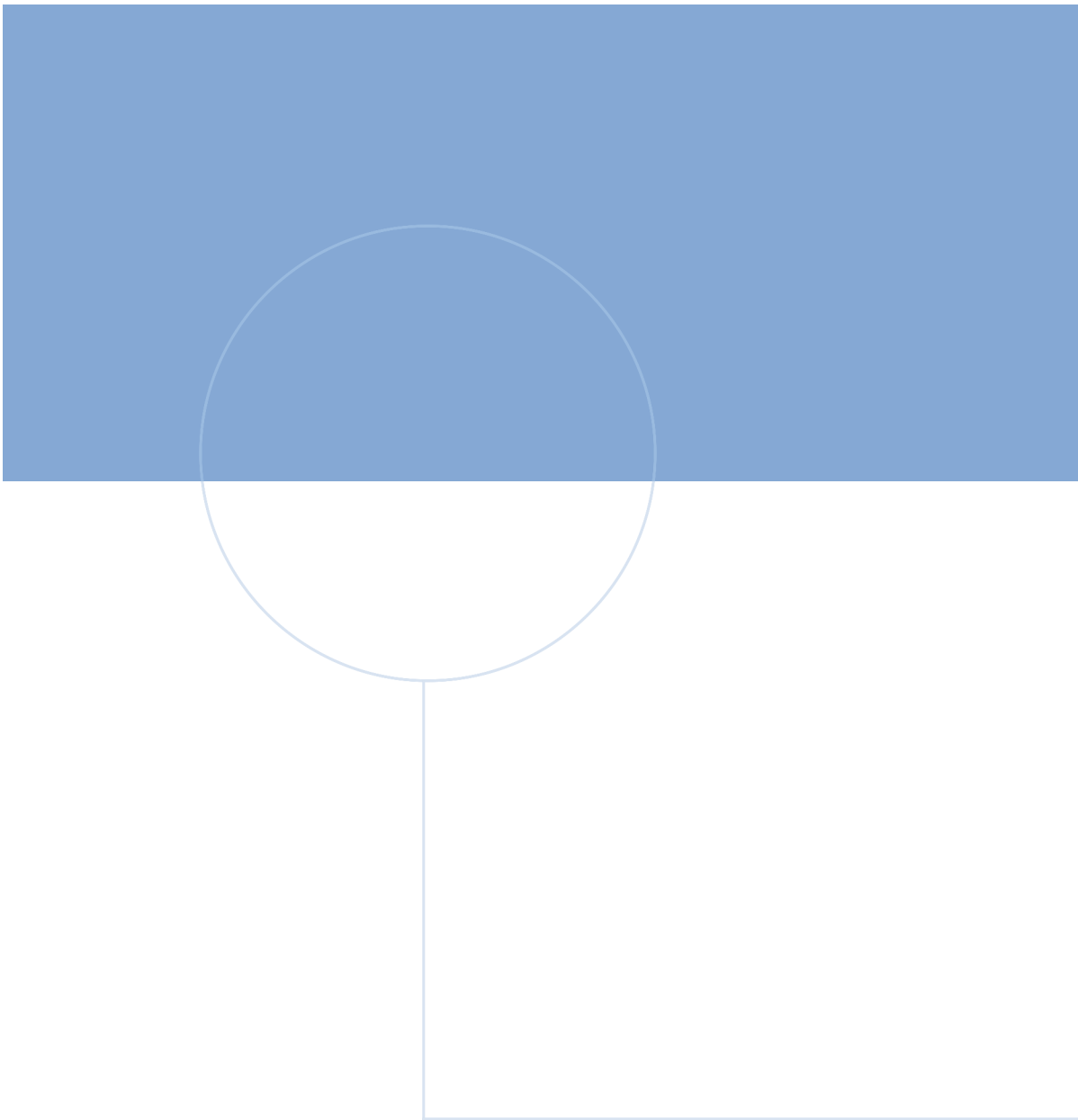
Se vedlagt zip-mappe.

20 Vedlegg I – Programkode mm.

Se vedlagt zip-mappe.

21 Vedlegg J – Presentasjon

Se vedlagt zip-mappe.



NTNU

Kunnskap for en bedre verden