2019•2020

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

Masterthesis

Ontkoppelde padplanning voor mobiele manipulatoren: verkennend onderzoek via simulatie van een robot die een oppervlak aftast

PROMOTOR :

Prof. dr. ir. Eric DEMEESTER

BEGELEIDER :

ing. David DE SCHEPPER

Thijs Mangelschots, Roel Vanonckelen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

Gezamenlijke opleiding UHasselt en KU Leuven

UHASSELT KU LEUVEN

UHASSELT KU LEUVEN

2019•2020

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

Masterthesis

Ontkoppelde padplanning voor mobiele manipulatoren: verkennend onderzoek via simulatie van een robot die een oppervlak aftast

PROMOTOR :

Prof. dr. ir. Eric DEMEESTER

BEGELEIDER :

ing. David DE SCHEPPER

Thijs Mangelschots, Roel Vanonckelen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

UHASSELT KU LEUVEN

Deze masterproef werd geschreven tijdens de COVID-19 crisis in 2020. Deze wereldwijde gezondheidscrisis heeft mogelijk een impact gehad op de opdracht, de onderzoekshandelingen en de onderzoeksresultaten.

Woord vooraf

Deze masterproef kadert in het onderzoek van de onderzoeksgroep ACRO van KU Leuven naar het gebruik van mobiele manipulatoren ter ondersteuning van het ARCHER-project. Dit project tracht de ontmanteling van nucleaire sites te automatiseren gebruik makend van mobiele manipulatoren. Dergelijke robots bieden nieuwe mogelijkheden in industriële automatisering, maar zijn wegens een complexe besturing van redundante vrijheidsgraden nog niet op grote schaal toepasbaar. Daarom onderzoekt deze masterproef welke padplanningsmethoden ervoor bestaan, welke zowel het mobiel platform als de manipulator aansturen en op welke manier deze toepasbaar zijn op een robot van ACRO, genaamd AMBER.

We kozen dit onderwerp aangezien het ons als studenten elektromechanica interessant leek om bijkomend op de mechanicaonderwerpen ook kennis op te doen omtrent automatiseringsonderwerpen zoals robotica.

Graag danken wij de personen die deze masterproef mogelijk gemaakt hebben en ons erdoorheen begeleid hebben. Daarom danken we Prof. dr. ir. Eric Demeester voor het mogelijk maken van de masterproef aan de onderzoeksgroep ACRO en ing. David de Schepper voor de bereidbaarheid en vlotte begeleiding tijdens de masterproef. Tot slot danken we onze familie voor de steun en het geduld tijdens de uitvoering van de masterproef doorheen het academiejaar.

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Abstract

De onderzoeksgroep ACRO van KU Leuven doet onderzoek naar de sensorgebaseerde aansturing van mobiele manipulatoren ter ondersteuning van het ARCHER-project, dat de ontmanteling van nucleaire sites wil automatiseren. Dergelijke robots bieden vele mogelijkheden in industriële automatisering, maar zijn wegens een complexe besturing en gebruik van redundante vrijheidsgraden tot hiertoe weinig gebruikt. Daarom onderzoekt deze masterproef welke padplanningsmethoden bestaan die zowel het mobiel platform als de manipulator aansturen. Een praktische validatie in simulatie analyseert op welke manier een dergelijke methode toepasbaar is op de mobiele manipulator AMBER voor het scannen van een muur naar gesimuleerde radiologische bronnen.

Een literatuurstudie zorgt voor kennis over bestaande padplanningsmethoden voor mobiele manipulatoren. Verder gebeurt de analyse van planners door simulaties, sensordataverwerking, mapping, lokalisatie en navigatie met behulp van het Robot Operating System dat zorgt voor de samenwerking tussen de simulatieomgeving Gazebo en de visualisatieomgeving RViz.

Uit de literatuurstudie volgt dat een ontkoppelde methode die de basis (met A* en DWA) en de arm (met RRT-Connect) apart plant, beter toepasbaar is binnen de masterproef dan gecombineerde planners. Een robotmodel van AMBER laat toe om planners in simulatie te testen. Onzekerheid op de basislokalisatie hindert manipulatorplanning gebruik makend van de basispose. Dit resulteert in verdere ontkoppeling. De methode is bruikbaar voor scanopdrachten tot 3,5 m.

Abstract in English

The research group ACRO of KU Leuven researches sensor-based control of mobile manipulators in support of the ARCHER project, which targets to automate the dismantling of nuclear sites. Such robots introduce new opportunities for industrial automation but are not yet deployed in many applications due to a complex control of their redundant degrees of freedom. Therefore, this master's thesis explores existing path planners that control both the mobile platform and manipulator. Furthermore, a practical validation in simulation analyses how such a planner is applicable to the mobile manipulator named AMBER for scanning a wall with simulated radioactive sources.

First, a literature study provides knowledge about existing path planning methods for mobile manipulators. Furthermore, the analysis of integrated planners using simulations, sensor data processing, mapping, localisation and navigation all happen using the Robot Operating System which provides the co-operation between the physics engine Gazebo and the visualisation environment RViz.

The literature study concludes that a decoupled method that controls the base (with A* and DWA) and the arm (with RRT-Connect) separately is more suitable for this master's thesis than a combined planner. A robot model of AMBER allows testing of planners for the execution of tasks in simulation. Uncertainties on the localisation of the base obstructs manipulator planning using the base pose. This results in further decoupling. The method is suitable for scan tasks up to 3,5 m.

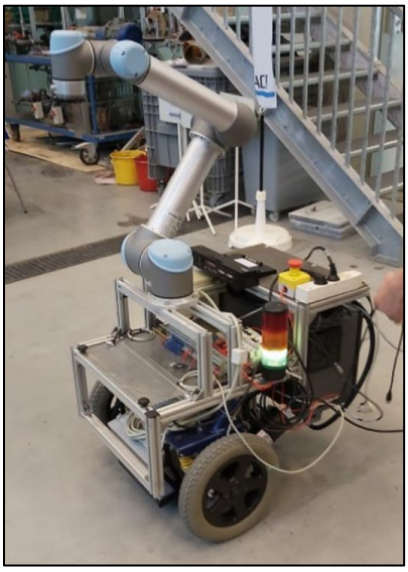
1 Inleiding

1.1 Situering

Deze masterproef kadert in verschillende onderzoeken omtrent autonome mobiele robots aan het onderzoekscentrum ACRO van de KU Leuven, gelegen in het technologiecentrum te Diepenbeek. De naam ACRO staat voor de expertise van de onderzoeksgroep: Automatisatie, Computervisie en Robotica. De onderzoeksgroep voert onderzoek uit naar deze vakgebieden en begeleidt bedrijven met de industriële applicatie ervan [1]. De masterproef kadert eveneens in het ARCHER-project (Autonomous Robotic platform for CHaractERrisation). Dit project tracht het karakterisatieproces tijdens het ontmantelen van nucleaire omgevingen te automatiseren met behulp van mobiele manipulatoren [2]. Het ARCHER-project is een lopend onderzoek aan de onderzoeksgroep ACRO en aan de onderzoeksgroep voor nucleaire technologie van de UHasselt, genaamd NuTeC. Het academisch onderzoek van de onderzoeksgroepen dient ter ondersteuning van de twee grootste bijdragers aan het ARCHER-project, namelijk de bedrijven Tecnubel en Magics. Technubel staat in voor het ontwerp en de bouw van een mobiele manipulator en krijgt hulp van Magics omtrent het ontwerp van de benodigde elektronica en sensoren.

De opdracht bestaat uit de implementatie van hardware (i.e. sensoren) en software (i.e. padplanners) voor een mobiele manipulator dat autonoom kan rondrijden en taken kan uitvoeren in een statische omgeving. Dit betreft de autonome lokalisatie en navigatie van de robot zoals het in kaart brengen van de omgeving en de padplanning doorheen de omgeving om een taak uit te voeren. Als *use case* bestaat deze taak uit het autonoom rondrijden en aftasten van een muuroppervlak in de omgeving om geïmiteerde radiologische bronnen in kaart te brengen. Wegens de maatregelen betreffende COVID-19 gebeurt dit enkel in simulatie voor een robotmodel in plaats van ook op een werkelijke mobiele manipulator, zoals origineel gepland.

Deze masterproef bouwt verder op een vorige masterproef waarvoor een mobiel platform ontwikkeld werd uitgaand van een rolstoelchassis [3]. Dit platform werd door doctoraatsstudenten van ACRO mechanisch en elektrisch aangepast en vervolledigd met een robotarm tot een mobiele manipulator, genaamd AMBER (ACRO MoBile Platform Extended with Robotic arm). Figuur 1.1 toont het rolstoelchassis van de vorige masterproef en de mobiele manipulator AMBER met de UR5-robotarm en het aangepast rolstoelchassis.



Figuur 1.1: (links) Mobiel platform gemaakt uit een rolstoelchassis tijdens een voorgaande masterproef [3]. (rechts) Het elektrisch en mechanisch aangepast mobiel platform uitgebreid met de robotarm UR5 tot de mobiele manipulator AMBER (ACRO MoBile Platform Extended with Robotic arm)

Mogelijke (industriële) toepassingen van de mobiele manipulator zijn:

- navigeren tussen werkplekken om objecten te verplaatsen;
- coöperatie tussen mens en robot op basis van krachtcontrole;
- navigeren doorheen radioactieve gebieden en het lokaliseren van radioactieve bronnen (ARCHER-project);
- rondleidingen geven aan bezoekers doorheen het robotlabo van ACRO in het Technologiecentrum.

1.2 Probleemstelling

Mobiele manipulatoren zoals AMBER en industriële varianten zijn nog niet (voldoende) in staat om autonoom te navigeren en taken uit te voeren in statische omgevingen. Dat vormt een obstakel voor industrieën waar de productieprocessen vaak veranderen om een ander model, serie of variant te produceren. Deze verandering is een gevolg van de hedendaagse trend van de industrie waarbij bedrijven overgaan naar de productie van kleinere seriegroottes op maat van de sneller veranderende vraag. Hierdoor is er nood aan een flexibele automatisering van productieprocessen. Verder is het gebrek aan autonome mobiele manipulatoren een obstakel voor industrieën waar werknemers zware repetitieve taken moeten uitvoeren. Tot slot zijn er naast de traditionele productie-industrie nog andere sectoren die mobiele manipulatoren kunnen gebruiken. Zo vormt het gebrek aan autonome mobiele manipulatoren een probleem voor bedrijven die verantwoordelijk zijn voor het ontmantelen van nucleaire omgevingen en dit willen automatiseren.

Manipulatoren worden tot nu toe hoofdzakelijk statisch gebruikt. De reden hiervoor is dat de integratie van manipulatorsoftware en mobielplatformsoftware tot één geheel nog niet genoeg ontwikkeld is om grootschalig, industrieel toe te passen. Immers, de combinatie van een mobiel platform en manipulator resulteert in een complexe besturing met meerdere vrijheidsgraden. De afwezigheid van zulke mobiele manipulatoren kan de verdere automatisering van industrieën hinderen met als gevolg een gebrek aan verbetering van arbeidsintensieve werkomstandigheden. Bijkomend resulteert een gebrek aan automatisering in de nucleaire sector in duurdere en langere

ontmantelingsprocessen van nucleaire omgevingen boven op de gevaarlijke radioactieve blootstelling aan operatoren.

Het antwoord op volgende hoofdonderzoeksvraag tracht oplossingen te bieden voor de eerder vermelde problemen:

- Welke mogelijke padplanners bestaan er en welke mogelijkheden zijn er om zowel het mobiel platform als de robotarm aan te sturen?

Deze hoofdonderzoeksvraag kan aangevuld worden met volgende deelvragen:

- Op welke manier is dergelijke padplanner toepasbaar op AMBER?
- Op welke manier functioneert AMBER met dergelijke planner bij het autonoom uitvoeren van taken in een statische omgeving?
- Hoe gaat de planner in een efficiënte manier om met redundantie in de kinematische structuur van AMBER?

1.3 Doelstellingen

De hoofddoelstelling van de masterproef is de implementatie van hardware (i.e. sensoren) en software (i.e. padplanners) voor een mobiele manipulator dat autonoom kan rondrijden en taken kan uitvoeren in een statische simulatieomgeving. Deze taak bestaat uit het autonoom rondrijden en aftasten van een muuroppervlak in de omgeving om geïmiteerde radiologische bronnen in kaart te brengen. Dit kan verder gespecificeerd worden in volgende doelstellingen:

- het gebruik van sensoren, zoals laserscanners of camera's, voor de besturing (waaronder kaartopbouw) van de mobiele manipulator;
- het plannen van een botsingsvrij pad om van een startpositie naar een eindpositie te gaan (deterministisch en/of stochastisch);
- het besturen van meerdere vrijheidsgraden (> 6), waarbij een deel van de vrijheidsgraden mobiel zijn (i.e. een mobiel platform dat beschikt over drie DoF[1]) en een deel statisch zijn (i.e. een industriële robotarm dat beschikt over zes DoF).

1.4 Methode

Allereerst zorgt een literatuurstudie omtrent padplanningsalgoritmes en kaarttransformaties voor begrip in de opbouw en werking van robotnavigatie. Verder levert de literatuurstudie naar bestaande padplanners voor mobiele platformen en manipulatoren inzicht in de huidige navigatietechnologieën voor dergelijk complexe robots. Hieruit moet blijken welke padplanner(s) toepasbaar zijn op de robotarm en het mobiel platform van AMBER en in welke mate deze eventueel combineerbaar zijn voor de mobiele manipulator met meer dan zes vrijheidsgraden.

Simulaties van de padplanner(s) op een computermodel van de mobiele manipulator testen de toepasbaarheid ervan. Deze taken worden toegepast op een *use case* in het kader van het ARCHER-project. Dit betreft het aftasten van een muur om scans uit te voeren naar geïmiteerde radiologische

[1] DoF = Degrees of Freedom (vrijheidsgraden)

bronnen. Toekomstig onderzoek zou de implementaties voor de simulatie kunnen toepassen op de werkelijke robot AMBER.

De simulaties, sensordataverwerking, mapping, lokalisatie en navigatie van de robot alsook de integratie van verschillende padplanners gebeuren allemaal met behulp van het Robot Operating System (ROS). ROS is een raamwerk voor het schrijven van robotsoftware dat compatibel is met verschillende robotplatformen en beschikt over een verzameling van bibliotheken en hulpmiddelen [4].

Het gebruikte materiaal en de gebruikte software doorheen de masterproef bestaat uit:

- AMBER,
- Sensoren van het merk Hokuyo en Microsoft Kinect,
- ROS,
- Gazebo: een *physics engine* als simulatieomgeving, compatibel met ROS,
- RViz: een visualisatieomgeving om de navigatie van mobiele basis en manipulator uit te voeren, compatibel met ROS.

1.5 Vooruitblik

Dit eindwerk bestaat uit verschillende hoofdstukken. Allereest zorgt een literatuurstudie voor de nodige kennis omtrent de werking van kaarttransformaties, van padplanningsalgoritmes als globale planners, van obstakelvermijding als lokale planners en van bestaande padplanningsmethodes voor mobiele manipulatoren met redundante vrijheidsgraden. Verder overloopt het hoofdstuk over de simulatie in het ROS-raamwerk de modellering en aansturing van AMBER in simulatie. Hierna behandelt het hoofdstuk omtrent de use case de toepasbaarheid van de geïmplementeerde planners op AMBER voor het aftasten van een muur om scans uit te voeren naar geïmiteerde radiologische bronnen. Tot slot zorgt een algemene conclusie voor een korte samenvatting van het eindwerk.

2 Literatuurstudie

2.1 Inleiding

Het doel van robotnavigatie is om een doelpositie zo efficiënt en zo betrouwbaar mogelijk te bereiken met behulp van omgevingskennis, een doelpositie en sensordata [5], [6]. De navigatie bestaat uit twee complementaire delen. Enerzijds bestaat de navigatie uit het plannen van een pad. Dit betreft de redenering op lange termijn om de globale bewegingen te plannen met de veronderstelling dat het wereldmodel correct is. Dit vergt computationeel veel rekentijd en is gebaseerd op een incomplete voorstelling van de wereld aangezien het geen rekening houdt met onvoorziene gebeurtenissen. Anderzijds bestaat de navigatie uit het vermijden van obstakels. Dit betreft redeneringen op korte termijn gebaseerd op sensordata om op onvoorziene gebeurtenissen te anticiperen en obstakels te vermijden. Dit is gebaseerd op nieuwe informatie over de omgeving doorheen het pad om op de dynamische wereld te reageren. De obstakelvermijding kan enerzijds het robotgedrag lokaal beïnvloeden waarbij het globale plan niet verandert. Anderzijds kan obstakelvermijding het gedrag globaal beïnvloeden waardoor de planning zich aanpast met behulp van nieuwe informatie. Tot slot is een planner slechts compleet wanneer het voor alle doelen de eindpositie kan bereiken, gegeven dat er een mogelijk pad bestaat. Indien er geen oplossing bestaat, moet de planner dit in een eindig tijdsinterval melden. Een volledig complete planner is in de praktijk moeilijk te realiseren wegens de computationele complexiteit van de wereldomgeving en de denkwijze.

Een basiskennis omtrent deze navigatieaspecten van mobiele robots en manipulatoren is vereist om de padplanningsdoelstellingen voor de mobiele manipulator AMBER te voltooien. Daarom behandelt dit hoofdstuk een literatuurstudie over de belangrijkste vereiste achtergrondkennis voor de masterproef. Allereerst schept het onderdeel over zoekalgoritmes inzicht omtrent het doel, de algemene werking en de verschillende soorten van padplanningsalgoritmes die een uitvoerbaar pad zoeken doorheen een omgeving. Vervolgens behandelt het onderdeel over kaarttransformaties de verschillende aanpakken die er bestaan om een discrete voorstelling van de ruimte te vormen. Verder bespreekt het onderdeel over obstakelvermijding de verschillende bijkomende algoritmes om onvoorziene obstakels te vermijden. Tot slot gaat het onderdeel over padplanning van mobiele manipulatoren met meerdere vrijheidsgraden na welke relevante padplanningsmethodes reeds bestaan. Hieruit volgt een toepasbare padplanner voor AMBER waarrond de masterproef verder werkt.

2.2 Soorten zoekalgoritmes

Dit onderdeel behandelt de bestaande algoritmes waarvan padplanningsalgoritmes gebruik maken. Het doel van dergelijke algoritmes is om een pad te zoeken van een beginpunt naar een eindpunt. Deze punten stellen de verzameling voor van mogelijke posities voor een robot of robotarm in de werkomgeving. De algoritmes kunnen bijkomend paden genereren op basis van voorwaarden zoals kortste weg, snelste rekentijd of zo weinig mogelijke mechanische arbeid.

Zoals beschreven in [6] stelt de toestandsruimte X de wereld voor waarin de padplanning gebeurt. De toestandsruimte is de verzameling van mogelijke poses van een robot in een omgeving of wereld. Hoofdstuk 2.2 over kaarttransformaties behandelt de manier waarop de wereld wordt voorgesteld als deze verzameling X en de manier waarop deze punten verbonden worden. Wanneer de verzameling gekend is, kan de wereld getransformeerd worden in acties die uitvoerbaar zijn door

de planner. Elke actie u, uitgevoerd vanuit de huidige status, heeft als gevolg een nieuwe status x'. Hierbij stelt de toestandstransitievergelijking de nieuwe status voor als een functie van de statustransformatiefunctie f die afhankelijk is van de huidige status en de uitgevoerde actie:

$$x' = f(x, u) \tag{2.1}$$

Verder stelt de *action space* U(x) een verzameling voor van alle mogelijke acties vanuit de status x. Hieruit volgt de verzameling U van alle mogelijke acties vanuit alle mogelijke statussen:

$$U = \bigcup_{x \in X} U(x) \tag{2.2}$$

Een discreet uitvoerbare padplanning bestaat uit de taak om een eindige opeenvolging van acties te vinden die een initiële status x_I transformeert naar een status die onderdeel uitmaakt van de verzameling van eindstatussen X_G. Het discreet padplanningsmodel bestaat uit:

1 Een niet lege *state space* X met een niet oneindige verzameling aan statussen.
2 Een eindige *action space* U(x) voor elke status x uit X.
3 Een statustransformatiefunctie f die een status f(x,u) produceert met x uit X en u uit U(x) waarbij de toestandstransitievergelijking bestaat uit x' = f(x,u).
4 Een initiële status x_I uit X.
5 Een verzameling van eindstatussen of *goal states* X_G die deel uitmaakt van X.

Grafegebaseerde algoritmen zijn, zoals beschreven in [6], algoritmes die elk punt (=een status uit de verzameling van mogelijke statussen) beurt om beurt controleert of het voldoet aan gegeven voorwaarden. Padplanningsalgoritmes worden beschouwd als Grafegebaseerde algoritmen bekeken vanuit een padplanningsperspectief, namelijk het zoeken van een aaneengesloten pad tussen punten tot aan het doel. Dergelijke algoritmes moeten van systematische aard zijn. Dit betekent dat het algoritme elke bereikbare status moet bezoeken (=controleren) in een eindige rekentijd, gegeven een eindige verzameling aan statussen, om tot een conclusie te komen of er een mogelijke oplossing bestaat. Bij het vinden van een pad moet het algoritme niet verder zoeken doorheen de punten. Het algoritme moet de statussen onthouden die reeds bezocht zijn om een oneindige circulatie doorheen de punten te voorkomen. Kortom, er mag geen overtollige exploratie van punten uit de verzameling voorkomen opdat het algoritme van systematische aard is. Onderstaande algoritme toont de algemene werkwijze van een voorwaarts padplanningsalgoritme. Hierbij begint het algoritme bij de startpositie en doorzoekt het alle punten totdat het de eindpositie bereikt.

```
FORWARD_SEARCH
1   Q.Insert(x_I) and mark x_I as visited
2   while Q not empty do
3       x ← Q.GetFirst()
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' not visited
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

Figuur 2.1: Algemene pseudocode voor een voorwaarts padplanningsalgoritme die de lijst Q van mogelijke posities doorloopt totdat het einddoel bereikt is [2, p. 33].

Tijdens bovenstaande iteratie zijn drie toestanden mogelijk. Allereerst bevinden alle punten die nog niet bezocht zijn door het algoritme, zich in de onbezochte toestand. Initieel geldt dit voor alle punten buiten de startpositie x_I. Ten tweede bevinden alle punten die bezocht zijn en waarvan alle naburige punten bezocht zijn, zich in de dode toestand. Een naburig punt is een volgende status x' waarvoor een actie u bestaat die deel uitmaakt van U(x) zodat x' = f(x,u). Een punt in de dode toestand draagt niet meer bij tot de zoektocht aangezien er geen nieuwe wegen uit volgen die leiden tot het vinden van een uitvoerbaar pad. Tot slot bevinden punten die reeds bezocht zijn maar waarvan de naburige punten nog niet bezocht zijn, zich in de levende toestand. Deze punten kunnen nog bijdragen aan de zoektocht naar een pad. Initieel bevindt enkel de startpositie x_I zich in de levende toestand.

Het padplanningsalgoritme verloopt als volgt. De wachtlijst Q bevat een verzameling van punten die zich in de levende toestand bevinden. Het enige verschil tussen de verschillende algoritmes die verder aan bod komen is de manier waarop de algoritmes deze wachtlijst sorteren of doorlopen. Bij de start van het algoritme bevat Q enkel de startpositie x_1. Vervolgens start lijn 2 een *while* iteratie op voorwaarde dat Q niet leeg is. De iteratie stopt enkel wanneer Q leeg is en er bijgevolg geen oplossing gevonden is, waarbij lijn 13 *failure* meldt. Bij het begin van elke iteratie van de *while* lus haalt het algoritme het eerste element x uit de lijst Q. Een oplossing is gevonden indien x een element is van de verzameling van de eindposities X_G, waarbij lijn 5 *succes* meldt. Indien niet, doorloopt het algoritme vanaf lijn 6 alle mogelijke acties u die behoren tot U(x) om een volgende status te bereiken. Voor elke volgende status x'=f(x,u) (=naburig punt) moet het algoritme vanaf lijn 7 nagaan of het reeds bezocht is. Indien x' onbezocht is, markeert het algoritme x' als bezocht en plaatst het x' in de wachtlijst Q. Indien x' reeds bezocht is, bevindt het zich in de dode toestand en zit x' al in Q.

Een punt kan ook een kost bevatten. Dit is een waarde voor de afstand naar het punt, voor de mechanische arbeid of voor het aantal te passeren punten vereist om het punt te bereiken. Deze kost heeft een navigerende functie aangezien het kan leiden tot een zo kort mogelijk of een minst

intensief pad. Stap 12 update de kost van een volgend punt x' indien de kost om punt x' via het reeds gekend pad te bereiken hoger is. Het Dijkstra en A* algoritme maken gebruik van een kostwaarde. Het voorbeeld in hoofdstuk 2.2.3 over het Dijkstra algoritme verduidelijkt dit principe.

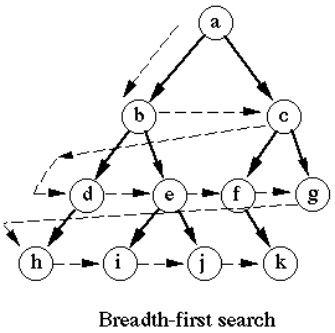
Het algoritme geeft enkel aan of er al dan niet een mogelijke oplossing bestaat. Door na lijn 7 een actie toe te voegen die x uit de huidige iteratie linkt met x uit de vorige iteratie, vormen deze links uit elke iteratie het totaal gevonden pad.

Het algoritme moet niet meer controleren of een punt reeds bezocht is indien Q van boomstructuur is, zoals bij 2.2.1 breadth-first en 2.2.2 depth-first. In deze gevallen zijn er geen herhaalde bezoeken mogelijk. Bij een rasterindeling van Q, kan een opzoektabel fungeren als controle om te achterhalen of een punt reeds bezocht is door het te vergelijken met alle levende en dode punten. Dit vergt echter veel computationeel rekenwerk wanneer de representatie van elk punt lang is. Een andere mogelijkheid is het toelaten van herhaalde controles. Dit verhoogt wederom de computationele rekentijd.

Volgende hoofdstukken overlopen verschillende soorten algoritmes. Elk van deze algoritmes berust op hetzelfde principe als het besproken voorwaarts padplanningsalgoritme met de sorteermethode voor Q als enig verschil.

2.2.1 Breadth-first

Een eerste algoritme is het *breadth-first* of 'breedte eerst' algoritme dat gebruik maakt van het *First-In First-Out* (FIFO-principe) of 'eerst in eerst uit' principe voor de wachtlijst Q [5]. Onderstaande Figuur 2.2 illustreert deze sorteermethode.

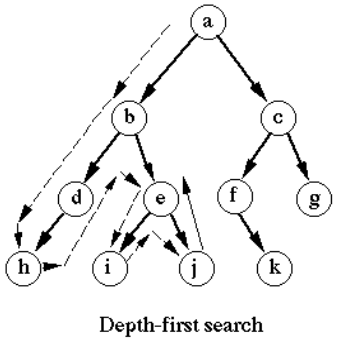


Figuur 2.2: Representatie van de breadth-first sortering van Q die de lijst Q eerst in de breedte doorloopt alvorens het volgend dieper niveau in de breedte te onderzoeken [7].

Dit algoritme is geschikt voor een wereldmodel met een boomstructuur. Hierbij voegt het algoritme eerst alle mogelijke volgende statussen van de huidige status toe aan de lijst. Hierna zoekt het van de eerste opvolger opnieuw alle volgende statussen alvorens dit te doen voor de tweede opvolger. Op deze manier groeit het zoekfront uniform doorheen de lijst Q. Hierdoor zijn alle paden met k aantal stappen uitgeput alvorens het algoritme de paden met k+1 aantal stappen onderzoekt waardoor het een pad met het minst aantal stappen garandeert [6]. Bovendien moet het algoritme in lijn 12 geen update uitvoeren bij het detecteren van een reeds bezocht punt aangezien dit punt slechts via één pad te bereiken is. Tot slot is het algoritme systematisch dankzij het uniform zoekfront.

2.2.2 Depth-first

Een tweede algoritme is het *depth-first* of 'diepte eerst' algoritme dat gebruik maakt van het *Last-In First-Out* (LIFO-principe) of 'laatste in eerst uit' principe voor de wachtlijst Q [5]. Zoals te zien in onderstaande Figuur 2.3 is dit algoritme wederom geschikt voor een wereldmodel met een boomstructuur.



Figuur 2.3: Representatie van de depth first sortering van Q die de lijst Q eerst in de diepte van een tak onderzoekt alvorens een volgende tak in de diepte te onderzoeken [7].

Hierbij voegt het algoritme eerst alle mogelijke volgende statussen van de huidige status toe aan de lijst. Hierna zoekt het van één van deze volgende statussen weer alle mogelijke volgende statussen, enzovoort. Zo onderzoekt het eerst een hele tak alvorens het overgaat naar de volgende tak. Op deze manier zoekt het algoritme in de diepte waardoor het eerst langere paden onderzoekt. Zoals beschreven in [6] bepaalt de actie in de *forall* lus op welke manier het algoritme een keuze maakt tussen de volgende mogelijke punten om als eerste dieper te onderzoeken. Wederom moet, dankzij de boomstructuur, het algoritme geen update uitvoeren in lijn 12 bij het bezoeken van een reeds bezocht punt. Het algoritme is systematisch bij een eindig aantal punten maar niet bij een oneindige aantal punten omdat het dan mogelijk is dat het een oneindig lange tak blijft onderzoeken. Hierdoor focust het algoritme bij een zeer groot aantal punten zich op één richting waardoor het grote delen van de te onderzoeken ruimte over het hoofd slaat.

2.2.3 Dijkstra

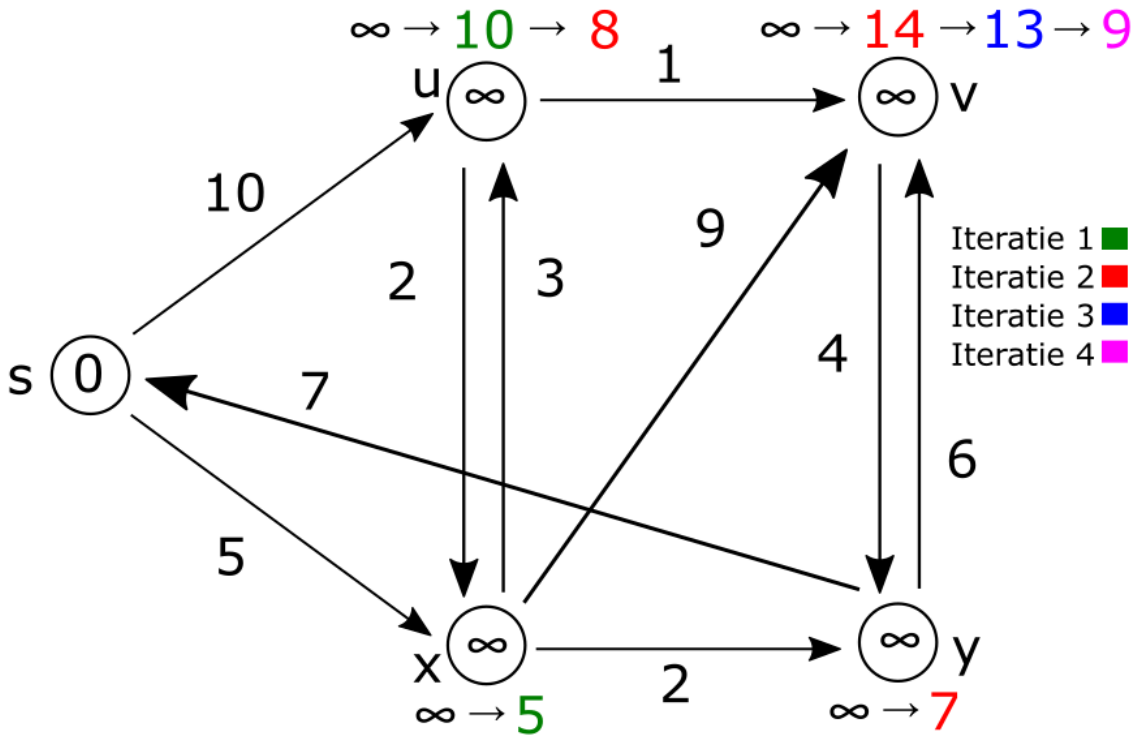
De tot hiertoe besproken algoritmes hebben geen reden om het ene pad over het andere te kiezen tijdens de zoektocht doorheen de punten. Zoals beschreven in [8] maakt Dijkstra's algoritme gebruik van een kostentoekenning aan de verbindingen tussen punten om via een actie u het punt x te bereiken. Deze kost kan een afstand of mechanische moeite zijn. Hierbij is Q een wachtlijst opgedeeld op basis van prioriteit met behulp van de *cost-to-come* (of kost om te bereiken) functie g(x). De kost om een volgend punt te bereiken is bijgevolg de som van de kost om tot het huidig punt te geraken en de kost voor de actie naar het volgend punt vanuit het huidig punt:

$$g(x') = g(x) + l(x, u) \tag{2.3}$$

Door deze prioriteit heeft het eerste punt in Q de laagste kost om bereikt te worden. Wanneer het algoritme uit Figuur 2.1 tijdens lijn 3 het punt x uit Q haalt, bestaat er geen enkel ander pad om x te bereiken met een lagere kost. Op deze manier vindt Dijkstra's algoritme steeds het kortste pad. Een nadeel is echter dat het alle punten in de lijst Q overloopt, waaronder ook punten die niet in de richting van het doel liggen. Dit algoritme is geschikt voor wereldmodellen waarbij alle punten

verbonden zijn. Volgend voorbeeld uit [5] verduidelijkt het Dijkstra algoritme op basis van het reeds besproken voorwaarts padplanningsalgoritme.

Beschouw Figuur 2.4 en veronderstel de volgende punten als mogelijke statussen in het wereldmodel met s als startpositie. Verschillende punten zijn met elkaar verbonden via acties die elk een kost hebben om uit te voeren. De kost om de startpositie te bereiken is uiteraard 0 en de kost om andere punten te bereiken is initieel oneindig aangezien het algoritme nog niet begonnen is.



Figuur 2.4: Voorbeeld van het Dijkstra algoritme waarbij de verschillende punten uit Q elk een kost hebben om te bereiken vanuit het startpunt [5].

Het algoritme gaat van start en voert de lijnen code uit Figuur 2.1 uit.

Tabel 2.1: Stapsgewijze uitvoering van Dijkstra's algoritme op basis van de pseudocode van een voorwaarts zoekalgoritme.

L1: Q: s(0,s)	Het algoritme vertrekt met enkel s in Q. De kost is 0 en aangezien s niet het doel is, worden alle mogelijke opvolgers berekend.
Iteratie 1 begint	
L8: opvolgers s(0,s): u(0+10,s) en x(0+5,s)	Het algoritme berekent de kost van de opvolgers van het eerste element uit Q om vanuit de *parent* (= huidig punt) de *child* (= volgend punt) te bereiken met child(g(x') = g(child), parent).

L10: Q: x(5,s); u(10,s)	Het algoritme voegt de mogelijke opvolgers toe aan Q op basis van prioriteit (= laagste kost) en past de kost om betreffend punt te bereiken aan (groen).
Iteratie 2 begint	
L8: opvolgers x(5,s): y(5+2,x) en v(5+9,x) en u(5+3,x)	Het algoritme berekent de kost van de opvolgers van het eerste element uit Q, namelijk x.
L10: Q: y(7,x); u(8,x); v(14,x)	Het algoritme voegt de mogelijke opvolgers aan Q toe op basis van prioriteit + past de kost om betreffend punt te bereiken aan (rood).
L12: update kost u	U stond al in Q en het algoritme past de kost om het punt te bereiken aan omdat het via een ander pad met lagere kost bereikt kan worden (rood).
	S en x zijn al via optimale paden bereikt aangezien ze bij de vorige iteraties als eerste staan. Tijdens de volgende iteraties vallen s en x buiten beschouwing.
Iteratie 3 begint	
L8: opvolgers y(7,x): v(7+6,y) en s(7+7,y)	Het algoritme berekent de kost van de opvolgers van het eerste element uit Q, namelijk y. S valt weg aangezien het de startpositie is.
L10: Q: u(8,x); v(13,y)	Het algoritme voegt de mogelijke opvolgers toe aan Q op basis van prioriteit (= laagste kost) en past de kost om betreffend punt te bereiken aan (blauw).
L12: update kost v	V stond al in Q en het algoritme past de kost om het punt te bereiken aan omdat het via een ander pad met lagere kost bereikt kan worden (blauw).
	Y is al via het optimaal pad bereikt aangezien het bij de vorige iteratie als eerste staat. Tijdens de volgende iteraties valt y buiten beschouwing.

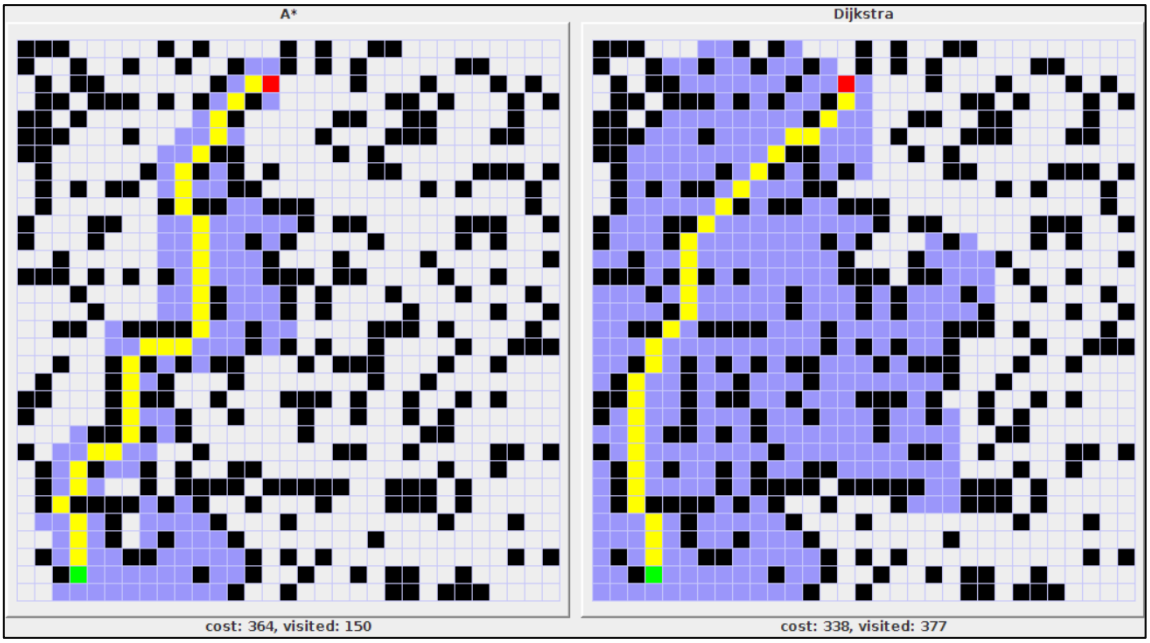
Iteratie 4 begint	
L8: opvolgers u(8,x): v(8+1,u) en x(8+2,u)	Het algoritme berekent de kost van de opvolgers van het eerste element uit Q, namelijk u.
L10: Q: v(9, u)	Het algoritme voegt de mogelijke opvolgers toe aan Q op basis van prioriteit (= laagste kost) en past de kost om betreffend punt te bereiken aan (roze).
L12: update kost v	V stond al in Q en het algoritme past de kost om het punt te bereiken aan omdat het via een ander pad met lagere kost bereikt kan worden (roze).
	U is al via het optimaal pad bereikt aangezien het bij de vorige iteratie als eerste staat. Tijdens de volgende iteraties valt u buiten beschouwing.
Iteratie 5 begint	
L8: opvolgers v(9,u): y(9+4,v)	Het algoritme berekent de kost van de opvolgers van het eerste element uit Q, namelijk v.
	Y werd reeds op een optimale manier bereikt en wordt niet toegevoegd aan Q.
	V is al via het optimaal pad bereikt aangezien het bij de vorige iteratie als eerst staat. Tijdens de volgende iteraties valt v buiten beschouwing.
Einde iteraties en algoritme	Optimale paden tussen alle punten gekend.

Nu heeft het algoritme de kleinste kost berekend om alle punten te bereiken vanuit de startpositie. Bij een gegeven doelpositie, is het kortste pad gekend.

2.2.4 A*

Zoals beschreven in [5] en [6] is A*[2] een uitbreiding op Dijkstra's zoekalgoritme dat het totaalaantal onderzochte statussen reduceert. Dit doet het door gebruik te maken van een heuristische schatting h(x) van de kost om het einddoel te bereiken vanuit de huidige status, ofwel de *cost-to-go* (kost om ergens heen te gaan). De werking van A* is identiek aan Dijkstra's algoritme met als enige verschil dat A* Q sorteert op basis van de som g(x') + h(x'). Indien h een onderschatting is van de totale kost om van de start tot de eindpositie te gaan, vindt A* gegarandeerd het optimaal pad. Deze onderschatting ligt liefst zo dicht mogelijk bij de werkelijkheid. Een mogelijke onderschatting is de kost in vogelvlucht naar het einddoel. Dit is een onderschatting aangezien het in rekening brengen van obstakels de totale kost sowieso verhoogt. Het algoritme is bovendien steeds systematisch. Doordat het algoritme rekening houdt met de lokale kost én zich richt tot het einddoel met behulp van de totale kost, wijkt het minder ver uit van het einddoel. Indien A* een punt onderzoekt waarvan de volgende stap een obstakel is, is de kost g(x') oneindig groot. Het algoritme vermijdt bijgevolg deze stap. Indien het algoritme te ver afwijkt van de doelpositie, vergroot de kost h(x') tot de doelpositie. Bijgevolg vermijdt A* deze richting. Het algoritme bouwt een pad rondom obstakels en in de richting van het doel aangezien de punten op dat pad een lage g('x) én h(x') hebben waardoor ze als eerst in Q staan.

Volgende Figuur 2.5 illustreert het verschil tussen Dijkstra's algoritme en A*. Dijkstra doorzoekt heel de omgeving totdat een pad gevonden is en A* doorzoekt de omgeving richting het doel totdat een pad gevonden is. Uit dit voorbeeld blijkt echter dat de kost voor het pad uit A* hoger is maar dat het aantal bezochte punten aanzienlijk minder is. De daling in het aantal bezochte punten vermindert de rekentijd.



Figuur 2.5: Verschil tussen het A algoritme (links) en Dijkstra's algoritme (rechts). Hierbij heeft A* een aanzienlijk minder aantal onderzochte posities waardoor de rekentijd afneemt maar de kost om het eindpunt te bereiken vergroot* [9].

[2] uitgesproken als: "A-ster"

In dynamische omgevingen kunnen onvoorziene obstakels op het pad van A* komen te liggen. Hierdoor moet A* terug vanaf het begin zoeken naar een pad rond deze obstakels tot het einddoel. Deze herberekening is echter computationeel zwaar. Het D* algoritme lost dit probleem in dynamische omgevingen op door bij het tegenkomen van een onvoorzien obstakel de kost van bepaalde delen uit het pad aan te passen [10]. Hierdoor kan D* het pad herplannen gebruik makend van een stuk van het reeds geplande pad. De A* en D* algoritmes worden echter computationeel veeleisend wanneer de zoekruimte groot is (door een hoge resolutie voor het probleem) of wanneer de zoekopdracht meerdere dimensies heeft (zoals bij het plannen van een arm met meerdere vrijheidsgraden) [10].

2.2.5 Iterative deepening

Zoals besproken in [6] is het *iterative deepening* (iteratief verdiepend) of meer specifiek het *iterative deepening depth-first* zoekalgoritme gunstig bij een boomstructuurverdeling met veel vertakkingen. Dit is het geval wanneer de volgende status veel meer mogelijke opvolgers heeft dan de huidige status. Het algoritme vertrekt van een *depth-first* zoekopdracht waarin het alle statussen vindt waarvan de afstand (= aantal stappen) tot x_i kleiner of gelijk is aan een gegeven waarde i. Indien het doel geen deel uitmaakt van deze statussen, laat het algoritme deze zoekopdracht buiten beschouwing. Vervolgens begint een nieuwe zoekopdracht naar punten met een afstand kleiner of gelijk aan i+1 tot de startpositie. Deze iteratie begint bij i = 1 en eindigt pas wanneer het doel gevonden is. Aangezien de lijst Q bestaat uit een boomstructuur met een grote vertakking, is het aantal gevonden statussen bij de iteratie i+1 veel malen groter (bv. factor 10) dan het aantal gevonden statussen bij iteratie i. Vandaar dat de kost van de punten uit iteratie i verwaarloosbaar is ten opzichte van de kost van de punten uit iteratie i+1. Op deze manier is dit algoritme een omvorming van het *depth-first* algoritme tot een systematisch algoritme. Dit voorkomt het nadeel van het *depth-first* algoritme waarbij het te snel en te langdurig een lang pad onderzoekt zonder andere opties in de breedte te beschouwen.

Een uitbreiding hierop is de combinatie met het A* algoritme tot het *iterative deepening A** (IDA) algoritme. Hierbij stelt i de som van de *cost-to-go* en de *cost-to-come* functie voor, namelijk g(x') + h(x'). Het algoritme verhoogt in dit geval tijdens elke iteratie de toelaatbare totale kost.

2.2.6 Best-first

Zoals beschreven in [6] is bij het *best-first* 'beste eerst' algoritme de prioriteitswachtlijst geordend op basis van een schatting van de optimale *cost-to-go*. De oplossing is hierdoor niet noodzakelijk optimaal waardoor het niet uitmaakt of de schatting de werkelijke optimale *cost-to-go* overschrijdt. Dit is echter wel een belangrijke voorwaarde bij A* om een optimale oplossing te vinden. Hoewel het algoritme geen optimale oplossingen vindt, onderzoekt het daarentegen veel minder punten waardoor de rekentijd kleiner is. De rekentijd is in het slechtste geval echter nog steeds groter dan die in het slechtste geval van A*. Bovendien is het algoritme niet systematisch en verspilt het in vergelijking met Dijkstra's algoritme nog meer onnodige stappen. Het *best-first* algoritme is vanwege de 'gulzigheid' van de geschatte *cost-to-go* bijgevolg geen ideaal algoritme.

2.2.7 Backward search

Elk besproken zoekalgoritme kan in de omgekeerde zin werken als een *backward search* [6]. Hierbij vertrekt het algoritme van het einddoel x_G en werkt het terug naar x_I. Deze methode is efficiënter wanneer er meer vertakkingen zijn vanaf de startpositie x_I. Een achterwaarts algoritme vermijdt de zoektocht doorheen deze vertakkingen aangezien het langs de andere kant komt. De structurele opbouw van het achterwaarts padplanningsalgoritme is analoog aan het voorwaartse. Het achterwaarts algoritme maakt daarentegen gebruik van een achterwaartse statustransformatiefunctie f^{-1} waarmee het x bekomt vanuit x' met behulp van de achterwaartse actie u^{-1}. De achterwaartse toestandstransitievergelijking beschrijft deze transformatie:

$$x' = f^{-1}(x', u^{-1}) \tag{2.4}$$

De functie f^{-1} stelt de puntenverdeling voor met elke verbinding tussen de punten omgekeerd in vergelijking met f. Hierdoor is het vinden van een oplossing met het achterwaarts algoritme hetzelfde als het vinden van een oplossing in de oorspronkelijke verdeling met het voorwaarts algoritme. Hierbij stelt elke actie u^{-1} een omgekeerde verbinding tussen twee punten voor.

2.2.8 Bidirectional search

Tot slot is het *bidirectional search* algoritme een combinatie van een voorwaarts en achterwaarts algoritme [6]. Hierbij begint het algoritme gelijktijdig vanuit de startpositie x_I met behulp van de voorwaartse variant en vanuit de doelpositie x_G met behulp van de achterwaartse variant. De zoekopdracht is een succes wanneer de twee zoekfronten samenkomen en faalt wanneer dit niet gebeurt. Deze methode reduceert bij veel toepassingen het aantal te onderzoeken punten. Zowel Dijkstra's algoritme als het A* algoritme hebben bidirectionele varianten die leiden tot optimale oplossingen.

2.2.9 Conclusie

Deze sectie besprak de manier waarop verschillende zoekalgoritmes paden zoeken doorheen het wereldmodel. Deze algoritmes berusten allemaal op hetzelfde principe van het voorwaarts padplanningsalgoritme met als enige verschil de manier waarop de algoritmes de wachtlijst Q van volgende statussen doorlopen. Zo sorteert het *breadth-first* algoritme de takken van een wereldmodel, opgebouwd uit boomstructuren, in de breedte. Het *depth-first* algoritme verdiept zich daarentegen eerst in een volledige tak alvorens het aan een andere tak begint. Vervolgens kent Dijkstra's algoritme kosten toe aan punten om hen te bereiken waardoor het Q sorteert op basis van een zo laag mogelijke kost. Op deze manier vindt het algoritme het pad met de laagste totale kost om het einddoel te bereiken in een wereldmodel waarin punten met elkaar verbonden zijn. Het A* algoritme is een verbetering van Dijkstra's algoritme. Het houdt namelijk bijkomend op de kost om een punt te bereiken, ook rekening met een schatting van de totale kost tot aan het einddoel. Op deze manier convergeert het algoritme richting het einddoel en onderzoekt het geen punten die niet in de richting van het doel liggen. Het *iterative deepening* algoritme is toepasbaar op wereldmodellen waarvan de boomstructuur veel vertakkingen heeft. Het is een verbetering van het *depth-first* algoritme waarbij de diepte waarmee het in een tak naar een oplossing mag zoeken, elke iteratie vergroot. Het IDA-algoritme breidt dit uit door de toelaatbare diepte te vervangen door de toelaatbare totale kost van A*. Verder maakt het *best-first* algoritme gebruik van een initiële

schatting van de *cost-to-come* functie. Dit algoritme levert echter geen optimale paden op. Al deze algoritmes kunnen als een achterwaartse variant voorkomen door de zoektocht te beginnen vanuit de eindpositie x_G en de acties tussen de punten om te keren. Tot slot maakt een bidirectionele versie van de besproken algoritmes tegelijk gebruik van zowel de voorwaartse als de achterwaartse versie om elkaar tegemoet te komen. Dit reduceert de te onderzoeken punten waardoor de rekentijd afneemt.

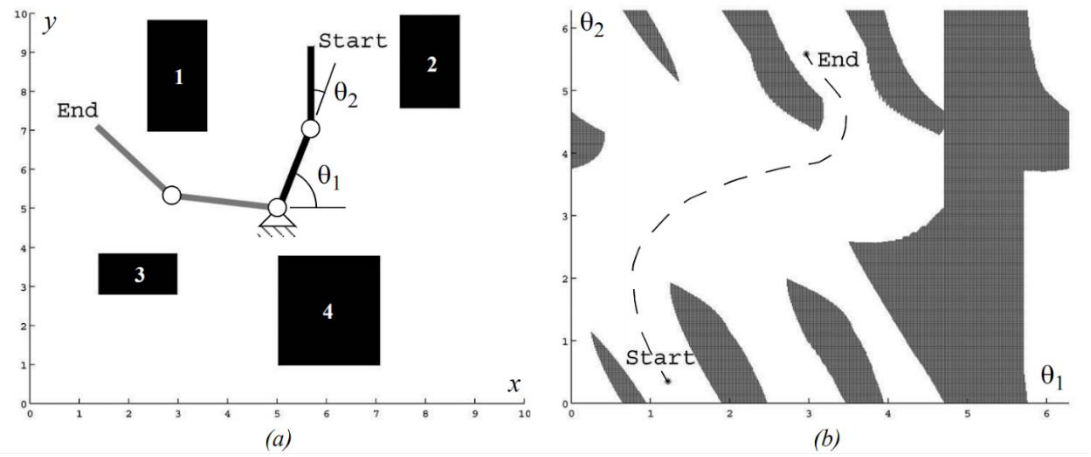
2.3 Kaarttransformaties

Deze sectie bespreekt de nodige stappen om een bestaande kaart van de omgeving te transformeren naar een discrete kaart die bruikbaar is voor het padplanningsalgoritme [11]. De eerste stap hierin is het vormen van de configuration space. In deze ruimte stelt een punt de configuratie van de robot voor. Vervolgens behandelt het deel over benaderende en exacte padplanners het verschil tussen deze twee methodes en hun voor- en nadelen. Tot slot bespreken de delen over *road maps*, *cell decomposition, Rapidly Exploring Dense Trees* en *potential field* elk een verschillende soort maptransformatie.

2.3.1 Configuratieruimte

De eerste stap bij padplanning is het opstellen van een configuratieruimte of *C-space*. Volgens [11] is de C-space een ruimte waarin elke mogelijke configuratie van een robot of robotarm voorgesteld is door een punt. Indien een robot *k* vrijheidsgraden heeft, stellen *k* aantal toestanden: q_1, …, q_k de configuratie voor. Een ruimte waarin een punt al deze toestanden beschrijft moet dus *k* dimensionaal zijn. Door de configuraties waarbij de robot in collisie is met een object in de fysieke ruimte voor te stellen als objecten in de C-space kan de volledige kaart getransformeerd worden naar de C-space.

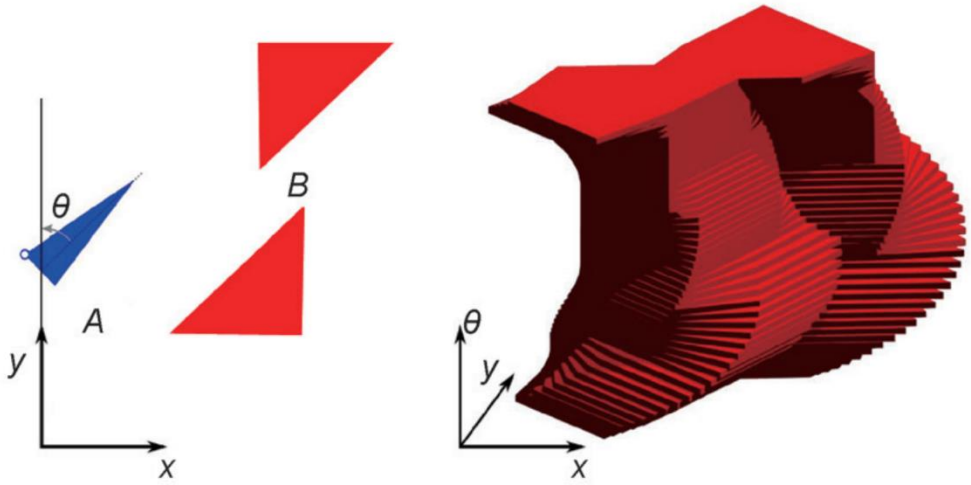
Onderstaande Figuur 2.6 toont de fysieke ruimte (a) en de C-space (b) voor robotarm met twee vrijheidsgraden (hier de hoeken van de armgewrichten) [11]. Doordat een punt de volledige configuratie van de robot weergeeft is het veel eenvoudiger om in de C-space een pad te plannen. Wanneer het pad binnen de C-space van start naar einde wordt afgelopen neemt de robot elke configuratie aan van de tussenliggende punten op het pad.



Figuur 2.6: (a) De fysieke ruimte met daarin de robotarm in de start configuratie en de eind configuratie. (b) De configuratieruimte van de robotarm met daarin het pad gepland van start tot einde waarbij de configuratie de hoekinstellingen van de twee gewrichten voorstelt. De C-space stelt de configuratie van de robotarm voor als een punt.

Hierbij stellen de grijze gebieden in de C-space de configuraties van de robotarm voor waarbij deze in botsing is met de zwarte obstakels uit de werkomgeving [11, p. 372].

Onderstaande Figuur 2.7 stelt de C-space voor voor een differentieel aangedreven mobiele robot [12]. De blauwe driehoek stelt de robot voor uitgerust met een eigen assenstelsel. De positie (x,y) van het robotassenstelsel in het wereldassenstelsel en de hoek (**θ**) tussen het robot- en wereldassenstelsel beschrijft de configuratie (hier pose = positie en oriëntatie) van de robot.



Figuur 2.7: (links) De fysieke wereld waarin de blauwe driehoek een differentieel aangedreven robot voorstelt en de rode driehoeken onbeweegbare objecten. (rechts) De configuratieruimte waarbij het rode object het obstakel getransformeerd naar de configuratieruimte voorstelt [12, p. 48].

De configuration space wordt opgebouwd door de robot langs de objecten te verplaatsen. Posities van het robotassenstelsel waarbij de robot in contact staat met een object stellen de grenzen van het object in de C-space voor. Aangezien de robot drie vrijheidsgraden heeft (x, y en **θ**) is de C-space driedimensionaal. Hierbij stelt een beweging volgens de verticale as een rotatie voor [12].

Door de transformatie naar een C-space stelt slechts één punt de volledige configuratie van de robot voor op die pose. Hierdoor moet de padplanning geen rekening houden met de geometrie van de robot. De transformatie van de werkruimte naar de C-space is echter rekenintensief waardoor het niet praktisch is [5].

2.3.2 Benaderende en exacte planners

De volgende stap in de kaarttransformatie is de overgang van een continue kaart naar een discrete kaart. De discretisatie is afhankelijk van de kaarttransformaties en indeelbaar in twee groepen: de benaderende en de exacte methoden. De benaderende methode beschrijft de C-space benaderend. Hierdoor bestaat de mogelijkheid dat de padplanner niet compleet is. Een padplanner is compleet wanneer deze in staat is elk mogelijk pad te vinden in een eindige tijd [5]. Exacte of combinatorische planners beschrijven de C-space exact [6].

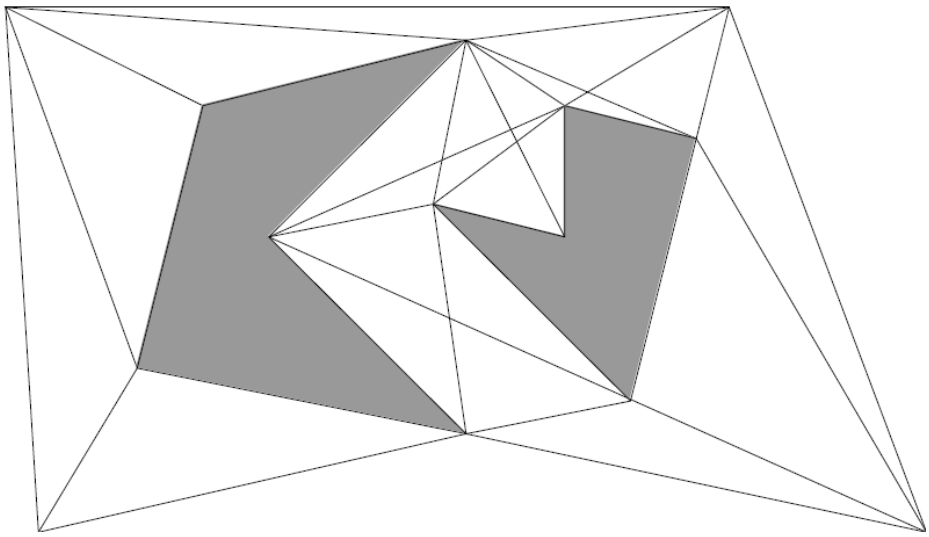
Volgens [6] bestaat er binnen de benaderende kaarttransformaties een veelgebruikte, efficiënte groep genaamd *sample-based motion planning*. Deze groep benadert de ruimte door deze te bemonsteren met punten. Nadien gaat een *collision-module* na of deze punten al dan niet in aanraking zijn met objecten in de C-space. Indien deze punten in aanraking zijn met de objecten

sluit de module deze uit. Op deze manier benaderen de overgebleven punten de beschikbare ruimte in de C-space. Padplanners binnen deze groep verschillen in de manier waarop ze de C-space bemonsteren. De *Open Motion Planning Library* (OMPL) is een veelgebruikte open-source bibliotheek die verschillende van dergelijke samplegebaseerde padplanningsalgoritmes bevat en een variatie aan bemonsteringsmethoden toelaat [13].

2.3.3 Road maps

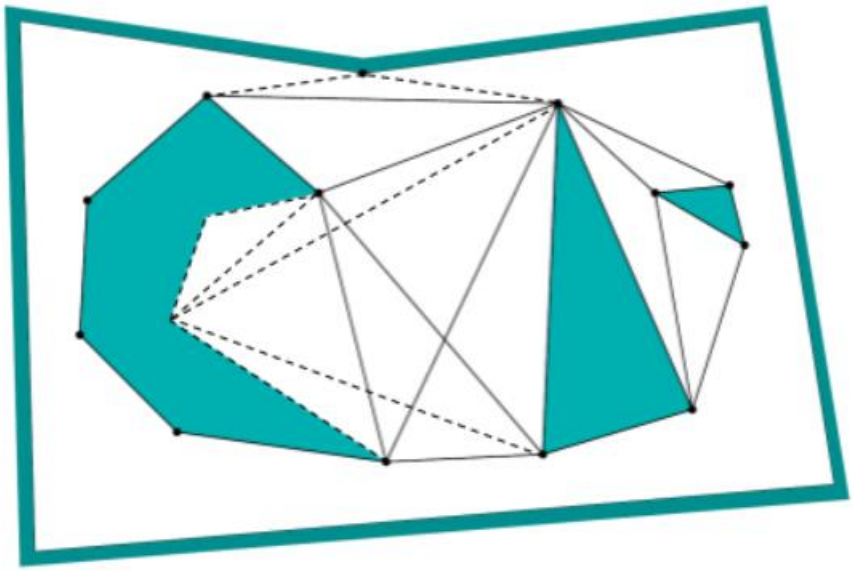
Volgens [14] is het doel van de *road maps* maptransformatie de reductie van een N-dimensionale C-space naar een groep van ééndimensionale paden. Deze groep van ééndimensionale paden vormt een netwerk waarin een graph search algoritme een gewenste oplossing zoekt (zie deel 2.2 zoekalgoritmes). Er bestaan verschillende oplossingsvoorwaarden zoals: kortste, snelste of meest energie-efficiënt pad. Deze voorwaarden geven niet noodzakelijk hetzelfde pad. Hieronder volgen padplanmethoden die elk de C-space transformeren naar een roadmap.

Onderstaande Figuur 2.8 geeft de *visibility graph* methode weer. Deze methode vormt een roadmap door alle hoekpunten van objecten in de C-space met elkaar te verbinden. Volgens [6] verbindt deze methode enkel hoekpunten wanneer de verbindingslijn tussen deze twee punten geen objecten in de C-space snijdt.



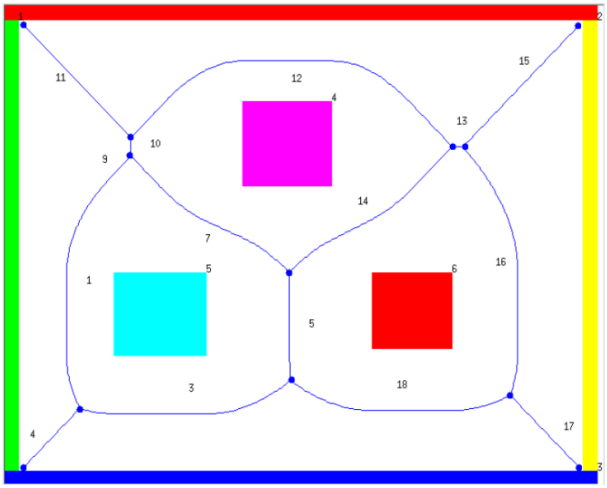
Figuur 2.8: Visibility graph van een omgeving met twee obstakels waarbij het verbinden van de hoekpunten van de obstakels een road map vormt [14, p. 7].

Er bestaat volgens [6] ook een variatie met een extra voorwaarde. Deze kortste pad methode verbindt enkel hoekpunten indien het verlengde van de verbindende lijn het object waartoe het hoekpunt behoort niet snijdt. Onderstaande Figuur 2.9 toont een voorbeeld van deze variatie. Hierbij zijn de streeplijnen lijnen die niet langer voldoen omwille van deze bijkomende voorwaarde.



Figuur 2.9: Kortste pad road map waarbij de streeplijnen niet voldoen aan de bijkomende voorwaarde dat hun verlengde niet mag snijden met een obstakel [6, p. 262].

Visibility graphs zijn complete maptransformaties. Hierdoor zijn de padplanners in staat alle mogelijke paden te vinden. Indien er geen pad bestaat, meldt de planner dit binnen een eindige tijd [5]. Bovendien vindt deze methode het kortste pad door vlak langs objecten te bewegen. Hierdoor moeten de objecten in de C-space iets groter voorgesteld worden dan in de werkelijkheid om botsing tussen robot en object te vermijden [6]. Bovendien zit er een onzekerheid op de beweging van een robot waardoor de kans op botsing met een object groot is wanneer deze dicht bij een object beweegt.



Figuur 2.10: Voronoi diagram waarbij de blauwe paden equidistant zijn van minstens twee objecten [14, p. 8]

Een voronoi diagram, zoals te zien in Figuur 2.10, tracht dit probleem op te lossen door zover mogelijk van objecten in de C-space te blijven. Elk punt op de blauwe paden is equidistant van minstens twee objecten. Hierdoor blijft de robot zover mogelijk van de objecten verwijderd [14]. Een nadeel is dat de paden hierdoor langer zijn.

Verschillend van bovenstaande methodes, is de probabilistische roadmap (PRM) een *multiple-query sample* gebaseerde kaarttransformatie. Bij een single-query methode is er vanaf de start een begin- en eindpunt opgegeven. De single-query kaarttransformatie vormt dan een pad rekening houdend met deze twee punten. De multiple-query kaarttransformatie bestaat uit twee fasen [6]:

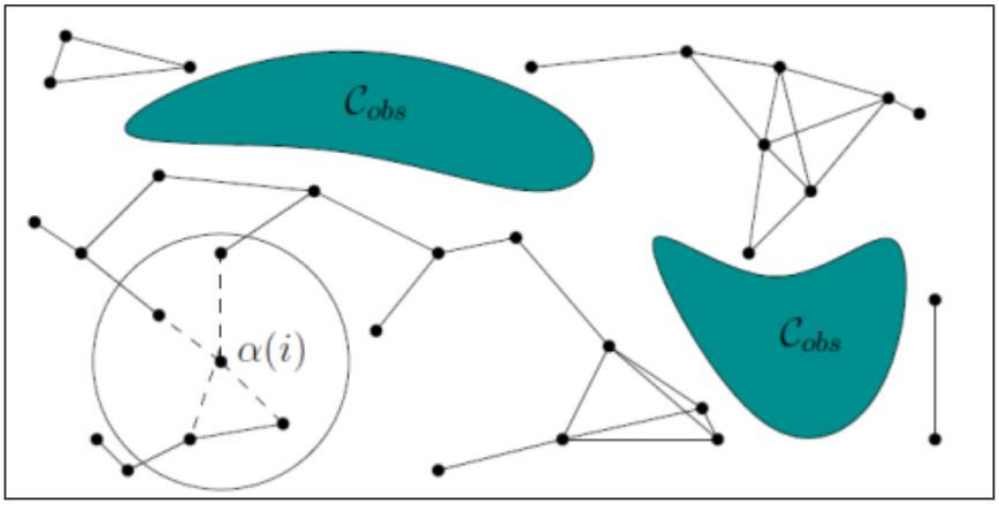
- De *preprocessing fase*: Deze bouwt een kaart op van de beschikbare ruimte in de C-space zonder rekening te houden met een start of doel positie.
- De *query fase*: Vanaf deze fase zijn de start en doel positie gekend. Hierbij gaat een lokale planner elk van deze punten proberen te verbinden aan het eerder gevormd netwerk.

De pseudocode in onderstaande Figuur 2.11 stelt de opbouw van de roadmap voor tijdens de preprocessing fase. Het resultaat van het doorlopen van de pseudo-code staat weergegeven in onderstaande Figuur 2.12 [6].

```
BUILD_ROADMAP
1  G.init(); i ← 0;
2  while i < N
3      if α(i) ∈ C_free then
4          G.add_vertex(α(i)); i ← i + 1;
5          for each q ∈ NEIGHBORHOOD(α(i),G)
6              if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
7                  G.add_edge(α(i), q);
```

Figuur 2.11: Pseudocode voor de opbouw van een road map met N aantal knooppunten [6, p. 238].

Regel 1 van de pseudo-code initialiseert de opbouw van de roadmap. Hierbij wordt het aantal toegevoegde knooppunten i gelijkgesteld aan 0. De volgende regel controleert of het aantal toegevoegde knooppunten kleiner is dan het gewenst aantal knooppunten. Zodra i = N zijn er evenveel knooppunten toegevoegd aan de roadmap als het gewenst aantal punten en stopt het algoritme. Indien i < N gaat het algoritme verder op regel 3. Deze regel gaat na of het nieuwe gesamplede punt α(i) binnen de vrije ruimte van de configuration space valt. Indien dit waar is, voegt regel 4 het knooppunt toe aan de roadmap en wordt het aantal toegevoegde punten i verhoogd. De functie NEIGHBORHOOD zoekt alle punten in de omgeving van α(i). De volgende regels gaan na of de reeds toegevoegde knooppunten, die deel uitmaken van de omgeving van het gesamplede punt, verschillend zijn van het punt en of ze verbonden kunnen worden met dit punt. Indien aan deze twee voorwaarden voldaan is, verbindt de laatste regel de punten in de omgeving met het laatst toegevoegd punt α(i).



Figuur 2.12: Roadmap opgebouwd door pseudo-code uit Figuur 2.11. Hierbij stelt α(i) een nieuwe sample in de C-space voor waarbij het algoritme alle punten gelegen binnen de cirkel met α(i) verbindt [6, p. 238].

Er bestaan verschillende manieren om de punten in de omgeving te bepalen. Het onderstaande deel beschrijft drie mogelijke manieren zoals beschreven in [6].

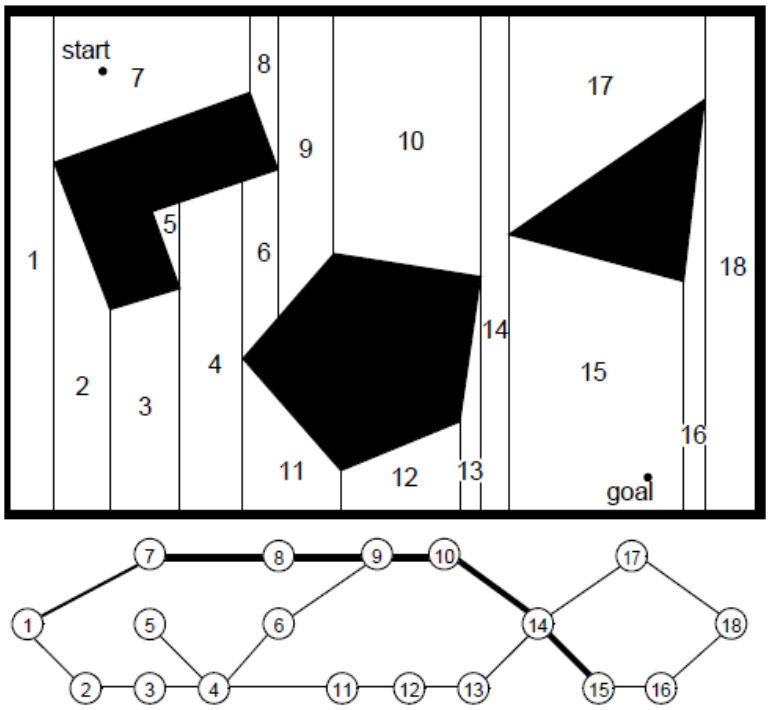
1. Straal: Selecteert alle punten die zich binnen een straal vanaf het laatst toegevoegd punt bevinden. Hierbij begrenst een parameter K het maximaal aantal verbindingen. Een andere methode voor het begrenzen van het aantal verbindingen is de straal laten afnemen wanneer het aantal geselecteerde punten toeneemt.
2. Zichtbaarheid: Deze methode selecteert alle punten die zichtbaar zijn voor het laatst toegevoegde punt.
3. Dichtstbijzijnde K: Deze methode selecteert het K-aantal dichtstbijzijnde punten.

De query fase maakt gebruik van hetzelfde algoritme. In deze fase volgen slechts twee nieuwe iteraties. Een eerste waarbij de startpositie α(i) vervangt en een tweede iteratie waarbij de eindpositie α(i) vervangt. Op deze manier verbindt de query fase de start- en eindpositie met de roadmap [6]. De PRM maakt deel uit van OMPL [13].

2.3.4 Cell decomposition

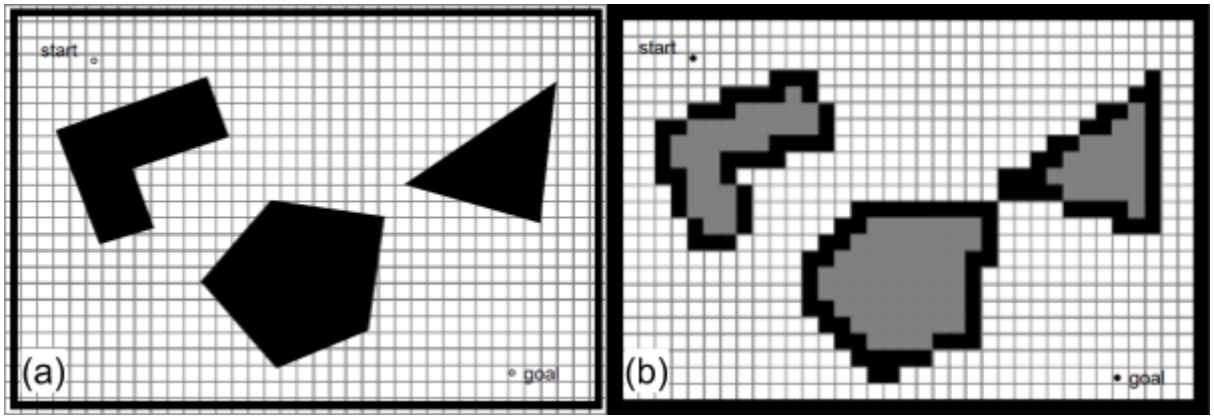
Cel decompositie voert een kaarttranfsormatie uit die de continue ruimte opdeelt in cellen. Deze cellen zijn herleidbaar tot een roadmap waarbij de vrije cellen knooppunten voorstellen en de aangrenzende cellen met elkaar verbonden zijn. Dit hoofdstuk bespreekt verschillende manieren waarop deze celindeling plaatsvindt. De eerste manier is een exacte cel decompositie. Het tweede onderdeel behandelt een benaderende cel decompositie die de ruimte indeelt in een grid. Vervolgens bespreekt dit onderdeel een variant waarbij de celgrootte veranderlijk is. Ten slotte bespreekt het laatste onderdeel en cel decompositiemethode voor een driedimensionale ruimte.

Exacte cel decompositie deelt de ruimte in cellen op die volledig bezet of vrij zijn. Dit is weergegeven op onderstaande Figuur 2.13. Hierbij zijn de cellen ingedeeld volgens de horizontale coördinaten van de obstakelhoekpunten [6].



Figuur 2.13: (bovenaan) Celindeling op basis van de verbinding van horizontale coördinaten van de obstakelhoekpunten. (onderaan) De road map gevisualiseerd door de verbinding van aangrenzende cellen [11, p. 377].

Een exacte cel decompositie is in werkelijkheid niet altijd haalbaar wegens een gebrek aan volledigheid van ruimtelijke informatie. Een veel gebruikte alternatieve methode is de *fixed decomposition* of vaste decompositie [11]. Deze benadert de ruimte op een discrete manier waarbij het de ruimte indeelt in een raster met vaste afmetingen zoals te zien op onderstaande figuur 14 (b). In deze Figuur 2.14 stellen de zwarte cellen, cellen voor waarin zich een object bevindt. Hiermee benadert de kaarttransformatie de objecten uit de continue ruimte. De witte cellen stellen cellen voor waarin er enkel vrije ruimte is. Dit soort kaart stelt een *occupancy grid* voor (of bezettingskaart) [11].

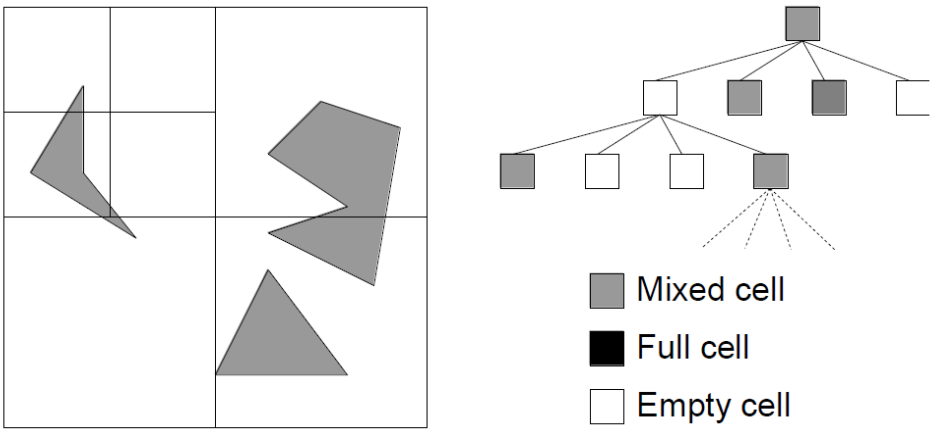


Figuur 2.14:(a) Continue ruimte waarover een raster gelegd is. (b) occupancy grid waarbij alle cellen waarin zich een gedeelte van een object bevindt, zich als object gedragen. Enkel cellen die volledig vrije ruimte bezitten, blijven vrij. Door te grote celafmetingen gaan er details zoals smalle gangen verloren [11, p. 289].

Deze kaarttransformatie heeft een aantal nadelen. Ten eerste zorgen relatief grote cellen voor een verlies van details, zoals te zien in Figuur 2.14 (b). Een mogelijke oplossing hiervoor is het

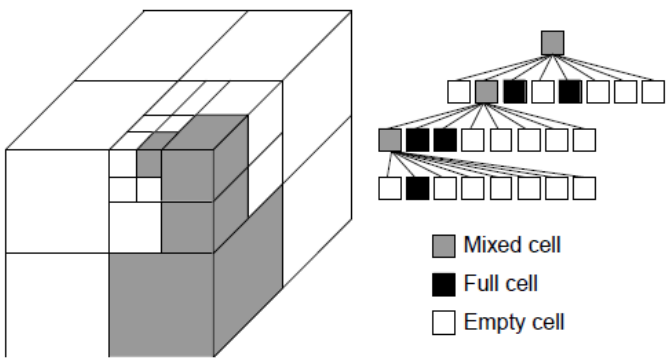
verminderen van de celgrootte. De kleinere celgrootte zorgt echter voor een toename van het nodige geheugen om de kaart op te slaan. Bovendien is het nodige opslaggeheugen sterk afhankelijk van de grootte van de ruimte. Een grote ruimte benaderd met kleine cellen heeft dus een groot opslaggeheugen nodig [11].

Variabele cel decompositie probeert hier een oplossing voor te bieden door gebruik te maken van een variabele celgrootte. Deze methode gaat na of een cel volledig leeg, volledig vol of gedeeltelijk vol is. Indien de cel gedeeltelijk bezet is door een object splitst deze methode de cel op in dochtercellen. Deze opsplitsing is afhankelijk van het aantal dimensies van de kaart. Bij een tweedimensionale kaart wordt één cel in vier deelcellen gesplitst. Een boomstructuur genaamd *quadtree* stelt deze opsplitsing voor tweedimensionale ruimtes. Figuur 2.15 toont deze opsplitsing en de boomstructuur. Dit is een iteratief proces: de methode gaat na iedere splitsing opnieuw na welke cellen deels bezet zijn en splitst ze dan weer op. Deze iteratie loopt totdat de kleinste celgrootte bereikt is of tot wanneer er enkel nog volledig bezette of lege cellen overblijven [14].



Figuur 2.15: (links) Variabele celdecompositie in tweedimensionale ruimte waarbij de methode de deels bezetten cellen in vier splitst. (rechts) Splitsing voorgesteld door quadtree. Deze splitsing loopt door totdat er geen deels bezetten cellen overblijven of totdat de minimale celgrootte bereikt is [14, p. 11].

Bij driedimensionale ruimtes is er een opdeling in acht subcellen, zoals te zien op onderstaande Figuur 2.16. De boomstructuur verkregen uit de opsplitsing van de driedimensionale ruimte heet een *octree* [14].



Figuur 2.16: (links) Variabele celdecompositie in driedimensionale ruimte waarbij deze methode de deels bezetten cellen in acht splitst. (rechts) Splitsing voorgesteld door octree. Deze splitsing loopt door tot wanneer er geen deels bezetten cellen meer zijn of de minimale celgrootte bereikt is [14, p. 11].

Door een variabele celgrootte te gebruiken, benadert deze methode de ruimte met minder cellen zonder verlies aan informatie door te grote cellen. Hierdoor is er minder geheugen nodig voor een gedetailleerde kaart in vergelijking met cel decompositie met vaste celgrootte [14].

2.3.5 Rapidly Exploring Dense Trees

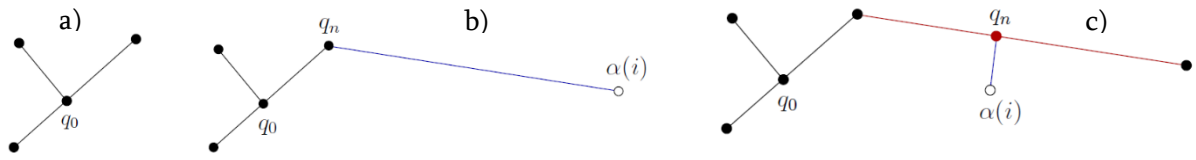
Dit deel behandelt *Rapidly exploring Dense Tree* of RDT padplanners. RDT is verschillend van bovenstaande kaarttransformaties doordat het niet enkel de continue ruimte transformeert naar een discrete kaart, maar ook tegelijkertijd een pad zoekt in deze kaart. De boomstructuur tot aan het einddoel is een mogelijk pad. Deze methode bouwt een netwerk incrementeel op waarbij de resolutie van het netwerk gradueel verbetert. Onderstaande Figuur 2.17 toont de pseudocode van een *Rapidly Exploring Dense Tree*. RDT is een overkoepelende term voor padplanningsalgoritmes die een boomstructuur opbouwen. RRT is dus een vorm van RDT waarbij de sampling van punten in de C-space op een willekeurige manier gebeurt. RRT is bijgevolg een sampling-based padplanningsmethode die de C-space benadert. De benadering van de ruimte verbetert bij een toenemend aantal iteraties. RRT maakt deel uit van OMPL [13].

```
SIMPLE_RDT(q_0)
1  G.init(q_0);
2  for i = 1 to k do
3      G.add_vertex(α(i));
4      q_n ← NEAREST(S(G), α(i));
5      G.add_edge(q_n, α(i));
```

Figuur 2.17: Pseudocode van een RDT-algoritme voor een C-space zonder objecten waarbij het algoritme een gesampeld punt α(i) verbindt met het dichtstbijzijnde punt om het netwerk uit te breiden. Bij RRT zorgt α(i) voor een willekeurige sampling van de C-space [6, p. 229].

Regel 1 initialiseert de opbouw van de RRT op startpositie q_o. De code in regel 2 herhaalt de onderstaande code totdat het een gewenst aantal knooppunten bereikt. Regel drie voegt een random gesampled punt α(i) als knooppunt toe aan het netwerk. Regel 4 zoekt naar het dichtstbijzijnde punt van het netwerk en noemt dit punt q_n. Tenslotte verbindt regel 5 het laatst toegevoegde knooppunt met het dichtstbijzijnde punt q_n [6].

Onderstaande Figuur 2.18 toont een mogelijk resultaat van de RRT padplanningsmethode. Op figuur 18 (b) is het dichtstbijzijnde punt vanuit α(i) een bestaand knooppunt. Op Figuur 2.18 (c) is dit punt een element van een verbindende lijn. In dit geval wordt er een knooppunt aan het lijnstuk toevoegt. Dit knooppunt splits het bestaande lijnstuk waardoor er een extra lijnstuk ontstaat [6].



Figuur 2.18: Mogelijk resultaat van de RRT padplanningsmethode met (a) Een bestaand netwerk. (b) Een nieuw toegevoegd punt α(i) waarvan het dichtstbijzijnde punt een bestaand knooppunt is. (c) Het dichtstbijzijnde punt is een element van een lijnstuk van het netwerk en het algoritme creëert een nieuw knooppunt q_n die het lijnstuk opsplitst [6, p. 229].

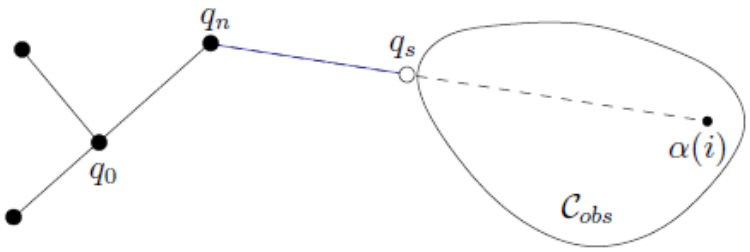
Onderstaande Figuur 2.19 toont de pseudocode van het algoritme dat rekening houdt met objecten in de C-space.

```
RDT(q_0)
1  G.init(q_0);
2  for i = 1 to k do
3      q_n ← NEAREST(S, α(i));
4      q_s ← STOPPING-CONFIGURATION(q_n, α(i));
5      if q_s ≠ q_n then
6          G.add_vertex(q_s);
7          G.add_edge(q_n, q_s);
```

Figuur 2.19: Pseudocode RDT voor een C-space met obstakels waarbij het met de functie Stopping- configuration een gesampeld punt, gelegen in een obstakel, zo dicht mogelijk bij het obstakel plaatst [6, p. 232].

Deze code verschilt weinig met de code uit Figuur 2.17 (simple_RDT). Deze code zoekt eerst naar het dichtstbijzijnde punt q_n van het netwerk vanuit α(i). Nadien bepaalt de functie *STOPPING-CONFIGURATION* het punt q_s dat zo dicht mogelijk bij de grens van het object ligt. Indien er tussen q_n en α(i) geen object aanwezig is geldt $q_s = \alpha$(i). Daarna controleert regel 5 of q_s verschillend is van q_n. Indien aan deze voorwaarde voldaan is voegt de volgende regel q_s als knooppunt toe aan het netwerk. Tenslotte verbindt de laatste regel q_s met q_n

Onderstaande Figuur 2.20 toont het resultaat van het algoritme dat beschreven staat in Figuur 2.19. Hierin stelt C_{obs} het gebied voor van een obstakel in de C-space.



Figuur 2.20: Netwerk opgebouwd door RDT-algoritme met obstakel in de C-space en punt q_s zo zicht mogelijk bij de grens van het obstakel ter vervanging van $\alpha(i)$ [6, p. 231].

Om een verbinding met de doelpositie te bekomen is er bij elke iteratie een kans dat α(i) gelijkgesteld wordt aan de doelpositie q_g. Indien α(i) = q_g forceert dit het algoritme om een verbinding met de doelpositie te maken.

Er bestaan tal van varianten op de RDT-padplanningsmethode. Zo bestaan er bi directionele-RDT-planners die vanuit zowel de start als doelpositie een boom vormen. Deze methode gaat ervan uit dat een golffront vertrekkende vanuit de startpositie meer oppervlakte bestrijkt om de eindpositie te bereiken dan twee golffronten (waarvan er een vertrekt vanuit de startpositie en de andere van de eindpositie) [6].

2.3.6 Potential field

Een laatste padplanningsmethode is de *potential field* aanpak, beschreven op basis van [5]. Bij deze methode is de robot (of robotarm) onderhevig aan een artificieel potentieel veld U waardoor de robot zich voortbeweegt volgens het veld. Het veld is zo opgebouwd dat het einddoel zich gedraagt

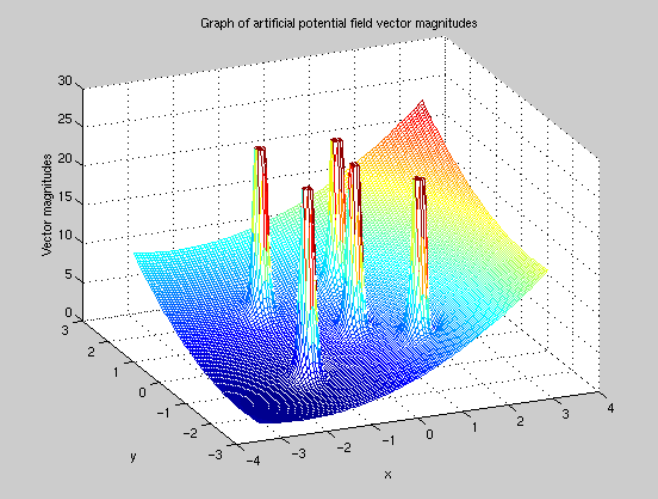
als een aantrekkende kracht en obstakels als afstotende krachten. De methode integreert nieuwe obstakels bij het updaten van U. De robot reageert op de superpositie van al deze krachten om op die manier het einddoel te bereiken zonder tegen obstakels te botsen. Deze methode waarvan verschillende varianten bestaan, is oorspronkelijk bedoeld voor robotarmen maar is ook toepasbaar op mobiele robots. De potential field methode is bovendien ook een controlewet voor het gedrag van de robot: de robot kan zijn volgende actie bepalen op basis van zijn positie eender waar in het veld.

Figuur 2.21 stelt een voorbeeld voor van een potential field weergave van een tweedimensionale omgeving voor een puntvormige robot met positie p(x,y). Hierbij is de robot op elke positie onderhevig aan het relatief artificieel veld F(q), waarbij:

$$F(q) = -\nabla U(q) \tag{2.5}$$

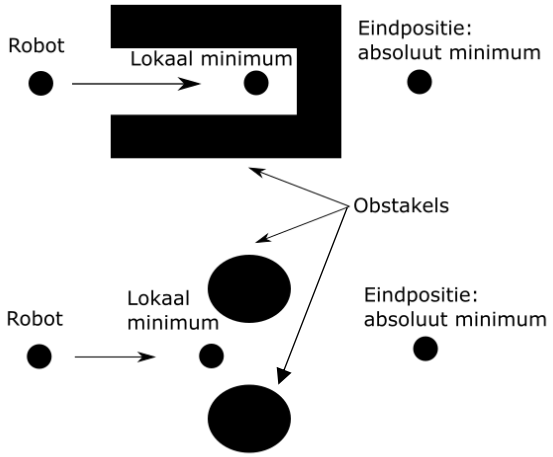
Waarbij ∇U(q) de gradiënt voorstelt van U op het punt q. De som van het aantrekkingsveld (de golvende vorm in de figuur) en het afstotend veld (de vijf pieken in de figuur) vormt het totaal potential field:

$$F(q) = F_{aantrekking}(q) - F_{afstoting}(q) \tag{2.6}$$



Figuur 2.21: Voorbeeld van potential field weergave van een tweedimensionale omgeving. Hierbij stelt het dalend vlak de aantrekking tot het einddoel voor en de vijf pieken de afstoting van de obstakels [15].

Het grootste voordeel aan deze methode is de makkelijke implementatie ervan, mits de afstemming van parameters. Het heeft echter hoofdzakelijk twee nadelen. Een eerste nadeel doet zich voor wanneer de robot zich bevindt in een lokaal minimum in het potentieel veld. Zoals te zien Figuur 2.22geraakt de robot vast te zitten in concave obstakels (bovenaan figuur) of voor obstakels met een even grote maar tegengestelde afstotingskracht (onderaan figuur). De robot zit vast aangezien het langs de ene kant de aantrekking voelt van het doel achter het obstakel en aan de andere kant de afstoting voelt van het obstakel. De robot ervaart deze positie als een minimum terwijl het uiterste minimum het einddoel is.



Figuur 2.22: Voorstelling van het lokaal minimaprobleem bij de potential field padplanningsmethode. Hierbij zit de robot vast in een lokaal minimum door enerzijds de aantrekking van het doel en anderzijds de afstoting van een cavitatie (boven) of twee obstakels met een tegengestelde even grote afstoting.

Een tweede nadeel komt voor wanneer de robot zich voortbeweegt in een smalle gang zoals een hal of een deuropening. De robot voelt in dit geval om de beurt de grotere afstoting van de ene of de andere wand waardoor het een oscillerende beweging maakt doorheen de gang.

2.3.7 Conclusie

Kaarttransformatie is een cruciale stap waarbij verschillende methoden een continue kaart transformeren naar een discrete kaart. Binnen deze kaart kan een zoekalgoritme, zoals beschreven in **2.2**, een pad kan zoeken. Deze transformatie kan benaderend of exact zijn. De exacte transformaties kosten meer rekentijd, maar zijn wel compleet. Hierdoor kunnen deze complete methoden in een eindige tijd aangeven of er een mogelijk pad is. De benaderende transformaties vergen minder rekentijd, maar kunnen omwille van de benaderende aanpak details van de ruimte, zoals smalle gangen, missen. Hierdoor zijn deze transformaties mogelijks incompleet. Exacte transformaties zoals visibility graph, voronoi diagram en exacte celdecomposite zijn omwille van de complexiteit van de achterliggende algortimes moeilijk implementeerbaar. PRM is een methode die goed toepasbaar is in een ruimte waarin meerdere padplanningsproblemen opgelost moeten worden. Variabele celdecompositie is een veelgebruikte methode doordat het instaat is een gedetailleerde kaart op te bouwen zonder gebruik te maken van een groot aantal kleine cellen die het opslaggeheugen onnodig verhogen. Vervolgens zijn potential field en RDT verschillend van de reeds eerder besproken kaarttransformaties doordat ze niet enkel de kaart transformeren maar er ook een

pad in plannen. RDT is een veelgebruikte techniek doordat deze snel implementeerbaar is. Ten slotte is potential field een gemakkelijk implementeerbare techniek. Deze is echter in de praktijk minder praktisch door nadelen zoals lokale minima en oscillaties.

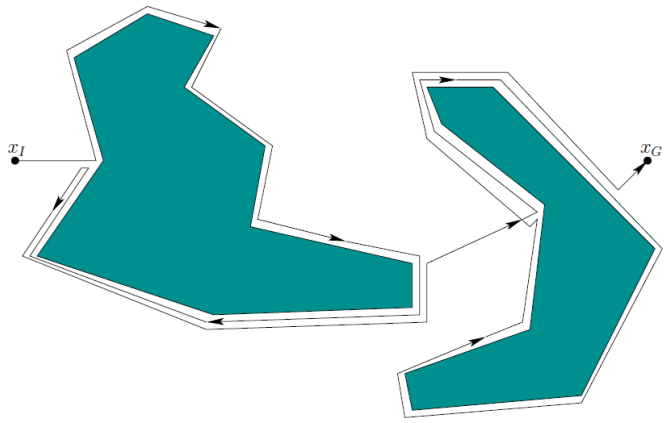
2.4 Obstakelvermijding

Zoals reeds besproken is het de taak van de obstakelvermijding (of lokale planner) om het pad aan te passen op basis van sensordata van onvoorziene obstakels. Op deze manier moet de planner de voorstelling van de omgeving niet updaten en geen nieuwe berekening van het volledig pad uitvoeren die veel tijd in beslag zou nemen [5].

2.4.1 Bug algoritme

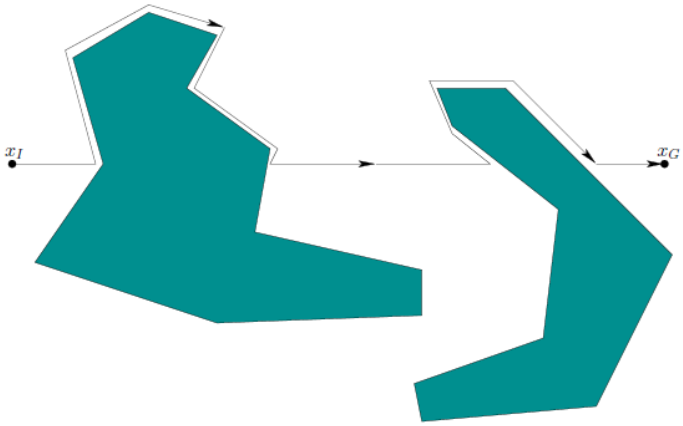
Zoals beschreven in [5], [6] is het bug algoritme een van de meest eenvoudige manieren om onvoorziene obstakels te vermijden. Hierbij volgt de robot de contouren van de obstakels gelegen op het pad om het einddoel te bereiken. De robot doet voor de obstakelvermijding beroep op de sensoren die twee taken uitvoeren [6]. Enerzijds geeft een sensor de euclidisch kortste afstand aan tot het doel en de richting tot het doel. Anderzijds voorziet een sensor de vorm van de rand van een obstakel vanop een kleine afstand ervan. Hiervoor moet de robot (bijna) in contact zijn met het object opdat de sensor nuttige data levert. Een nadeel van deze methode is dat het enkel rekening houdt met de meest recente sensordata waardoor het enerzijds ruisgevoelig is en waardoor deze data anderzijds onvoldoende kan zijn. Hoewel probabilistische planners dit probleem kunnen oplossen, vallen ze wegens hun verdere complexiteit buiten het bestek van deze literatuurstudie. Een tweede nadeel is dat het bug algoritme geen rekening houdt met de kinematica van de robot aangezien het de robot voorstelt als een punt [11].

Figuur 2.23 toont een eerste variant van het bug algoritme. Zoals beschreven in [6] beweegt de robot bij deze bug 1 strategie zich voort naar het doel en komt tot stilstand wanneer het een obstakel tegenkomt. Hierna roteert de robot naar links en volgt het de contour van het obstakel. Wanneer de robot de volledige contour heeft gevolgd tot aan het beginpunt van het obstakel, keert hij terug naar het punt op de contour waar de afstand tot het einddoel het kortst is. Tot slot beweegt de robot vanuit dit punt weer verder naar het doel. Het algoritme moet met behulp van een markering van de afgelegde weg of met behulp van een herkenning van herhalende sensordata achterhalen wanneer de robot de volledige contour gevolgd heeft.

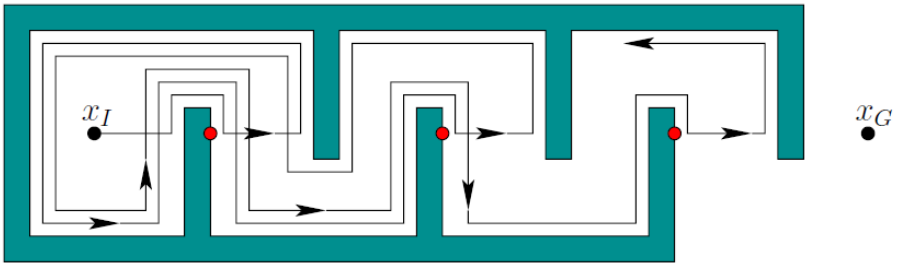


Figuur 2.23: Het bug 1 algoritme waarmee de robot een obstakel vermijdt om het doel te bereiken. Hierbij volgt de robot de volledige contour van het obstakel waarna het terugkeert naar het punt op de contour waar de afstand tot het doel het kleinst is. Vanuit dit punt vertrekt de robot verder naar het doel [6, p. 668] .

Figuur 2.24 toont een verbetering van het bug 1 algoritme. Zoals beschreven in [6] volgt de robot zich bij deze bug 2 strategie nog steeds op dezelfde manier de contour van een obstakel. Hier verlaat de robot de contour echter eerder. De robot volgt initieel een rechtlijnig pad tussen zijn eigen positie en het einddoel totdat de robot een obstakel tegenkomt. De robot stopt met het volgen van de contour wanneer het zich opnieuw op deze rechte lijn bevindt. Vanuit dit punt verlaat de robot de contour en volgt hij opnieuw de lijn om het doel te bereiken. Zoals te zien in Figuur 2.25 vereist deze aanpak een extra voorwaarde om oneindige cycli te voorkomen. De robot onthoudt namelijk de afstand tot het doel vanuit de laatste positie waarop hij de contour verlaten heeft. Vervolgens verlaat robot enkel de contour wanneer de afstand vanuit de huidige positie tot het doel korter is dan de te onthouden afstand. Dit herhaalt zich totdat de robot het einddoel bereikt of geen oplossing mogelijk blijkt te zijn.



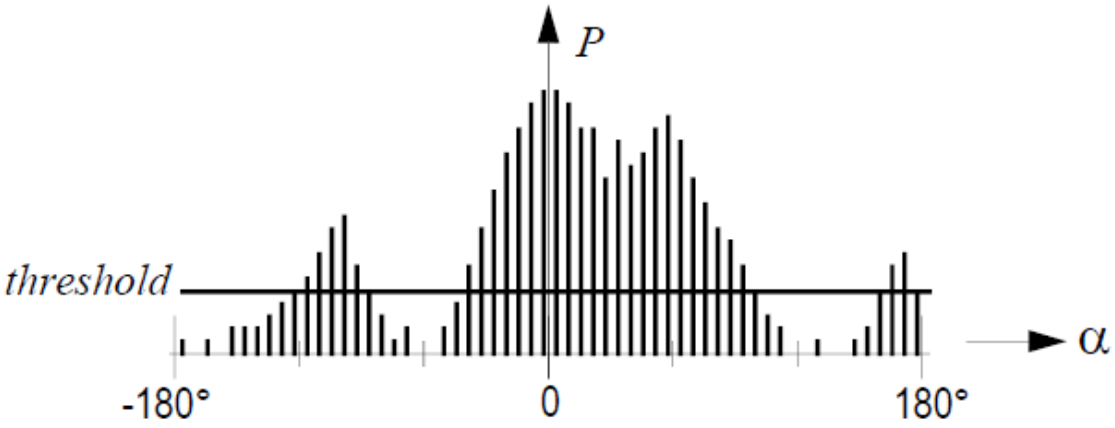
Figuur 2.24: Het bug 2 algoritme waarmee de robot een obstakel vermijdt om het doel te bereiken. Hierbij volgt de robot de contour van het obstakel totdat hij de lijn bereikt die het einddoel verbindt met het punt waarop de robot de contour begint te volgen. Hieruit vertrekt de robot verder naar het doel [6, p. 670].



Figuur 2.25: Voorbeeld van een slecht gebruik van het bug 2 algoritme. Hierbij gebruikt het algoritme zijn extra voorwaarde om de contour slechts te verlaten wanneer de afstand vanuit de huidige positie tot het doel korter is dan vanuit het punt waarop de robot de laatste keer de contour verliet [6, p. 670].

2.4.2 Vector field histogram

De vector field histogram (VFH) techniek overkomt de beperkingen van het bug algoritme dat enkel rekening houdt met de meest recente sensordata [11]. Zoals beschreven in [16] bouwt VFH een lokale kaart van de omgeving op rondom de robot op basis van relatief recente sensormetingen. Deze kaart stelt een bezettingsrooster voor rondom de robot. De VFH-methode vormt deze kaart om in een ééndimensionaal polair histogram. Zoals te zien in Figuur 2.26 stelt de x-as de hoek rondom de robot voor waarin een obstakel is waargenomen en stelt de y-as de waarschijnlijkheid voor dat op die hoek zich werkelijk een obstakel bevindt. Deze waarschijnlijkheid baseert zich op de cellen uit het bezettingsrooster rondom de robot.



Figuur 2.26: Polair histogram van de VFH-techniek waarbij de waarschijnlijkheid dat een cel uit het bezettingsrooster rondom de robot daadwerkelijk een obstakel bevat een functie is van de hoek rondom de robot [16, p. 238].

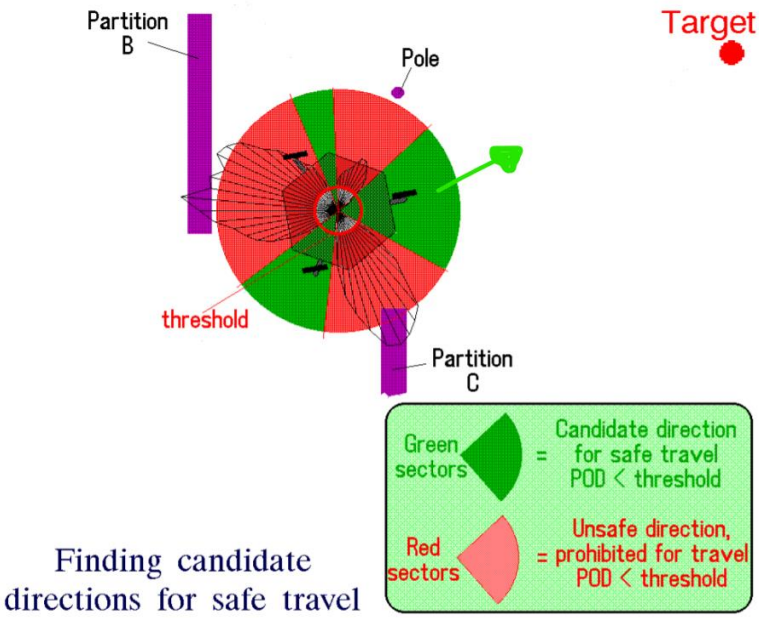
Vervolgens bepaalt de techniek de richting waarin de robot verdergaat. Dit gebeurt door eerst vanuit het polair histogram alle openingen te bepalen die breed genoeg zijn voor de robot om erdoor te navigeren (op Figuur 2.26 met behulp van de threshold waarde). Elke opening krijgt een kost toegewezen waarna de robot door de opening met de kleinste kost navigeert. Deze kost bestaat ui een combinatie van volgende gegevens:

- De richting van het doel: De maat waarin het pad van de robot zich aligneert met de richting tot het doel. Hierbij is een goede alignering gewenst voor een lage kost.
- De oriëntatie van de wielen: Het verschil tussen de nieuwe en de huidige oriëntatie van de wielen. Hierbij is een klein verschil in oriëntatie gewenst voor een lage kost.

- De vorige richting: Het verschil tussen de vorige geselecteerde en de nieuwe geselecteerde richting. Hierbij is een klein verschil van de richting gewenst voor een lage kost.

Elke term van de kostenfunctie heeft een vermenigvuldigingsfactor om de functie aan te passen naar een gewenste voorkeur van een van de termen.

Onderstaande Figuur 2.27 illustreert een voorbeeld van VFH-techniek.



Figuur 2.27: Weergaven van de VFH-techniek met de waarschijnlijkheid waarbij een obstakel zich bevindt in een hoek rond de robot. Hierbij stellen de groene zones de openingen voor waardoor de robot zich kan navigeren. Verder kiest de robot voor de opening met de groene pijl aangezien de kost ervan het laagst is dankzij de alineëring van de robot met de richting naar het doel [5].

De VFH-techniek heeft als nadeel dat het geen rekening houdt met de kinematica en dynamica van de robot. Uitbreidingen op de VFH-techniek houden wel rekening met de kinematica van de robot. Hierbij stellen de waarschijnlijkheden op obstakels in het polair histogram bijkomend de acties voor die de robot wegens zijn kinematica niet kan uitvoeren [11].

2.4.3 Dynamic window approach

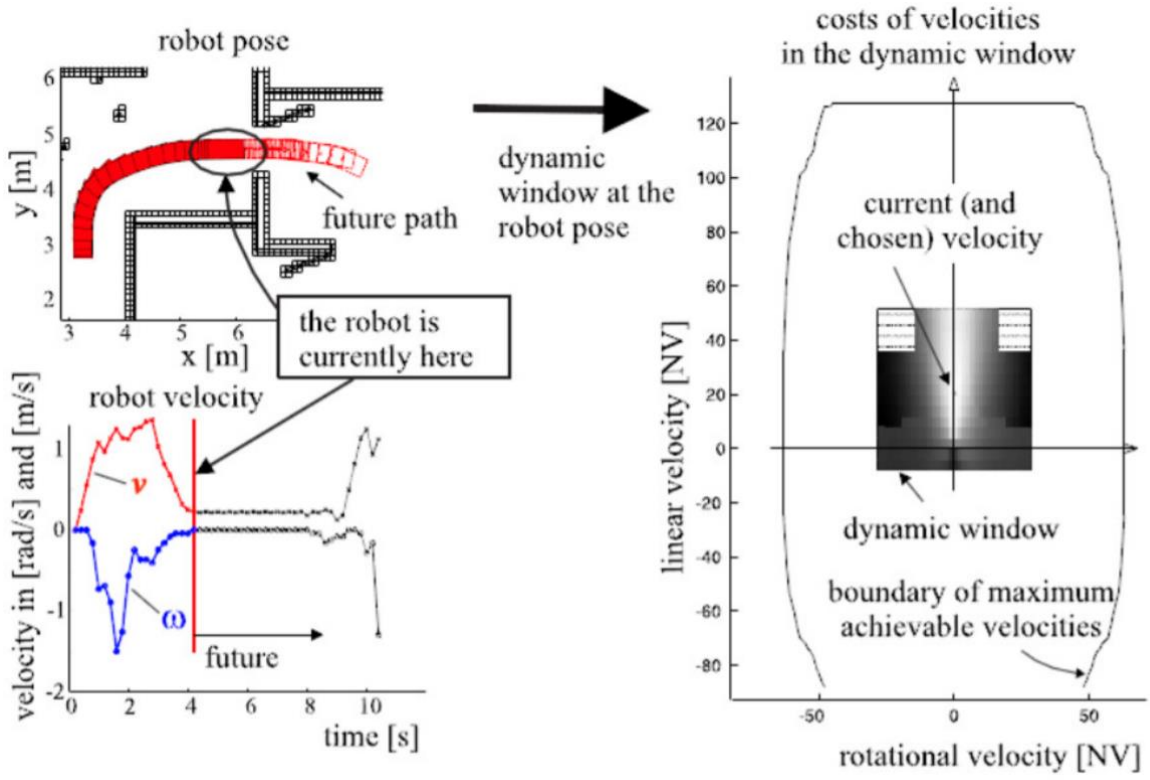
Een andere techniek is de dynamic window approach (DWA) die rekening houdt met de kinematische en dynamische beperkingen van een robot. Zoals beschreven in [11], [17] maakt de DWA een aanname van enkel rechtlijnige of gebogen paden. De DWA-methode brengt de fysische beperkingen (maximale snelheid, maximale versnelling en obstakels in de omgeving) in kaart in een snelheidsruimte. Hierbij houdt de DWA in deze ruimte enkel rekening met de mogelijke snelheden in een afbakening (window) rondom de huidige snelheid. De maximale versnellingen van de robot bepalen de grootte van deze afbakening. Hierdoor bezit de afbakening enkel de snelheden die de robot kan behalen in de volgende stap. Door de afwezigheid van mogelijke snelheden die voor een botsing zouden zorgen met een obstakel, bezit de afbakening informatie over de positie van de te vermijden obstakels. De keuze voor de nieuwe snelheid van de robot baseert zich op een

kostenfunctie die enkel betrekking heeft op de ruimte van mogelijke snelheden in de afbakening. De functie bestaat uit de combinatie van volgende gegevens:

- De snelheidsrichting: Een voorwaartse snelheid is het meest gunstig.
- De richting tot het doel: Een alignering met de richting tot het doel is gewenst.
- De afstand tot obstakels: Een grote afstand tot de obstakels is gewenst.

Ook hier heeft elke term van de functie een vermenigvuldigingsfactor om de functie aan te passen naar een gewenste voorkeur van een van de termen.

ROS kan op een eenvoudige manier de DWA implementeren voor een robot, gegeven een globaal pad en een kostenkaart [18]. Hierbij geeft het bevelen aan de mobiele basis van de robot om een bepaalde snelheid aan te nemen. De voorstelling van de robot moet hiervoor bestaan uit een veelhoek of een cirkel.



Figuur 2.28: Schematische weergave van de dynamic window approach waarbij de robot een nieuwe snelheid kiest op basis van een afbakening van mogelijke snelheden en op basis van een keuzefunctie. In deze figuur stelt het vierkant in de rechter tekening de afbakening voor van mogelijke snelheden met daarin twee lege vierkanten. Deze vierkanten zijn een afwezigheid van mogelijke snelheden omdat deze snelheden een botsing met een obstakel zouden veroorzaken. Vervolgens kiest de robot een snelheid recht vooruit om de obstakels te vermijden en het doel te bereiken [19, p. 2698].

2.4.4 Conclusie

Dit deel besprak de mogelijke technieken om lokaal, onvoorziene obstakels op het globale pad te vermijden. De eenvoudigste manier is het bug algoritme waarbij de robot de contour van een obstakel volgt. Bij het bug 1 algoritme verlaat de robot de contour na een volledige omtrek vanuit het punt op de contour dat de kleinste afstand heeft tot het doel. Bij het bug 2 algoritme daarentegen, verlaat de robot de contour al wanneer hij de lijn bereikt die het doel verbindt met het punt waarop

de robot de contour begon met volgen. Het bug algoritme houdt echter enkel rekening met de meest recente sensordata en houdt geen rekening met de kinematische beperkingen van de robot. De vector field diagram (VFD) techniek houdt rekening met meerdere sensordata om een polair histogram op te stellen van de waarschijnlijkheid dat een obstakel zich bevindt rondom de robot. Op basis hiervan maakt de robot een keuze voor de richting waarin het gaat navigeren. Dit gebeurt met behulp van een minimumwaarde voor de openingen tussen obstakels en met behulp van een kostenfunctie omtrent de pose van de robot ten opzichte van het doel. Hoewel verbeteringen op de VFD-techniek wel rekening houden met de kinematica van de robot, houdt de VFD-techniek geen rekening met de dynamica van de robot. Tot slot houdt de Dynamic Window Approach (DWA) wel rekening met de kinematische en dynamische beperkingen van de robot. Dit doet het door de mogelijke volgende snelheden weer te geven in een afbakening in de snelheidsruimte. Deze afbakening houdt bijgevolg ook rekening met de obstakels. Vervolgens kiest de robot een volgende snelheid uit de afbakening met behulp van een keuzefunctie omtrent de ligging van het doel en van de obstakels. Bovendien is een eenvoudige implementatie van de DWA in ROS mogelijk.

2.5 Padplanning voor mobiele manipulatoren met een hoog aantal vrijheidsgraden

De combinatie van een mobiel platform en een manipulator resulteert in een mobiele manipulator met een hoog aantal vrijheidsgraden (>6). Hierdoor is het aantal dimensies in de C-space ook hoog. Het gebruik van samplegebaseerde planners in deze C-space is computationeel duur [20]. Bovendien zorgt de combinatie van basis en arm voor redundante vrijheidsgraden. Dit betekent dat er meerdere aanstuurmogelijkheden zijn voor eenzelfde beweging van de eindeffector waardoor de aansturing complex is. Aangezien de aansturing van de robotarm nauwkeuriger is, bestaat de traditionele aanpak uit een ontkoppelde planning van een mobiele manipulator met als eerste stap het rijden van de basis naar het doel. Na het parkeren van de basis, voert de manipulator een taak uit waarna de basis weer wegrijdt [21]. Dit gebeurt bijvoorbeeld met een globale Dijkstra of A* planner in combinatie met een lokale DWA-planner voor het aansturen van de basis en met een sample-gebaseerde OMPL planner zoals RRT voor de manipulator. Hedendaags onderzoek focust zich echter op de soms meer voordelige of noodzakelijke combinatie van basis- en armbewegingen voor het uitvoeren van een taak. Hoewel de complexiteit van de aansturing hierbij toeneemt, stijgt de gewenste flexibiliteit en vrijheid bij het manoeuvreren van grote objecten, het ontwijken van obstakels of het openen van deuren [21].

Volgende delen bespreken verschillende methoden voor het plannen van een pad of de control van mobiele manipulatoren. De verschillende aansturingen bestaan hoofdzakelijk uit twee groepen, namelijk de model- en de behavior-based aanpak. De modelgebaseerde aanpak tracht de bewegingen van de robot op voorhand punt per punt te plannen waardoor de aanpak zeer doordacht is. Dit vereist echter een exacte beschrijving (model) van de omgeving en van de te manipuleren objecten. Hierdoor is de aanpak niet flexibel in andere omgevingen. Binnen de modelgebaseerde groep zitten enerzijds planners die het aantal vrijheidsgraden proberen te beperken door een opsplitsing te maken tussen manipuleren (arm) en navigeren (basis). Anderzijds bestaan er planners die werken met het volledig aantal vrijheidsgraden. De planners uit de modelgebaseerde groep zijn meestal een van (of combinatie van) de reeds besproken padplanners en obstakelvermijdingsalgoritmes. Hierbij is eventueel een bovenliggend algoritme vereist dat de mobiele planners en armplanners coördineert (samenwerking tussen arm bewegen of basis bewegen) om zodanig de redundantie van de mobiele manipulator op te lossen. De behavior-based

aanpak daarentegen plant geen bewegingen op voorhand maar maakt gebruik van gedragen (behaviors) die de gewenste bewegingen van de robot in bepaalde situaties beschrijven. Op deze manier is de robot meer reactief ten opzichte van de omgeving. Bij deze aanpak is geen exact wereldmodel vereist waardoor het gebruik van de robot flexibeler is en de robot overweg kan met variaties in de omgeving.

Hieronder volgen eerst enkele modelgebaseerde aanpakken gevolgd door enkele behavior-based aanpakken.

2.5.1 Modelgebaseerde aanpak: Opsplitsing configuratieruimtes mobiele basis en manipulator

De planning van taken met verschillende randvoorwaarden, zoals het hanteren van objecten met de eindeffector, in de hoogdimensionale configuratieruimte van een mobiele manipulator met meerdere vrijheidsgraden is computationeel veeleisend. Een mogelijke oplossing hiervoor is het aanpassen van deze configuratieruimte naar een kleinere ruimte of het splitsen ervan in de configuratieruimte van de mobiele basis en de manipulator. Hiervoor is een bepaalde aanpak vereist om de controle tussen arm en manipulator te regelen. Wat volgt zijn enkele mogelijke padplanningsmethoden die de configuratieruimte opsplitsen of aanpassen.

2.5.2 HAMP

Het onderzoek in [22] introduceert een probabilistisch compleet, samplegebaseerd padplanningsraamwerk voor mobiele manipulatoren, genaamd HAMP (Hierarchical and Adaptive Mobile Manipulator Planner), dat gebruik maakt van de afzonderlijke configuratieruimtes van de basis en de arm. Hierbij dient de manipulator slechts van configuratie te veranderen indien het in botsing is met de omgeving tijdens het volgen van het pad van de mobiele basis. Met deze aanpak moet de manipulator niet noodzakelijk in een rustconfiguratie[3] zijn gedurende het verplaatsen van de basis. HAMP maakt gebruik van de PRM als *subplanner* voor basis en arm afzonderlijk. Andere OMPL-planners zoals RRT zijn evenzeer bruikbaar.

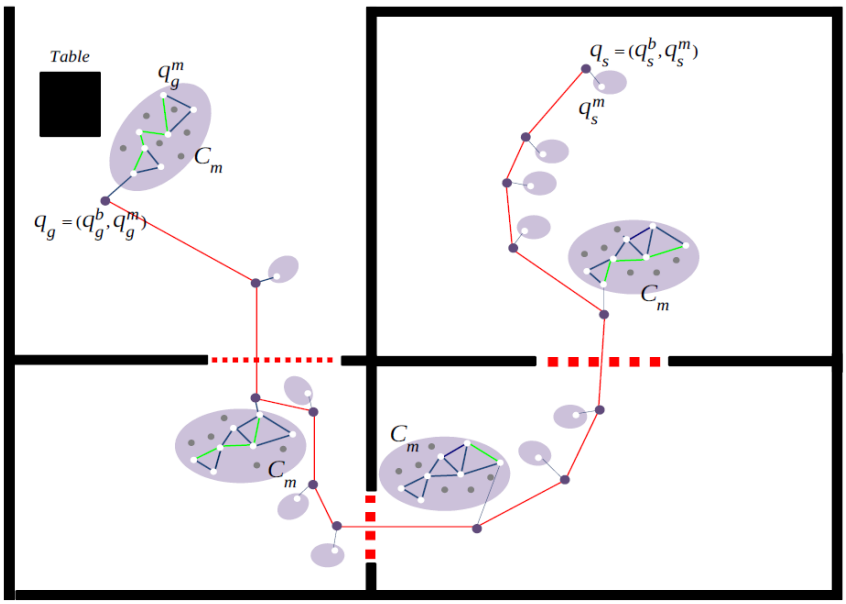
Het HAMP-algoritme bestaat uit twee fasen en doorloopt deze totdat een geldig pad voor de mobiele manipulator gevonden is of totdat de tijdslimiet verstreken is. De eerste fase stelt een roadmap op voor de basis, gebruik makend van een PRM in de basisconfiguratieruimte, om de start- en eindpositie van de robot te verbinden. Dit gebeurt voor de robotconfiguratie waarbij de arm zich in de rustconfiguratie bevindt. In deze stap zoekt het PRM-algoritme dus voor geldige posities van de basis doorheen de configuratieruimte van de basis maar controleert het op botsingen van de volledige robotconfiguratie met de arm in rustconfiguratie. Elk knooppunt in de basisroadmap krijgt bovenop de basispose en de armconfiguratie ook een optimale padkost toegekend.

In de tweede fase doorzoekt het algoritme de basisroadmap voor een optimaal pad met behulp van een verstrengeling tussen Dijkstra's algoritme en een algoritme dat de armconfiguratie aanpast. Zoals besproken in 2.2.3 doorloopt Dijkstra's algoritme de lijst Q van knooppunten in de roadmap.

[3] De rustconfiguratie of de *homeconfiguration* is een configuratie van de arm die het kan aannemen tussen het uitvoeren van taken, tijdens het rijden, tijdens het wachten of wanneer de robot is uitgeschakeld. Deze configuratie is niet noodzakelijk gelijk aan de startconfiguratie van het pad. De startconfiguratie kan echter voortvloeien uit de eindconfiguratie van een vorige taak.

Tijdens het doorlopen controleert het algoritme voor de armconfiguratie voor elk knooppunt of de arm in botsing zou komen gedurende de verplaatsing naar het volgend knooppunt. Indien dit het geval is, zoekt een PRM-algoritme in de configuratieruimte van de arm naar een nieuwe armconfiguratie die niet in botsing is met de omgeving tijdens de volgende verbinding. De arm neemt dan deze configuratie aan vanaf dit knooppunt tijdens de uitvoering van het pad. Deze zoektocht doorheen de knooppunten stopt bij het eindknooppunt. Indien het algoritme geen armconfiguratie vindt waarvoor de robot niet in botsing is tijdens de verbinding tussen het huidig knooppunt en het volgende, herstart het algoritme de eerste fase van HAMP waarbij een uitbreiding van de knooppunten van de PRM voor de basis plaatsvindt. Tot slot zoekt het algoritme op dezelfde manier (PRM en Dijkstra in de configuratieruimte van de arm) naar een pad tussen de huidige armconfiguratie in het laatste knooppunt en de gewenste eindconfiguratie van de arm.

Onderstaande Figuur 2.29 toont de werking van de HAMP-aanpak.



Figuur 2.29: Voorstelling van de HAMP-aanpak waarbij de robot start in startconfiguratie q_s met de startconfiguratie van de basis q_s^b en met de startconfiguratie van de manipulator q_s^m. Het algoritme bouwt een PRM op voor de basis met behulp van de basisconfiguratieruimte. Vervolgens controleert het algoritme voor elk knooppunt of de armconfiguratie voor een botsing tijdens de verbinding naar een volgend knooppunt zou zorgen. Indien niet, is enkel de huidige armconfiguratie (in het begin q_s^m) nodig in het knooppunt. Dit is weergegeven door het enkele witte punt in de kleine ellips dat de C-space van de arm voorstelt. Indien voorspelde botsing, zoals tijdens het passeren onder de overhangende muur (stippellijn), past de armconfiguratie zich aan naar en nieuwe (niet-botsende) configuratie gevonden in de armconfiguratieruimte C_m (weergegeven door de grote ellips) met behulp van PRM en Dijkstra. Tot slot zoekt het algoritme op dezelfde manier naar een pad tussen de huidige armconfiguratie in het eindknooppunt en de eindconfiguratie van de arm q_g^m om de volledige eindconfiguratie q_g te bereiken [22, p. 46].

Het algoritme faalt met het zoeken naar een oplossing indien:

1. Het niet in staat is om een pad te vinden tussen begin- en eindconfiguratie van de mobiele basis met de arm in rustconfiguratie.
2. Het niet in staat is om een nieuwe armconfiguratie te vinden in een knooppunt waar dit nodig is. Enerzijds omdat de armconfiguratie zonder botsing doorheen de verbinding niet gevonden werd binnen de tijd of anderzijds omdat er geen pad gevonden werd binnen de tijd tussen de huidige armconfiguratie en de nieuwgevonden armconfiguratie.

3. Het algoritme bereikt het eindknooppunt maar is niet in staat om een pad te plannen tussen de huidige armconfiguratie en de eindconfiguratie van de arm.

Het onderzoek vergelijkt de HAMP-aanpak met een PRM-aanpak in de volledige 9 DoF C-space van een mobiele manipulator. Hieruit volgt dat HAMP in staat is om in minder tijd paden te zoeken die korter zijn en een hoger succesgehalte hebben dan de 9 DoF PRM. Bovendien vermijdt HAMP onnodige berekeningen in de C-space van de arm en onnodige bewegingen van de arm.

In het artikel van het onderzoek staat de volledige pseudocode van het HAMP-algoritme en van alle nodige onderliggende algoritmes. Bovendien maakt het gebruik van open-source OMPL subplanners zoals PRM of RRT. Hierdoor valt HAMP relatief eenvoudig na te maken voor implementatie op een mobiele manipulator.

Tot slot stelt het uitbreidend onderzoek [23] een samplegebaseerde padplanning voor mobiele manipulatoren voor dat rekening houdt met de onzekerheid op de basispose en de effecten van deze onzekerheid op de armbewegingen, gebruik makende van de HAMP-aanpak.

2.5.3 Graph-based representation

Het onderzoek in [24] introduceert een aanpak voor het plannen van een deuropeningstaak voor mobiele manipulatoren die bovendien uit te breiden is naar andere taken. De methode maakt gebruik van een laag gedimensioneerde, grafegebaseerde representatie (vertakte structuur) van de mogelijke (botsingsvrije) configuraties van de mobiele basis om een taak van de robot uit te voeren. Vervolgens genereert de methode met behulp van inverse kinematica het traject van de manipulator volgens het pad van de basis. Dit gebeurt rekening houdend met de randvoorwaarden van de taak (in dit geval dat de eindeffector steeds in contact moet zijn met de klink om zodanig de deur te openen). Door deze splitsing in twee afzonderlijke ruimtes, is de planning van de robotbeweging voor een bepaalde taak sneller en minder computationeel intensief in vergelijking met het plannen van een taak met randvoorwaarden in de volledige, hoog-dimensionale configuratieruimte. Het onderzoek demonstreert deze aanpak door het succesvol openen van zowel een duw- als trekdeur door een PR2 robot met omnidirectionele basis en een arm met 7 DoF.

De voorgestelde graph-based aanpak voor het openen van een deur met de PR2 robot werkt als volgt:

1. De methode maakt gebruik van een basisruimte die de pose van de basis en de toestand van de deur (waarde tussen 0 en 1 voor resp. open of toe) beschrijft. Bijkomend heeft elke status van de basis een verzameling van hoeken van de deur toegekend waarvoor de deur binnen bereik is van de arm zonder met de robot de omgeving te botsen. Verder beschouwt de voorgestelde methode enkel die statussen in deze basisruimte waarvoor er minstens één mogelijke deurhoek bestaat.
2. Vervolgens verbindt een boomstructuur (graph), die zich uitbreidt vanuit de startconfiguratie, deze punten in de basisruimte. Hierbij stelt de verbinding tussen twee punten de overgang naar een volgende basisconfiguratie voor. Door het gebruik van deze structuur is elke gevonden oplossing ook een werkelijk uitvoerbare oplossing wat deze aanpak geschikt maakt voor niet-holonomische robots met meerdere randvoorwaarden.

3. Hierna krijgt elk punt een kost toegekend op basis van de afstand van de basis in dat punt tot het dichtstbijzijnde obstakel. Deze kostverdeling gebeurt op basis van een 2D kostenmap door een projectie van de 3D omgeving tot een 2D obstakelomgeving.
4. Bijkomend krijgt elk punt een kost toegekend op basis van de positie van de deurklink ten opzichte van de manipulator zodanig dat de klink (makkelijk) te bereiken is, rekening houdend met de limieten van de manipulator in die basisconfiguratie.
5. Aan elke actie tussen de statussen is nu een kost geassocieerd om die actie uit te voeren.
6. Het Anytime Repairing A* algoritme (ARA*) zoekt, met een beperkte rekentijd, een pad doorheen de boomstructuur [25]. Dit algoritme genereert snel een initieel, suboptimale oplossing en focust zich hierna op het verbeteren van die oplossing zolang de opgegeven rekentijd het toestaat. Hierbij houdt het rekening met de opgegeven ondergrens van suboptimaliteit ten opzichte ven het optimaal pad.
7. Hierna is het pad van de basis gekend. Verder is ook de toestand van de deur, de positie van de klink en de configuratie van de eindeffector gekend bij elke pose van de basis (beweging deur bij het openen, van waarde 0 tot 1).
8. Tot slot genereert een algoritme het Cartesiaans pad van de eindeffector tijdens het openen van de deur. Een solver voor de inverse kinematica zet dit pad om naar de opeenvolgende configuraties van de arm. Wegens de redundantie van de arm (7 DoF), doet de solver eerst een gok van de configuratie van een van de joints om daarna een analytische oplossing te zoeken voor de overige 6 DoF.

2.5.4 Adaptive Dimensionality

Het onderzoek in [26] stelt een mobiele manipulator planningsmethode voor, gebruik makende van een deterministisch zoekalgoritme in een dimensioneel adaptieve configuratieruimte. Hierbij zoekt het algoritme doorheen een configuratieruimte bestaande uit een laagdimensionale ruimte met enkel hoogdimensionale gebieden waar nodig (complexe omgeving, veel obstakels). Dit resulteert in een snellere planning doorheen de ruimte in vergelijking met de volledig hoogdimensionale ruimte.

Zoals reeds besproken maken sommige aanpakken eerst gebruik van een 3D ruimte van de eindeffector om het initieel pad te plannen. Hierna zet een solver voor de inverse kinematica dit pad om naar het traject in de ruimte met de volledige dimensies. Volgens dit onderzoek zijn er echter situaties waarin de initiële planning rekening moet houden met het volledig aantal dimensies.

Om deze reden maakt de voorgestelde methode gebruik van een (adaptieve) configuratieruimte S_{ad} met bijhorende verzameling T_{ad} aan transities (verbindingen) tussen mogelijke statussen. De ruimte S_{ad} bestaat hoofdzakelijk uit laagdimensionale statussen en transities en enkele hoogdimensionale statussen en transities waar vereist voor het vinden van een geldig pad. Deze zijn afkomstig uit de laag- en hoogdimensionale ruimtes S_{ld} en S_{hd} die, in het geval van een 11 DoF mobiele manipulator, respectievelijk de configuratie van de 3 DoF eindeffector en van de 11 DoF robot voorstellen. De methode maakt gebruik van projecties om van de hoogdimensionale ruimte over te gaan naar de laagdimensionale waar nodig. De transities tussen alle statussen (laag- en hoogdimensionaal) uit S_{ad} vormen een vernette structuur (graph) G_{ad} van adaptieve dimensionaliteit.

Alle statussen uit S_{ad} (afkomstig van S_{ld} en S_{hd}) en dus ook transities tussen statussen hebben reeds een kost toegekend omtrent de afstand tot obstakels op basis van sensordata of de gekende omgeving. Vervolgens bestaat de planning uit twee opeenvolgende fasen. In de eerste, adaptieve planningsfase

zoekt een 3D Dijkstra algoritme doorheen G_{ad} naar een pad met minimale kost. Vervolgens construeert de meeloop of opsporingsfase (*trackingfase*) met behulp van het ARA* algoritme een uitvoerbaar pad doorheen de hoogdimensionale ruimte dat overeenkomt met het pad uit de adaptieve planningsfase. Hierbij is het eerste pad uit de adaptieve fase een hulpmiddel voor de zoekopdracht in de hoogdimensionale ruimte. Op deze manier vermijdt de voorgestelde methode een computationeel dure, initiële planning in de hoogdimensionale ruimte.

Het onderzoek demonstreert de capaciteiten van de voorgestelde methode met simulaties en enkele testen met een 11 DoF PR2 robot met een 7 DoF arm. De resultaten tonen aan dat de adaptieve methode sneller opmerkt indien de zoekopdracht in de foute richting evolueert dan de methode die gebruik maakt van de volledige 11 DoF configuratieruimte. Bijkomend heeft de adaptieve methode, ten koste van een hogere rekentijd, een hoger succesgehalte dan een samplegebaseerde methode die gebruik maakt van bidirectionele RRT in de volledige configuratieruimte. De resultaten bewijzen de compleetheid van de planner en de werking van een opgegeven ondergrens voor suboptimaliteit ten opzichte van een pad uit de volledige configuratieruimte. De resultaten van de adaptieve planner zijn echter sterk afhankelijk van de ingestelde parameters.

2.5.5 Modelgebaseerde aanpak: Volledige configuratieruimte mobiel manipulator

De eerder besproken methoden beschouwen de twee deeltaken navigeren en manipuleren om een optimaal pad van een mobiele basis te bekomen voor een gegeven taak. Volgens [27] zorgt een combinatie van deze geoptimaliseerde deelproblemen tot een suboptimale oplossing van het geheel, aangezien deze deelproblemen elkaar beïnvloeden. Onderstaande methoden trachten een optimaal pad te plannen voor de volledige mobiele manipulator zonder opsplitsing te maken tussen mobiele basis en manipulator.

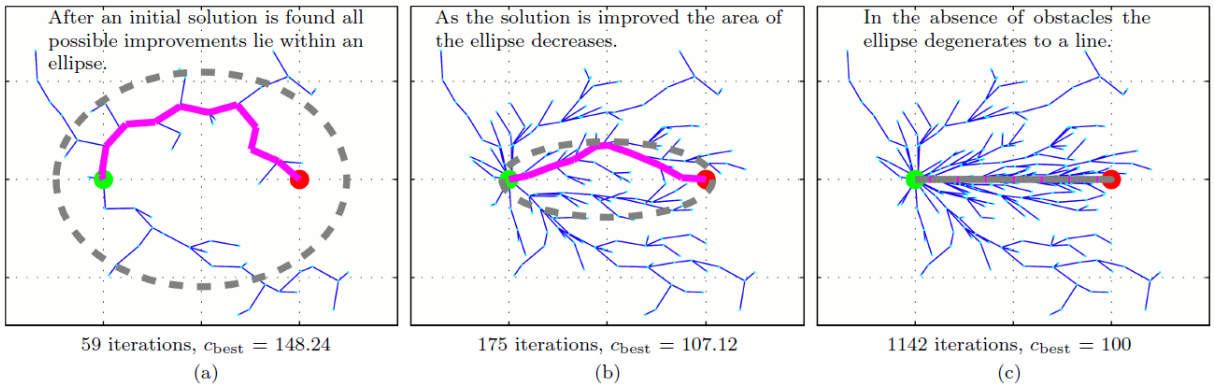
2.5.6 Optimalisatiegebaseerde aanpak voor pick-and-place operaties

Het onderzoek [28] introduceerde een aanpak rondom grijpen en padplannen voor het uitvoeren van autonome pick-and-place operaties van een mobiele manipulator in een statische omgeving. Hierbij moet de robot een object verplaatsen tussen een gekende start- en eindconfiguratie, gegeven het gekende robot- en omgevingsmodel. Het algoritme bestaat uit een optimalisatie- en planningsfase. In de optimalisatiefase zoekt een co-evolutionair algoritme de beste combinatie van robotbasis- en grijperconfiguratie in de gekende start- en eindpositie. Dit gebeurt op basis van een score voor de kwaliteit van de robotconfiguratie die rekening houdt met: in welke maten de grijper het object op een krachtgesloten manier vastheeft, in welke maten de robotconfiguratie kinematisch mogelijk is en in welke maten de robotconfiguratie botst met of dicht staat bij obstakels in de omgeving. Nadat de optimale mobiele basis- en grijperconfiguratie in start- en eindpositie gekend zijn, bepaald een invers kinematica algoritme de armconfiguraties in beide posities zodanig dat de volledige robotconfiguratie in beide posities gekend is. Vervolgens start de planningsfase waarin een bidirectioneel RRT-algoritme een optimaal pad zoekt tussen de start- en eindconfiguraties (uitgezonderd de grijper: deze configuratie blijft behouden zodanig de grip niet wijzigt tijdens het transport). Dit gebeurt in de volledige, robotafhankelijke configuratieruimte (C-space). Het voorgesteld algoritme is in staat om efficiënt en snel grijper- en basisconfiguraties te vinden die voldoen aan een hoge score en is in staat om hierna een pad te plannen voor complexe pick-and-place toepassingen. Deze methoden is echter statisch (volledige herplanning tussen A en B vereist

bij onvoorziene gebeurtenissen) en computationeel complex wegens het gebruik van de volledige C-space. Deze aanpak vormde een basis voor verschillende verdere onderzoeken naar mogelijke aanpakken.

2.5.7 BI2RRT*

Verschillende methoden trachten de computationele complexiteit en rekentijd te verlagen door de C-space niet langer uniform te samplen. Het onderzoek in [29] stelt voor om dit te doen aan de hand van Informed RRT*. Deze planner werkt zolang er geen initiële oplossing gevonden is analoog aan RRT*. Zodra er een eerste pad gevonden is, plant het algoritme enkel nog in een subset die de gevonden oplossing kan verbeteren. Onderstaande Figuur 2.30 illustreert dit. Bij elk korter gevonden pad verkleint de subset waardoor de planner sneller convergeert.



Figuur 2.30: Voorstelling van de sampling bij BI2RRT* (a) Ellips vormt de subset waarin gesampled wordt na het vinden van een eerste pad. (b) Ellips verkleint bij elke verbetering van het pad. (c) Wanneer er geen obstakels zijn, vormt het kortste pad een lijn. De ellips verkleint ook tot een lijn aangezien er geen kortere paden zijn [29, p. 2999]

De convergentiesnelheid van deze planner is groter dan RRT* bij toenemend aantal dimensies. Hierdoor is deze planner interessant voor robots met veel vrijheidsgraden. Een verdere uitbreiding op Informed RRT* is Bidirectionele Informed RRT* (BI²RRT*). BI²RRT* start in zowel de begin- als eindconfiguratie een zoekboom. Hierdoor is de planner sneller instaat om een eerste oplossing te vinden en kan deze dus sneller samplen in de subset. Door deze verbetering convergeert deze planner nog sneller naar het optimale pad. Deze planner is ook instaat te plannen voor *constrained tasks*. Dit zijn taken waarbij er randvoorwaarden opgelegd worden aan de eindeffector. Een voorbeeld hiervan is het verplaatsen van een glas water. Hierbij zijn de randvoorwaarde de minimale en maximale hoeken van de eindeffector in het wereldassenstelsel zodat de eindeffector geen water morst [27].

2.5.8 Repetition roadmap

De repetiton roadmap is een manier om de samplegebaseerde RRT-planner te ondersteunen bij repetitieve taken. Bij deze repetitieve taken treden variaties op in pose en omgeving van de robot. Hierdoor is het niet mogelijk om deze bewegingen op voorhand te coderen maar is een planner vereist. Planning aan de hand van een repetition roadmap gebeurt in twee delen. In het eerste deel bouwt het algoritme uit onderstaande Figuur 2.31 de repetition roadmap op. Deze roadmap is enkel bruikbaar voor een bepaalde repetitieve taak en wordt offline berekend.

Algorithm 1: Construct Repetition Roadmap.

Input: A set of M previous solution paths $\mathcal{P}_0 ... \mathcal{P}_M$
Output: The Repetition Roadmap G
foreach *path* $\mathcal{P}_i$ **do**
    Extract key configurations $\mathbf{q}_0 ... \mathbf{q}_n$ from path $\mathcal{P}_i$
    Insert key configurations $\mathbf{q}_0 ... \mathbf{q}_n$ into learning set $\mathbf{Q}$
Compute a GMM based on the learning set $\mathbf{Q}$
Extract from the GMM all K Gaussians $\mathcal{N}_0 ... \mathcal{N}_K$
foreach *Gaussian* $\mathcal{N}_k$ **do**
    Insert a new vertex v_k into the roadmap G
    Assign the sampling distribution $\mathcal{N}_k$ to v_k
foreach *path* $\mathcal{P}_i$ **do**
    foreach *consecutive configurations* $\mathbf{q}_j$ *and* $\mathbf{q}_{j+1}$ *in* $\mathcal{P}_i$ **do**
        Match $\mathbf{q}_j$ to a Gaussian $\mathcal{N}_k$
        Match $\mathbf{q}_{j+1}$ to a Gaussian $\mathcal{N}_{k+1}$
        if $\mathcal{N}_k \neq \mathcal{N}_{k+1}$ **then**
            Insert a new edge e_{new} into the roadmap G from the respective vertex v_k to v_{k+1}

Figuur 2.31: Algoritme voor het opbouwen dan de repetition roadmap. Deze wordt opgebouwd uit voorgaande paden [30, p. 3886].

Het algoritme bouwt de repetition roadmap op uit een set van voorbeeldpaden voor een bepaalde taak. Eerst worden deze paden ontleed tot *key configurations.* Het algoritme deelt deze configuraties op in groepen met vergelijkbare poses aan de hand van een *Gaussian Mixture Model* (GMM). Vervolgens berekent het voor elk van deze groepen de parameters van de normale verdeling die deze groep beschrijven. Eén van deze parameters is het gemiddelde waaraan een knooppunt van de roadmap is toegekend. Om de verschillende knooppunten met elkaar te verbinden gaat een algoritme na of opeenvolgende key configurations van een bepaald pad tot verschillende knooppunten behoren. Indien hieraan voldaan is, betekent dit dat er een verbinding tussen deze knooppunten bestaat en worden deze knooppunten met elkaar verbonden. Bovendien kent het algoritme een waarde van connectiviteit toe aan elke verbinding. De connectiviteit van een verbinding wordt berekend op basis van hoe vaak de verbinding tussen bepaalde knooppunten voorkomt [30].

In het volgende deel plant het algoritme uit onderstaande Figuur 2.32 effectief een pad tussen begin- en eindpositie.

Algorithm 2: Query Repetition Roadmap.

Input: Start configuration $\mathbf{q}_{start}$, goal constraints S_{goal}, Repetition Roadmap G
Output: A solution path $\mathcal{P}_{sol}$
Initialize each vertex v_k of G with a search tree T_k with the sampling distribution $\mathcal{N}_k$
Match the start configuration $\mathbf{q}_{start}$ to a Gaussian $\mathcal{N}_{start}$ and the respective tree T_{start}
Grow T_{start} until it connects to $\mathbf{q}_{start}$
Find a goal configuration $\mathbf{q}_{goal}$, satisfying the constraints S_{goal}, with projection sampling.
Match the goal configuration $\mathbf{q}_{goal}$ to a Gaussian $\mathcal{N}_{goal}$ and the respective tree $Tgoal$
Grow T_{goal} until it connects to $\mathbf{q}_{goal}$
repeat
    Compute shortest path $\mathcal{P}_G$: v_{start} to v_{goal} through G
    foreach *Edge* e_i *on* $\mathcal{P}_G$ *connecting* v_k *and* v_{k+1} **do**
        Attempt to connect tree T_k to T_{k+1}.
        if *fails* **then**
            Expland trees T_k and T_{k+1}
            Lower utility u_i of e_i: $u_i = \alpha_u * u_i$
until $\mathbf{q}_{start}$ *is connected to* $\mathbf{q}_{goal}$
Extract $\mathcal{P}_{sol}$

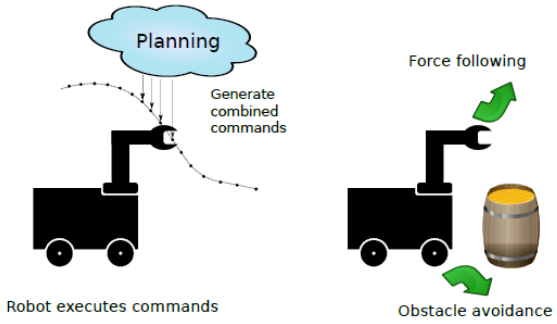
Figuur 2.32: Algoritme voor het opbouwen van een pad tussen start- en eindconfiguratie aan de hand van een eerder opgestelde repetition roadmap [30, p. 3887].

Het algoritme start door in elk knooppunt een zoekboom te starten. Hierbij gebeurt de sampling op basis van de bijbehorende Gaussiaan. Vervolgens zoekt het algoritme de knooppunten die het best passen bij de start- en eindconfiguratie. Vanuit deze knooppunten groeien zoekbomen totdat deze verbonden zijn met de start- en eindconfiguratie. Hierna zoekt het algoritme het kortste pad in de repetition roadmap tussen het start- en eindknooppunt. Hierbij hebben paden met een grotere connectiviteit een lagere padwaarde. Tenslotte probeert het algoritme de zoekbomen te verbinden van de knooppunten die tot het kortste pad behoren. Op deze manier verbindt het algoritme de start- en eindknooppunten met elkaar. Indien het algoritme niet in staat is de om de zoekbomen van een bepaalde verbinding met elkaar te verbinden, breidt het de zoekbomen verder uit en daalt de connectiviteit van deze verbinding. Hierna berekent het algoritme opnieuw een kortste pad. Doordat de connectiviteit van bepaalde verbindingen gedaald is, kan het zijn dat het kortste pad verandert. Het nadeel van dit algoritme is de nood aan bestaande paden. Bovendien is deze planner enkel geschikt voor repetitieve taken waarvan de configuratieruimte niet te sterk wijzigt. Bij sterke wijzigingen zijn de voorbeeldpaden niet langer van toepassing. Hierdoor gidst de repetition roadmap de sampling naar ruimtes die niet langer voor een oplossing zorgen [30].

Volgens [20] is een nadeel van samplegebaseerde planners het gebrek aan controle over de arm tijdens het navigatiegedeelte van een pick-and-place taak. Hierdoor kan het zijn dat de arm onnodige bewegingen maakt tijdens het navigeren tussen start- en eindpositie.

2.5.9 Behavior-based

De behavior-based aanpak maakt, in tegenstelling tot de modelgebaseerde methodologie, geen gebruik van een zo volledig mogelijke voorstelling van de omgeving of van de identificatie van obstakels [5], [31], [32]. Deze aanpak maakt gebruik van zogenaamde gedragen of acties die resulteren in de gewenste beweging met of in de omgeving. Dergelijk gedrag is een module dat bestaat uit een bepaalde relatie tussen sensordata en actuatoren waarbij elk gedrag afzonderlijk commando's zendt naar de hardware. De gedragen maken gebruik van gedeeltelijke, onvolledige wereldmodellen en genereren doelafhankelijke bewegingen op basis van omgevingsinformatie. Op deze manier is de sturing van de robot verdeeld over de verschillende gedragen die allen bepaalde actuatoren kunnen aansturen. Dit in tegenstelling tot de modelgebaseerde aanpak met een modelgebaseerde, centrale besturing (voor snelheid- en krachtbesturing) dat alle actuatoren aanstuurt. Hierdoor is de techniek niet gelimiteerd tot een vooropgesteld, exact model voor een specifieke handeling. Daarentegen is de techniek toepasbaar op verschillende toepassingen waarin dergelijke gedragsbewegingen gewenst zijn. Zo kan een gedrag bestaan uit het volgen van een muur waardoor een robot op basis van de behavior-based aanpak verschillende kamers kan rondrijden door de muren te volgen zonder dat het een kaart van de kamer vereist. Een veelvoorkomend mobiel gedrag is obstakelvermijding op basis van de artificial potential field aanpak. Een ander voorbeeld is het openen van een deur waarvan bij de traditionele aanpak een zojuist mogelijk model van de deur vereist is. De behavior-based aanpak maakt daarentegen gebruik van bepaalde gedragen die bewegingen met de omgeving uitvoeren (in dit geval bijvoorbeeld 'duwen met eindeffector') waardoor de robot verschillende vormen van deuren kan openen.

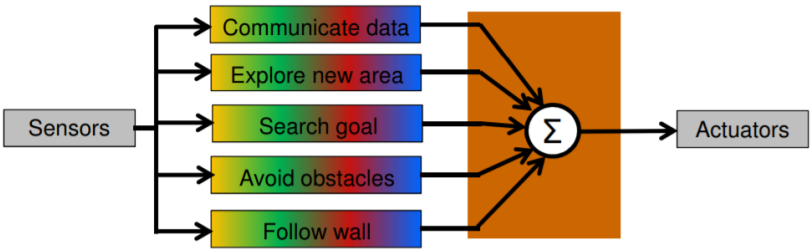


Figuur 2.33: Het verschil tussen de twee ontwerpmethoden voor een mobiele manipulator met sensoren op de manipulator en op het mobiel platform: (links) de traditionele centrale controle waarbij de robot de finale punten in de ruimte (na kaarttransformatie) doorloopt en (rechts) de behavior-based aanpak waarbij de robot reageert op de omgeving met verdeelde gedragen [31, p. 21].

Het grootste voordeel van deze aanpak is dat er geen exacte modellen van de omgeving of obstakels vereist zijn. Hierdoor is een behavior-based robot meer flexibel, adaptief en reactief. Aangezien er geen kaartopbouw vereist is, heeft ruis op sensordata geen nadelig effect tijdens de kaarttransformatie. Bijgevolg is er, zoals te zien op Figuur 2.33, geen nood aan een exacte planning doorheen de kaart. Het grootste nadeel van de behavior-based aanpak is het gebrek aan beraadslaagde, doordachte acties. Verder zijn de gedragen van de robot niet steeds functioneel in andere omgevingen dan de omgeving waarvoor de robot bedoeld is. Het is echter tijdsintensief om de gedragen te ontwerpen en deze nauwkeurig af te stemmen op een bepaalde omgeving.

De behavior-based aanpak heeft nood aan een verzameling van basisgedragen en aan een component die instaat voor het coördineren van gedragen. Deze *behavior fusion* component bepaalt het

resultaat van de interactie van twee aparte gedragen en vermijdt conflicten tussen de twee gedragen. Dit resultaat kan enerzijds een volgorde van de basisacties zijn of anderzijds een volledig nieuwe actie zijn, uit een combinatie van de basisacties. Zo kan bijvoorbeeld de combinatie van een draai- en duwbeweging van de eindeffector resulteren in een schroefbeweging van de eindeffector.



Figuur 2.34: Voorstelling van de behavior-based aanpak voor het aansturen van de actuatoren van een mobiele manipulator waarbij verschillende gedragen parallel lopen en het behavior fusion component instaat voor combinaties, volgordes of voorkeuren van acties op de actuatoren [5].

Er zijn twee soorten conflicten die zich kunnen voordoen. *Task level conflicts* treden op wanneer twee of meerdere gedragen met verschillende doelen bewegingen genereren die elkaar beïnvloeden. Dergelijk conflict kan ontstaan wanneer bijvoorbeeld een volggedrag, dat instaat voor het volgen van een bepaald beeld met een camera op de eindeffector, de eindeffector aanstuurt om het beeld te volgen terwijl het obstakelvermijdingsgedrag de eindeffector een andere richting opstuurt bij het detecteren van een mogelijke botsing. Een wegingsfactor van prioriteit voor elk gedrag kan dergelijke conflicten vermijden. *Hardware conflicts* treden op wanneer verschillende gedragen tegelijkertijd tegenstrijdige commando's sturen naar een actuator (bv. motor of robotgrijper) terwijl een actuator slechts één signaal tegelijk kan verwerken tot een output. De *behavior fusion* component treedt op als een extra stap voorafgaand aan de actuatoren om dergelijke conflicten te vermijden.

2.5.10 Behavior-based: Constraint-based behavior fusion mechanism

Het onderzoek uit [31] stelt een behavior-based aanpak voor die de brug maakt tussen de traditionele, beredeneerde modelgebaseerde planning en de reactieve, behavior-based robotcontrole. Het *constraint-based behavior fusion mechanism* (CBFM) maakt gebruik van *constraints* (randvoorwaarden) die de gewenste acties van bepaalde gedragen specifiëren voordat de actuatoren deze acties rechtstreeks uitvoeren. Deze aanpak baseert zich enerzijds op de modelgebaseerde aanpak [33], [34] en op de constraint aanpak [35]. De randvoorwaarden beschrijven de objectieven van de robotcontrole afkomstig van de behavior-based gedragen (bv. blijf op 30 cm afstand van obstakels). Op deze manier is er een meer beredeneerde controlelaag onder de reactieve gedragen en boven de actuatoren. De gedragen kunnen op deze manier niet rechtstreeks de actuatoren aansturen via het behavior fusion component maar moeten eerst door het behavior fusion mechanisme dat de randvoorwaarden genereert. Dit vermijdt snelle, impulsieve acties waardoor de algemene stabiliteit van de robot stijgt.

De traditionele modelgebaseerde planning beschrijft de bewegingen van de robot met exacte bewegingsvergelijkingen waaruit een reeks *setpoints* voor de actuatoren volgen. Net zoals andere behavior-based aanpakken, specifieert CBFM gedragen die de algemene bewegingen van de robot beschrijven. Deze gedragen kunnen overweg met kleine variaties van de omgeving. Verder

genereert CBFM uit deze gedragen de randvoorwaarden of randvoorwaardenvergelijkingen die de bewegingen beschrijven (algemener dan de bewegingsvergelijkingen van de modelgebaseerde aanpak). Deze vergelijkingen definiëren kracht, positie, impedantie of snelheid voor een beperkte verzameling van vrijheidsgraden en laat de andere vrijheidsgraden in een *don't care* status. Dit lost het probleem rondom de redundantie van vrijheidsgraden op. Deze randvoorwaarden komen tot stand in het behavior fusion mechanisme dat de verschillende gedragen gecombineert. Aangezien CBFM gebruik maakt van de tussenliggende laag van randvoorwaarden tussen de gedragen en de behavior fusion, is het een combinatie tussen de behavior aanpak en de modelgebaseerde aanpak.

Dankzij het gebruik van de randvoorwaarden heeft de behavior fusion component meer vrijheid bij de combinatie van gedragen (bv. vooruitrijden terwijl de eindeffector een obstakel vermijdt) dan bij het gebruik van setpoints. Hierdoor stijgt de flexibiliteit voor het definiëren van gedragen op verschillende coördinatenstelsels en voor de optimalisatie van bewegingen. Dankzij deze coördinatie tussen de gedragen is CBFM meer geschikt voor complexe toepassingen zoals mobiele manipulatoren.

Het onderzoek uit [31] implementeert het CBFM in de OROCOS-softwareomgeving van de KU Leuven [36]–[38] op hun mobiele manipulator LiAS[4] en maakt gebruik van een afstotende snelheidsvector *Virtual Force Field* (VFF) [39] als obstakelvermijdingsalgoritme. Enkele experimenten demonstreren de werking van een vaste eindeffector gedrag, een voorwaarts bewegingsgedrag en een gedrag waarbij de eindeffector een kracht volgt (geleid door een mens). Hieruit volgt dat de gebruikte mobiele manipulator verschillende doelen kan voltooien terwijl het botsingen met de omgeving vermijdt. Een ander experiment toont de mogelijkheid van de mobiele manipulator om deuren met verschillende vormen, groottes en klinken ongehinderd te openen met behulp van de CBFM-aanpak.

2.5.11 Conclusie en vergelijking

Dit deel besprak de verschillende padplanningsmethoden voor een mobiele manipulator met veel vrijheidsgraden. Allereerst bestaan er twee verschillende denkwijzen omtrent het controleren van het robotgedrag. Enerzijds de beredeneerde, voorop geplande, modelgebaseerde aanpak die nood heeft aan een exacte weergave van de omgeving en van de te manipuleren objecten. Anderzijds heeft de reactieve behavior-based aanpak minder nood aan dergelijke exacte modellen. De modelgebaseerde aanpak bestaat uit twee soorten strategieën waarbij de ene soort een opsplitsing of aanpassing van de volledige configuratieruimte hanteert terwijl de andere soort de volledige configuratieruimte van de mobiele manipulator in zijn geheel beschouwt. Uit de literatuur blijkt echter een gebrek aan open-source praktische implementaties van gecombineerde padplanningsmethoden. Verder zit de meerderheid van de methoden nog in een onderzoeksfase. Dit toont dat mobiele manipulatoren nog niet industrieel toepasbaar zijn wegens hun complexe besturing. Onderstaande Tabel 2.2 vergelijkt de verschillende methoden omtrent de toepasbaarheid op AMBER in het kader van deze masterproef.

[4] Soortgelijke mobiele manipulator aan AMBER: een niet-holonomische robot bestaande uit een mobiele basis met twee differentieel aangedreven wielen, een casterwiel en een robotarm bovenop gemonteerd.

Tabel 2.2: Vergelijkende tabel van padplanningsmethoden voor mobiele manipulatoren met redundante vrijheidsgraden waaruit de te gebruiken padplanner volgt. Hierbij duidt een hogere score op een gunstiger criterium van de methode en zorgt een hoger gewicht voor een grotere doorweging van de score voor het betreffend criterium.

		Model - based						Behavior-based
		Opsplitsing/aanpassing configuratieruimte				Volledige configuratieruimte		CBFM
	Gewicht	Traditioneel (eerst basis, dan arm)	HAMP	Graph-based representation	Adaptive Dimensionality	BI²RRT*	RepMap	
Efficiëntie (rekentijd)	1	2	4	3	3	2	3	4
Padoptimalisatie	1	2	3	3	3	5	4	3
Volledigheid (geen bijkomende planners vereist)	1	3	4	3	3	4	3	3
Redundantieoplossing	2	1	3	3	3	3	2	4
Toepasbaarheid MP (beschikbaarheid, open-source, zelf te maken, haalbaarheid)	3	5	1	1	1	2	1	1
Totale score		**24**	**20**	**18**	**18**	**23**	**17**	**21**

In bovenstaande tabel krijgt het criterium over de toepasbaarheid voor de masterproef het grootste gewicht aangezien sommige planners zeer complex en uitgebreid zijn voor gebruik bij een masterproef met beperkte tijd en achtergrondkennis. Door een beperkte tijd is het niet mogelijk om zelf een planner (na) te schrijven ook al zijn sommige zeer goed gedocumenteerd (in pseudocode). Ook het criterium over de redundantieoplossing krijgt een hoger gewicht aangezien deze masterproef hoofdzakelijk gaat over mobiele manipulatoren met meerdere (en redundante) vrijheidsgraden.

Uit de vergelijking blijken de BI²RRT*-planner en de traditionele, ontkoppelde aanpak de meest gunstige padplanningsmethoden voor AMBER binnen deze masterproef. Dit komt doordat de BI²RRT*-algoritmes open-source beschikbaar zijn. Deze moeten echter wel volledig aangepast worden voor een niet-holonomische robot wat niet eenvoudig blijkt. Hierdoor is de traditionele aanpak makkelijker implementeerbaar. Hierbij gebeurt de planning ontkoppeld tussen de basis en arm als volgt: de robot beweegt met de basis naar een doel, stopt vervolgens, voert een taak uit met de manipulator en rijdt hierna verder met de basis. Deze aanpak maakt echter geen gebruik van de flexibiliteit die verkregen wordt door het hoog aantal vrijheidsgraden van de volledige mobiele manipulator. Tot slot heeft deze aanpak nog bijkomende code nodig die de doelpositie van de eindeffector vertaalt naar de doelpositie van de basis. Deze masterproef maakt gebruik van de ontkoppelde padplanningsmethode. Hierbij onderzoekt het in volgende hoofdstukken in welke maten deze methode volstaat voor een scanopdracht van een muur als use case voor AMBER.

2.6 Conclusie

Deze literatuurstudie zorgt voor de nodige basiskennis over autonome navigatie van een robot(arm) om de padplanningsdoelstellingen voor de mobiele manipulator AMBER te voltooien.

De eerste sectie besprak de manier waarop verschillende zoekalgoritmes paden zoeken doorheen een wereldmodel. Deze algoritmes berusten allemaal op hetzelfde principe van het voorwaarts padplanningsalgoritme met als enige verschil de manier waarop de algoritmes de wachtlijst Q van volgende statussen doorlopen. Eén van de bekendste algoritmes is het Dijkstra algoritme dat kosten toekent aan punten om hen te bereiken. Hierdoor sorteert het Q op basis van een zo laag mogelijke kost. Op deze manier vindt het algoritme het pad met de laagste totale kost om het einddoel te bereiken in een wereldmodel waarin punten met elkaar verbonden zijn. Het A* algoritme is een verbetering van Dijkstra's algoritme dat bijkomend rekening houdt met een schatting van de totale kost tot aan het einddoel. Op deze manier convergeert het algoritme richting het einddoel en onderzoekt het geen punten die niet in de richting van het doel liggen. Tot slot maakt een bidirectionele versie van de besproken algoritmes tegelijk gebruik van zowel de voorwaartse als de achterwaartse versie om elkaar tegemoet te komen. Dit reduceert de te onderzoeken punten waardoor de rekentijd afneemt.

De tweede sectie besprak de verschillende manieren waarop kaarttransformaties een continue kaart transformeren naar een discrete kaart van de omgeving waarin een zoekalgoritme een pad kan zoeken. Exacte transformaties zijn compleet maar kosten meer tijd terwijl benaderende transformaties minder tijd kosten maar details van de ruimte kunnen missen. Door de complexiteit van de achterliggende algoritmes zijn exacte transformaties echter moeilijk implementeerbaar. De benaderende aanpak van de PRM daarentegen is een methode die goed toepasbaar is in een ruimte waarin meerdere padplanningsproblemen opgelost moeten worden. Verder is de variabele celdecompositie een veelgebruikte methode doordat het instaat is een gedetailleerde kaart op te bouwen zonder gebruik te maken van een groot aantal kleine cellen die het opslaggeheugen onnodig verhogen. Tot slot zijn potential field en RDT verschillend van de reeds eerder besproken kaarttransformaties doordat ze niet enkel de kaart transformeren maar er ook een pad in plannen. Hiervan is RDT en meer specifiek RRT een veelgebruikte techniek doordat deze snel implementeerbaar is.

De derde sectie besprak de mogelijke technieken om lokaal, onvoorziene obstakels op het globale pad te vermijden. De eenvoudigste manier is het bug algoritme waarbij de robot de contour van een obstakel volgt. Het bug algoritme houdt echter enkel rekening met de meest recente sensordata en houdt geen rekening met de kinematische beperkingen van de robot. De vector field diagram (VFD) techniek houdt rekening met meerdere sensordata om een polair histogram op te stellen van de waarschijnlijkheid dat een obstakel zich bevindt rondom de robot. Op basis hiervan maakt de robot een keuze voor de richting waarin het gaat navigeren. De VFD-techniek houdt echter geen rekening met de dynamica van de robot. Tot slot houdt de Dynamic Window Approach (DWA) wel rekening met de kinematische en dynamische beperkingen van de robot. Dit doet het door de mogelijke volgende snelheden weer te geven in een afbakening in de snelheidsruimte. Vervolgens kiest de robot een volgende snelheid uit de afbakening met behulp van een keuzefunctie omtrent de ligging van het doel en van de obstakels. Deze DWA kent een eenvoudige implementatie in het ROS-raamwerk.

De laatste sectie besprak de verschillende padplanningsmethoden voor een mobiele manipulator met een groot aantal en vooral redundante vrijheidsgraden. Hierin bestaan twee verschillende denkwijzen omtrent het controleren van het robotgedrag. Enerzijds de beredeneerde, vooropgeplande modelgebaseerde aanpak die nood heeft aan een exacte weergave van de omgeving en van de te manipuleren objecten. Anderzijds heeft de reactieve behavior-based aanpak minder nood aan dergelijke exacte modellen. De modelgebaseerde aanpak bestaat uit twee soorten strategieën waarbij de ene soort een opsplitsing of aanpassing van de volledige configuratieruimte hanteert terwijl de andere soort de volledige configuratieruimte van de mobiele manipulator in zijn geheel beschouwt. Uit de literatuur blijkt echter een gebrek aan open-source, praktische implementaties van gecombineerde padplanningsmethoden. Verder zit de meerderheid van de methoden nog in onderzoeksfasen. Dit toont dat mobiele manipulatoren nog niet industrieel toepasbaar zijn wegens hun complexe besturing.

Verschillende varianten van deze denkwijzen van padplanningsmethoden werden besproken en vervolgens vergeleken op basis van de belangrijkste criteria en de toepasbaarheid voor deze masterproef. Hieruit blijken de BI²RRT* methode en de traditionele, ontkoppelde aanpak het meest gunstig als padplanningsmethoden voor AMBER binnen deze masterproef op basis van de toepasbaarheid voor de masterproef (achtergrondkennis, open-source software en haalbaarheid) en de redundantieoplossingen die ze bieden voor mobiele manipulatoren met meerdere vrijdheidsgraden. De code van de BI²RRT* methode moet volledig aangepast worden voor een niet-holonomische robot wat niet eenvoudig blijkt waardoor de traditionele aanpak makkelijker implementeerbaar is. Deze masterproef maakt gebruik van de ontkoppelde padplanningsmethode. Hierbij onderzoekt het in volgende hoofdstukken in welke maten deze methode volstaat voor een scanopdracht van een muur als use case voor AMBER.

3 Simulatie van het robotmodel in het ROS-raamwerk

Dit hoofdstuk beschrijft de voorbereiding en de opbouw van het robotmodel van AMBER om er zodanig simulaties mee uit te voeren in het ROS-raamwerk. Dit bevat de beschrijving van de koppeling tussen verschillende processen en software in het ROS-raamwerk, de opbouw van het robotmodel en de aansturing en padplanning van mobiele basis en manipulator in simulatie.

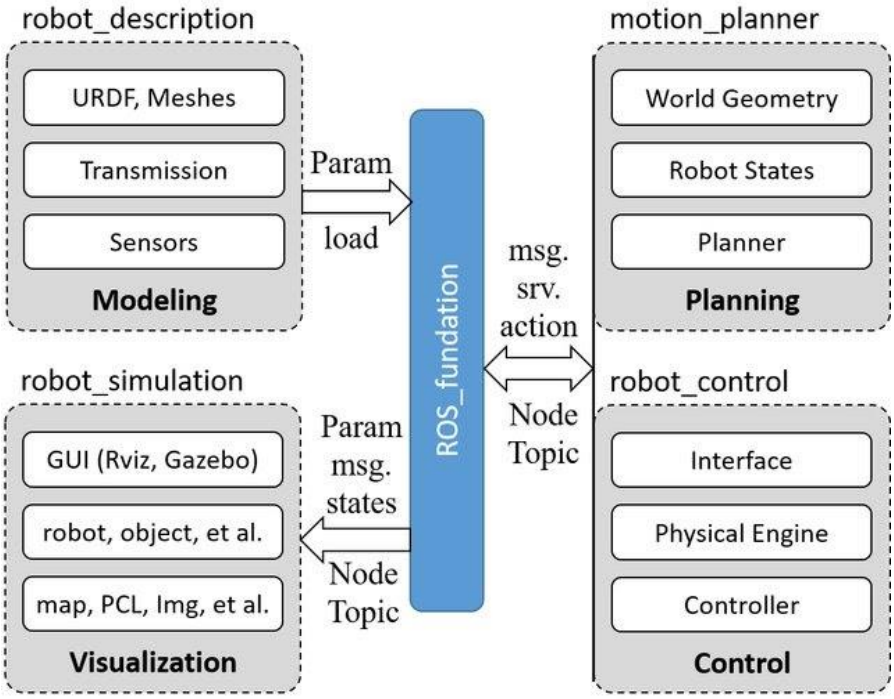
3.1 Koppeling tussen RViz en Gazebo in ROS

Zoals beschreven in [40] maakt ROS gebruik van *nodes*: processen die computationele taken uitvoeren (zoals het mappen van de omgeving, uitvoeren van een padplanning…) geschreven in C++ of Python. Nodes communiceren met elkaar via *messages* die bestaan uit een bepaalde datastructuur van tekst. Deze communicatie gebeurt over een *topic* (onderwerp) waarop nodes *publishen* om messages te verzenden en andere nodes op *subscriben* om messages te ontvangen. De datastroom van messages over een topic verloopt continu, zoals het bijhouden van sensordata of de robotstatus. Bijkomend voorziet een *service* de nodes van *request/reply* interacties om de kortstondige status van een node op te vragen of snelle berekeningen uit te voeren. De *actions* voeren acties uit die over een langere periode lopen en voorzien feedback over de uitvoering ervan, zoals de robot een commando geven en doen bewegen naar een bepaalde positie. Deze actions zijn *client-server* gebaseerd waarvoor de *actionlib* bibliotheek gereedschappen voorziet om een server te starten die de gewenste goals van een client uitvoeren [41]. De communicatie tussen client en server gebeurt met de acties *goal* (zendt een doel om uit te voeren), *result* (toont het resultaat na het uitvoeren van het doel) en *feedback* (toont de status van het uit te voeren doel). Tot slot ordent ROS alle software in *packages*, een soort van map horend bij een bepaald proces (zoals 2D-navigatie van de robot) [40]. Deze packages bevatten alle nodige bestanden voor het proces zoals o.a. de nodes, bibliotheken en configuratiebestanden. Verschillende packages zijn vrij te downloaden van ontwikkelaars of zelf te maken.

Het ROS-raamwerk maakt communicatie tussen verschillende robots en robotsoftware zoals *Gazebo* en *RViz* mogelijk. RViz of *ROS Visualiser* is de primaire 3D visualisatiemethode voor ROS om topics en dergelijke visueel voor te stellen [42]. Hierin kan men in een virtuele omgeving start- en eindposities voor robotactiviteiten opgeven, verschillende padplanners uitproberen en de resultaten hiervan visualiseren. De *MoveIt! plugin* voor RViz is een raamwerk dat toelaat om padplanning uit te voeren voor manipulatoren [43]. ROS zorgt voor de koppeling tussen de commando's in RViz en de uitvoering van de bewegingen van een aangesloten robot. Zoals reeds vermeld kunnen deze commando's ook door actions uitgevoerd worden. RViz zendt onderliggend ook dezelfde messages naar het betreffend topic met behulp van actions wanneer men gebruik maakt van de visuele tools. Een robot hoeft echter niet fysiek aanwezig te zijn. Gazebo is een *physics engine* dat de omgeving en het gedrag van een robotmodel driedimensionaal simuleert [44]. Zo is het mogelijk om in Gazebo een omgeving op te bouwen en de robot sensordata ervan te laten opnemen. RViz luistert (via ROS) naar deze data en visualiseert het. Hierna is het mogelijk om met behulp van ROS in RViz een kaart op te bouwen op basis van de sensordata waarin een gewenste beweging opgegeven wordt. ROS gekoppelde algoritmes (in de vorm van actions) en MoveIt! zorgen voor de padplanning van deze gewenste beweging doorheen de RViz omgeving (opgebouwd uit de omgeving in Gazebo). Tot slot gebeurt dankzij ROS dezelfde beweging van het robotmodel doorheen de simulatieomgeving in Gazebo. Elke software heeft duidelijk zijn eigen taak. Allereest

simuleert Gazebo de robot en de omgeving. Vervolgens laat ROS toe om Gazebo met andere software te doen communiceren. Verder zorgen padplanningsalgoritmes (via actions) en MoveIt voor bewegingstaken die gevisualiseerd worden in RViz. Tot slot gebeurt via ROS de communicatie met Gazebo om het robotmodel in de simulatieomgeving te doen bewegen.

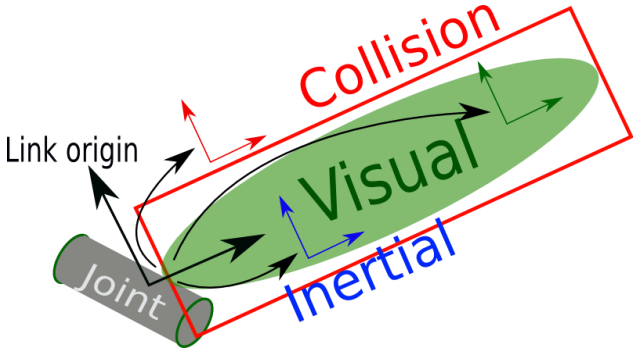
Onderstaande Figuur 3.1 toont de samenwerking tussen de verschillende processen en software voor de besturing en simulatie van een robot in het ROS-raamwerk. Volgende secties van dit hoofdstuk bespreken de belangrijkste componenten hiervan.



Figuur 3.1: Schematische voorstelling van de samenwerking tussen verschillende processen en software voor de besturing en simulatie van een robot met behulp van het ROS-raamwerk [45].

3.2 Beschrijving robotmodel in het Unified Robot Description Format (URDF)

Het *Unified Robot Description Format* of URDF is een bestand dat een robotmodel beschrijft in *XML*-formaat. Het URDF-bestand, ook de *robot description* genoemd, bestaat hoofdzakelijk uit *links* die de gelederen van de robot beschrijven en uit *joints* die de onderlinge relaties en vrijheidsgraden tussen de links beschrijven. Figuur 3.2 toont het verband tussen joints en links. Bovendien is op deze figuur te zien dat een link extra informatie onder de vorm van *collison*, *visual* en *inertial* elementen kan bevatten. Het XML-gebaseerd formaat van het URDF laat toe om eenvoudig informatie toe te voegen indien de toepassing dit vereist [46].

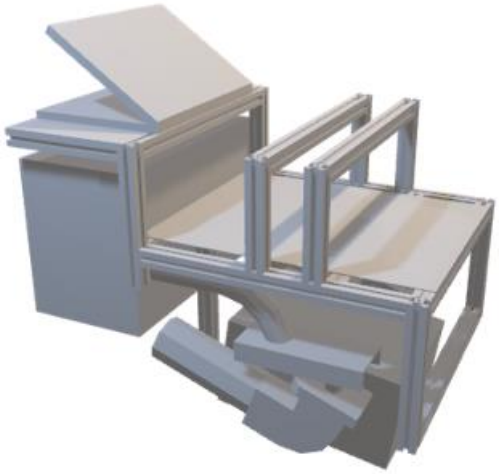


Figuur 3.2: Voorstelling van de collision, visual en inertial elementen van een link verbonden aan een joint [47].

Een visual element geeft informatie over hoe de link er visueel uitziet. Dit kunnen simpele vormen zijn zoals balken en cilinders of een geïmporteerde *mesh* (van een CAD-model) van een complexe vorm. Een collision element is noodzakelijk tijdens de berekeningen die nagaan of het model in botsing is met zichzelf of de wereld. Deze kan bestaan uit eenzelfde of verschillende vorm of mesh als het visual element. Een inertial element van een link geeft informatie over het massamiddelpunt, de massa en de traagheidsmomenten van die link. Het mobiele basis gedeelte van het bestaande URDF-bestand van AMBER bevatte enkel collison en visual elementen. Het manipulator gedeelte daarentegen beschikte reeds over collison, visual en inertial elementen van de verschillende manipulatorlinks afkomstig. Om AMBER correct te simuleren in de physics engine Gazebo ontbraken de inertial elementen in de links van het basisframe en de wielen. Wanneer bijvoorbeeld de traagheidsmomenten nul zijn, leidt dit tot een oneindige versnelling in Gazebo bij het aanleggen van een eindig moment. Bijgevolg viel tijdens het simuleren in Gazebo het robotmodel door de grond door een gebrek aan inerties [48].

Het originele URDF-bestand beschreef elk onderdeel (stangen, platen…) van het basisframe als een aparte link. Doordat het frame een statische constructie is, is het niet praktisch om voor elk van deze onderdelen het massamiddelpunt, de massa en de traagheidsmomenten apart te bepalen. Bijgevolg is het raadzaam om het volledige basisframe als één link te beschouwen. *CAD*-programma`s, zoals *PTC Creo*, zijn instaat om deze eigenschappen te berekenen van complexe constructies. Het basisframe is eerst opgemeten en daarna getekend in PTC Creo. Vervolgens zijn aan alle onderdelen de juiste massa's en materiaaleigenschappen toegekend. Creo berekent aan de hand van dit CAD-model de gewenste parameters van het gehele basisframe. Bovendien is een mesh van het CAD-model gebruikt voor de visualisatie en collision van het basisframe link. Onderstaande Figuur 3.3 toont deze mesh van het basisframe. De wielen zijn wel apart gewogen, gemeten en daarna toegevoegd in het URDF met een cilindervorm. Tot slot zijn alle inerties en

massamiddelpunten toegevoegd aan de inertial elementen van de wielen en het basisframe waarna Gazebo het model correct kon weergeven.



Figuur 3.3: CAD-model van het basisframe van AMBER in STL-bestandsformaat.

Onderstaande Figuur 3.4 toont een stuk uit het uit het URDF-bestand als voorbeeld van de opstelling van links en joints om het robotmodel te beschrijven.

src > amber_description > urdf > acro_mobile_manipulator.urdf.xacro

```
<link name="base_link">
    <origin xyz="0 0 0" rpy="0 0 0" />
</link>

<link name="base">
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://amber_description/meshes/basis/visual/amber.dae"/>
        </geometry>
    </visual>
    <inertial>
        <mass value="30"/>
        <origin rpy="0 0.0 0.0" xyz=" -0.15 0.0 -0.1"/>
        <inertia ixx="4.2088082" ixy="-0.000036205230" ixz="2.4055082"
        iyy="7.2572353" iyz="0.00016257984" izz="6.7000177"/>
    </inertial>
    <collision name="collision">
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://amber_description/meshes/basis/collision/amber.stl"/>
        </geometry>
    </collision>
</link>

<joint name="base_footprint_joint" type="fixed">
    <origin xyz="0.0 0.0 0.1625" rpy="0 0 0" />
    <parent link="base_link" />
    <child link="base" />
</joint>
```

Figuur 3.4: Deel uit het URDF-bestand als voorbeeld van de opstelling van links en joints om het robotmodel te beschrijven.

Verder zijn er aan het simulatiemodel sensoren toegevoegd. Door gebruik te maken van een *gazebo_sensor_plugin* binnen het URDF-bestand simuleert Gazebo de sensoren en publiceert het de sensordata op het gewenste topic. De gesimuleerde sensordata is van dezelfde vorm als sensordata

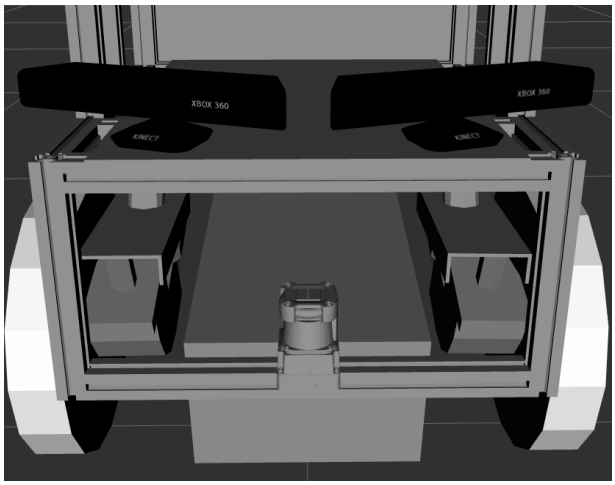
van een echte sensor. Hierdoor is de verdere verwerking van de gesimuleerde sensordata analoog aan de verwerking van sensordata uit de reële wereld.

De gebruikte sensoren zijn een *Hokuyo 2D-laserscanner* en twee *Microsoft Kinect 3D-camera's*. Dit gebeurt respectievelijk met de *ray plugin* en de *depth camera plugin*. De parameters van deze plugins zijn zo ingesteld in het URDF dat ze de reële sensoren simuleren. Zo is het ook mogelijk om ruis op deze sensordata te simuleren. De ray plugin zendt sensordata van de lasersensor naar een *LaserScan* topic genaamd */hokuyo/laser/scan*. Deze data beschrijft de afstand van de sensor tot een obstakel onder een bepaalde hoek. De depth camera plugin zendt sensordata van een 3D-camera naar een *PointCloud* topic genaamd *kinect/depth/points*. Deze data beschrijft een driedimensionale omgeving als een puntenwolk. Deze communicatie gebeurt in beide gevallen in de vorm van *sensor_msgs* berichten. Het robotmodel van AMBER maakt gebruik van twee 3D-camera's om een breder gezichtsveld te hebben voor het opbouwen van de driedimensionale botsingsomgeving. Onderstaande Figuur 3.5 en Figuur 3.6 tonen respectievelijk een gedeelte uit het URDF dat de sensoren definieert en de sensoren op het robotmodel in RViz.

src > amber_description > urdf > acro_mobile_manipulator.urdf.xacro

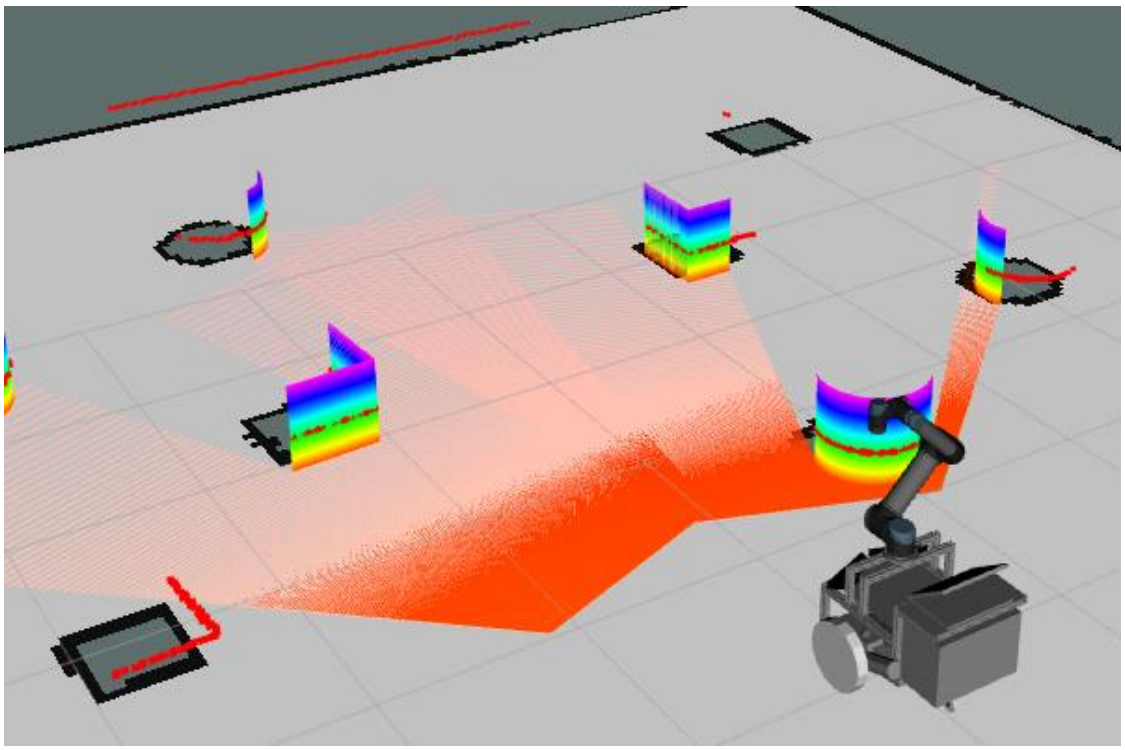
```
548     <!-- Description of sensors mounted on the ACRO mobile platform -->
549     <!-- Kinect basis -->
550     <gazebo reference="kinect">
551         <sensor type="depth" name="kinect_camera">
552             ...
553             <plugin name="kinect_controller" filename="libgazebo_ros_openni_kinect.so">
554                 ...
555             </plugin>
556         </sensor>
557     </gazebo>
558 >   <link name="kinect">…
576     </link>
577 >   <joint name="base_kinect" type="fixed">…
581     </joint>
582 >   <link name="kinect_optical_link">…
594     </link>
595 >   <joint name="kinect_optical_joint" type="fixed">…
599     </joint>
600
601     <!-- Hokuyo Laser -->
602     <gazebo reference="hokuyo">
603         <sensor type="ray" name="head_hokuyo_sensor">
604             ...
605             <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
606                 ...
607             </plugin>
608         </sensor>
609     </gazebo>
610 >   <link name="hokuyo">…
628     </link>
629 >   <joint name="hokuyo_joint" type="fixed">…
634     </joint>
```

Figuur 3.5: Deel van het URDF-bestand dat de Kinect camera en de Hokuyo laserscanner beschrijft met de nodige Gazebo plugins, links en joints.



Figuur 3.6: De Hokuyo lasersensor (onderaan) en de twee Kinect 3D-sensoren (bovenaan) op het robotmodel van AMBER in RViz.

Onderstaande Figuur 3.7 toont het robotmodel van AMBER in RViz met de sensordata gevisualiseerd. Hierin zijn de rode punten rond obstakels de afstandspunten van de Hokuyo lasersensor tot de obstakels en de gekleurde gebieden de puntenwolken van de Kinect 3D-sensoren met de RGB-kleurenverdeling als voorstelling van de hoogte in de ruimte.

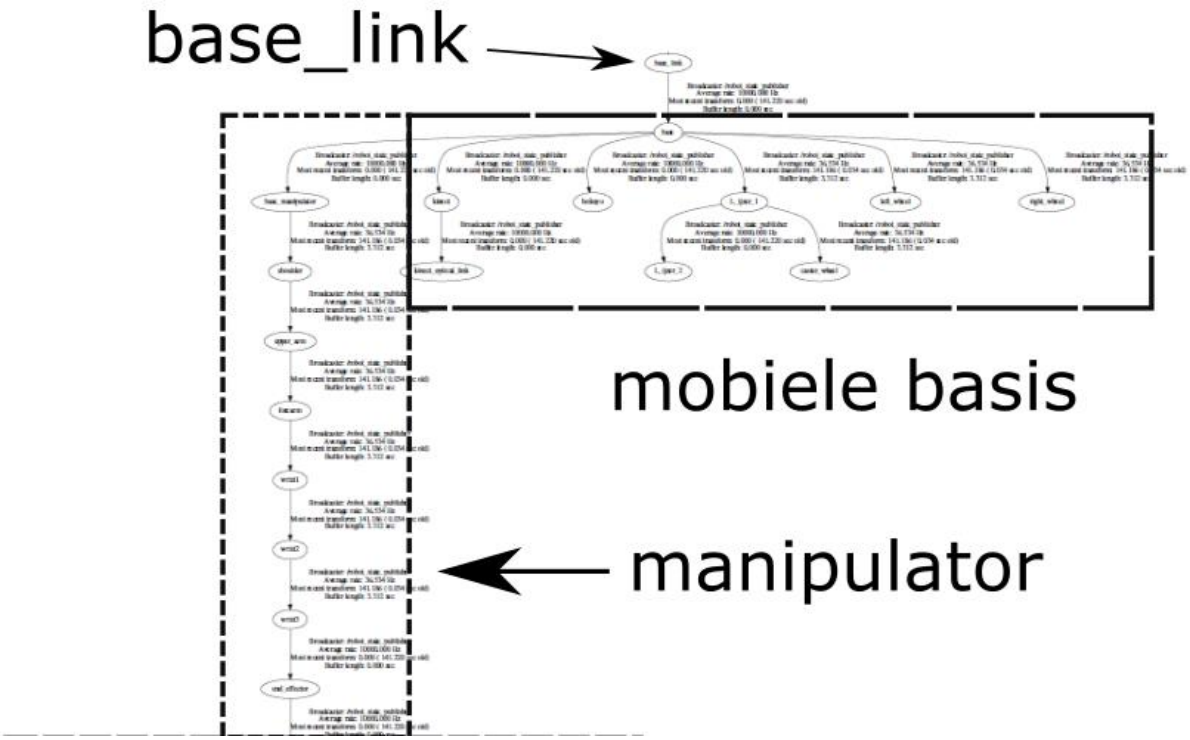


Figuur 3.7: Het robotmodel van AMBER in RViz met de sensordata gevisualiseerd. Hierin zijn de rode punten rond obstakels de afstandspunten van de Hokuyo lasersensor tot de obstakels en de gekleurde gebieden de puntenwolken van de Kinect 3D-sensoren met de RGB-kleurenverdeling de voorstelling van de hoogte in de ruimte.

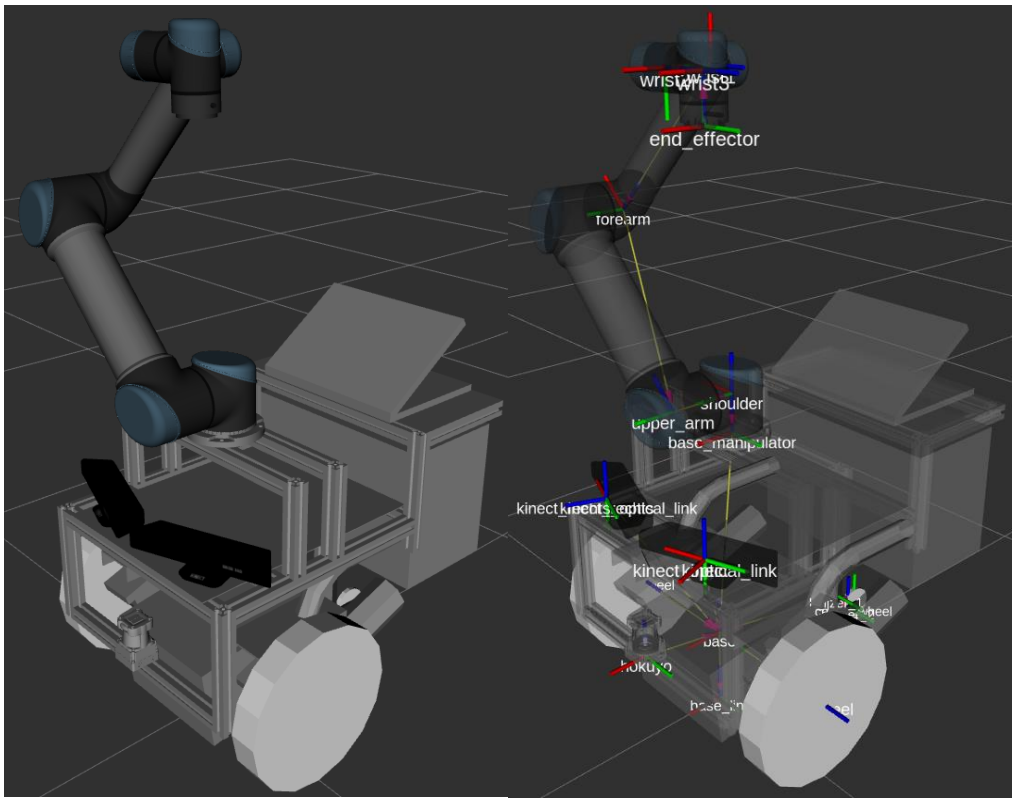
Na het opstellen van de robot description staat het package *joint state publisher* in voor het vrijgeven van de statussen van alle joints van de robot [49]. De joint state publisher publisht deze statussen van een gegeven URDF op het topic *joint states* via semsor_msgs berichten. Hierna gebruikt het *robot state publisher* package de informatie van het topic joint states en van de robot description om

de voorwaartse kinematica te berekenen van de driedimensionale relaties tussen de linkassestelsels en geeft deze vrij in de vorm van een kinematische boomstructuur van de robot (zoals gedefinieerd in het URDF) aan het topic *tf* [50]. Het gelijknamig package *tf* (*transform frames*) publisht het tf topic dat instaat voor het bijhouden van de onderlinge transformaties tussen de assenstelsels doorheen de tijd in de vorm van de kinematische boomstructuur [51]. Bovendien laat het topic tf toe om punten en vectoren tussen twee assenstelsels te transformeren op elk moment in de tijd. Op deze manier is de volledige status van de robot doorheen de tijd gekend.

Onderstaande Figuur 3.8 en Figuur 3.9 tonen respectievelijk de visualisatie van de kinematische boomstructuur van de verschillende assenstelsels van de links van AMBER door het tf package en de visualisatie van het robotmodel van AMBER in RViz met de assenstelsels van de links en de transformaties ertussen (geel).



Figuur 3.8: Visualisatie van de kinematische boomstructuur van de verschillende assenstelsels van de links van AMBER door het tf package.



Figuur 3.9: Visualisatie van het robotmodel van AMBER in RViz met (rechts) de assenstelsels van de links en de transformaties ertussen (geel).

Tot slot genereert een grafische *user interface* van MoveIt!, genaamd de *MoveIt! Setup Assistant*, uitgaand van het URDF de bijkomende configuratiebestanden van de robot die noodzakelijk zijn voor het gebruik van MoveIt! [52]. Het belangrijkste hiervan is het *Semantic Robot Description Format (SRDF)*. Dit is een XML-gebaseerd bestand dat instaat voor de semantische beschrijving van bepaalde robotonderdelen zoals groepen van links en joints (bv. de manipulator), definiëring van de eindeffector, het overslaan van de botsingsdetectie tussen bepaalde links op basis van de *collisionmatrix* enzovoort [53].

3.3 Aansturing mobiele basis en manipulator in simulatie met behulp van controllers

In werkelijkheid beweegt een robot door de actuatoren van de wielen en arm aan te sturen. Zoals reeds vermeld zorgt Gazebo voor de modellering van de robot en zijn omgeving. Er is dus ook nood aan modellering van de actuatoren. Dit gebeurt door *transmission* elementen in de URDF die de werkelijke hardware nabootsen. Alle door actuatoren aanstuurbare joints hebben een bijhorende transmission van een bepaald *hardware interface type*. ROS gebruikt de hardware interfaces om soortafhankelijke commando's te sturen naar de hardware [54], [55]. Zo hebben de wielen een transmission van het type *VelocityJointInterface* om een bepaalde rotatiesnelheid op te geven en de andere gewrichten een *PositionJointInterface* type transmission om de joints een bepaalde positie te doen innemen. Onderstaande Figuur 3.10 toont een voorbeeld van een transmission als modellering van een actuator.

```
<transmission name="right_wheel_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="right_wheel_joint">
        <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    </joint>
    <actuator name="right_wheel_motor">
        <hardwareInterface>VelocityJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

Figuur 3.10: De transmission right_wheel_trans met een hardware interface van het type VelocityJointInterface horend bij de right_wheel_joint.

De zogenaamde *controllers* uit het *ros_control* package staan in voor de aansturing van de gesimuleerde hardware [56], [57]. De controllers, samen met de *gazebo_ros_control plugin* in de URDF, laten toe om de nagebootste hardware van de robot via ROS aan te sturen in Gazebo op dezelfde manier als de ROS-aansturing van de werkelijke actuatoren [55].

Drie controllers staan in voor de aansturing van het volledig robotmodel van AMBER. Allereerst zendt de *joint_state_controller* de status van elke bron van data geregistreerd op een *hardware_interface* (werkelijke actuatoren of zoals hier transmissions) naar het topic *JointState* met *sensor_msgs* berichten. Dit bericht bevat data dat de status beschrijft van een momentgecontrolleerde joint en bestaat uit de positie van een joint, de snelheid van een joint en het aangelegd moment op de joint. Op deze manier is op elk moment de status van elke joint gekend.

Verder controleert de *diff_drive_controller* de wielen van de mobiele basis [58]. Dit doet het door allereerst berichten te ontvangen van het topic *cmd_vel*. Deze berichten zijn van het zogenaamde *twist* type dat een snelheid doorheen de ruimte beschrijft als een combinatie van een lineaire snelheid in de x, y en z en een rotatiesnelheid rond de x-, y- en z-as van de robot. In het geval van de niet-holonomische basis van AMBER bestaat dit bericht enkel uit een snelheid in de x richting en een bepaalde rotatie rond de z-as als hoekverdraaiing van de basis. De controller berekent de *odometrie* van de robot en stuurt de rotatiesnelheid van de wielen aan zodat de robot de juiste snelheid doorheen de ruimte aanneemt. Odometrie is een onderdeel van het gegist bestek betreffende de positieschatting van een robot in functie van de tijd door gebruik te maken van de opeenvolging van sensordata van de beweging van de wielen van de robot met behulp van wielsensoren genaamd *encoders*. Hiervoor heeft de controller zo correct mogelijke informatie nodig

over de afmetingen van de wielen en de afstanden ertussen. Deze informatie volgt uit de eerdere metingen van AMBER voor het CAD-bestand. Tot slot zendt de controller de odometriedata naar het topic *odom* en transformatiedata tussen het assenstelsel van de odometrie en de basislink van de robot (*base_link*) naar het topic tf.

Zo zet bijvoorbeeld het *teleop_twist_keyboard* package met behulp van een node drukken op bepaalde toetsen van een toetsenbord om naar twist messages op het cmd_vel topic. De diff_drive_controller luistert naar dit topic en beweegt de mobiele basis op basis van de aansturing met het toetsenbord. In een verder hoofdstuk volgt de implementatie van de *move_base* node die autonoom berichten zendt naar hetzelfde cmd_vel topic om een bepaald traject uit te voeren.

Tot slot controleert de *joint_trajectory_controller* het traject van een groep van joints doorheen de ruimte [59]. Meer bepaald staat de *position_controller* van het type *joint_trajectory_controller* in voor de positiebesturing van de groep van joints van de manipulator aangezien deze joints transmissions hebben van het hardware interface type PositionJointInterface. Deze controller ontvangt *trajectory_msgs* berichten van het topic *JointTrajectory* om het traject van een verzameling van punten uit te voeren. De arm is op deze manier aanstuurbaar met de MoveIt! plugin voor RViz dat instaat voor de padplanning ervan.

Onderstaande Figuur 3.11 Figuur 3.12 tonen respectievelijk het configuratiebestand van de ROS-controllers voor AMBER en het opstartbestand waarin de *controller_spawner* node van het package *controller_manager* de verschillende controllers opstart.

src > amber_moveit_config > config > ! ros_controllers.yaml > {} acro_mobile_manipulator

```yaml
acro_mobile_manipulator:
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50
  arm_controller:
    type: position_controllers/JointTrajectoryController
    joints:-
    gains:-

  mobile_base_controller:
    type        : "diff_drive_controller/DiffDriveController"
    left_wheel  : 'left_wheel_joint'
    right_wheel : 'right_wheel_joint'
    publish_rate: 50.0               # default: 50
    pose_covariance_diagonal : [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
    twist_covariance_diagonal: [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
```

Figuur 3.11: Deel van het configuratiebestand van de ROS-controllers joint_state_controller, arm_controller en mobile_base_controller voor AMBER.

src > amber_moveit_config > launch > ros_controllers.launch

```xml
<?xml version="1.0"?>
<launch>

  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find amber_moveit_config)/config/ros_controllers.yaml" command="load"/>
  <rosparam file="$(find amber_moveit_config)/config/gazebo_controllers.yaml" command="load"/>
  <!-- Load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" ns="/acro_mobile_manipulator"
          args="--namespace=/acro_mobile_manipulator
          joint_state_controller
          arm_controller
          mobile_base_controller
          --timeout 20"/>
</launch>
```

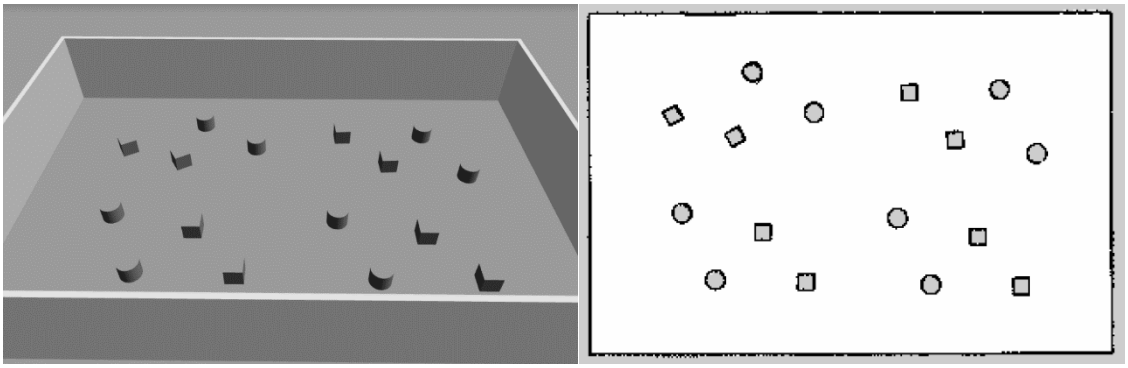
Figuur 3.12: Opstartbestand waarin de controller_spawner node de verschillende controllers opstart.

3.4 Ontkoppelde padplanning van de mobiele basis en de manipulator

3.4.1 Mobiele basis: implementatie van de navigation stack

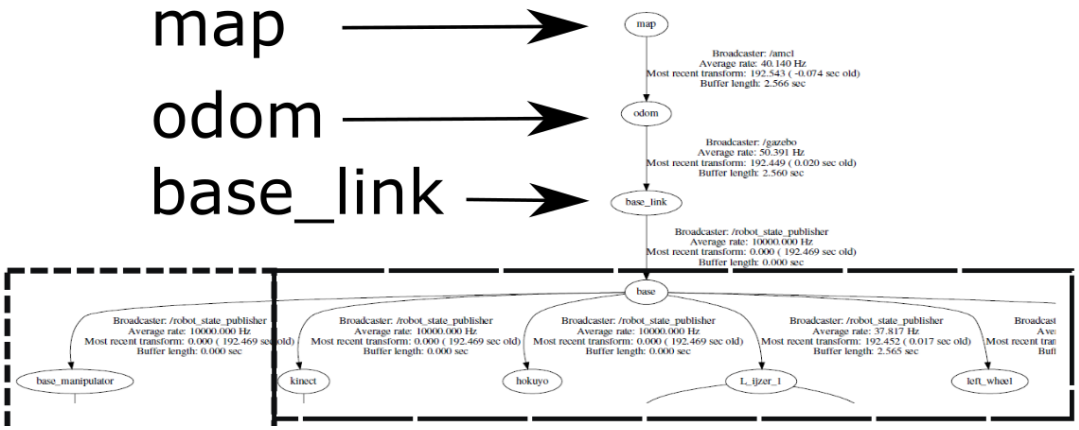
De *navigation stack* van ROS is een package dat toelaat om 2D-padplanning te implementeren voor een mobiele robot [60]. Hiervoor moet het robotsysteem reeds aan enkele zaken voldoen. Allereerst moet het systeem met behulp van tf informatie over de onderliggende transformaties publishen. Ten tweede gebruikt de navigation stack sensordata van *sensor_msgs* messages van een LaserScan of PointCloud topic om obstakels in de omgeving te ontwijken. Vervolgens moet informatie over de odometrie van het topic odom via de tf gekend zijn. De navigation stack zendt twist berichten naar het cmd_vel topic dat aanwezig moet zijn. Deze vereisten zijn tot hiertoe reeds geïmplementeerd en in voorgaande hoofdstukken uitgelegd. Tot slot luistert de navigation stack naar het topic *map* als input van een kaart om padplanning in uit te voeren.

De navigation stack gebruikt een *2D-occupancy grid map* als kaart. Dit stelt de omgeving waarin de robot navigeert voor als zwarte gebieden voor obstakels, witte gebieden voor vrije ruimte en grijze gebieden voor nog niet gekende ruimte. Het *gmapping* package maakt gebruik van sensordata (in dit geval van de Hokuyo lasersensor) en van de odometrie van de robot om tijdens het rondrijden zulke kaart op te bouwen [61]. Het gmapping package maakt gebruik van *Simultaneous Localisation And Mapping (SLAM)* algoritmes op basis van *particle filters (PF)* en van de huidige kaarttoestand om zich in de kaart te lokaliseren die het gelijktijdig opbouwt [62]. Hierna publisht het de kaartinformatie op het topic map via *nav_msgs/OccupancyGrid* berichten. Deze informatie kan tijdens het rondrijden van de robot rechtstreeks gebruikt worden als input voor de navigation stack. Anderzijds kan het package *map_server* de met gmapping opgestelde kaart opslaan en achteraf publishen op het topic map. In beide gevallen luistert de navigation stack naar hetzelfde topic map en gebruikt het, op lokalisatiefouten na, dezelfde kaart van dezelfde omgeving. Deze masterproef maakt gebruik van de tweede aanpak aangezien het veronderstelt dat een op voorhand gekende omgeving in kaart is gebracht. Onderstaande Figuur 3.13 toont een kamer met cilinder- en kubusvormige obstakels als simulatieomgeving in Gazebo en de 2D-occupancy grid map ervan opgebouwd door gmapping gebruik makend van de lasersensor.



Figuur 3.13: (links) Kamer met cilinder- en kubusvormige obstakels als simulatieomgeving in Gazebo en (rechts) de 2D-occupancy grid map ervan opgebouwd door gmapping gebruik makend van de lasersensor.

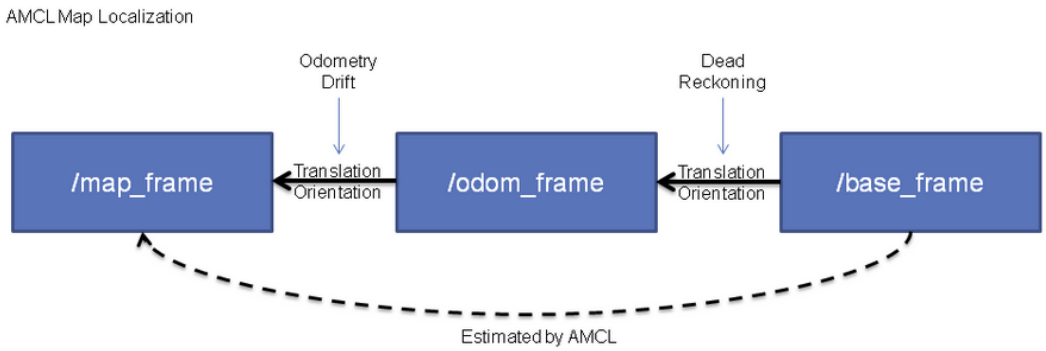
Onderstaande Figuur 3.14 toont de kinematische boomstructuur van de verschillende assenstelsels van de links van AMBER door het tf package na koppeling met de kaart.



Figuur 3.14: Visualisatie van de kinematische boomstructuur van de verschillende assenstelsels van de links van AMBER door het tf package na koppeling met map.

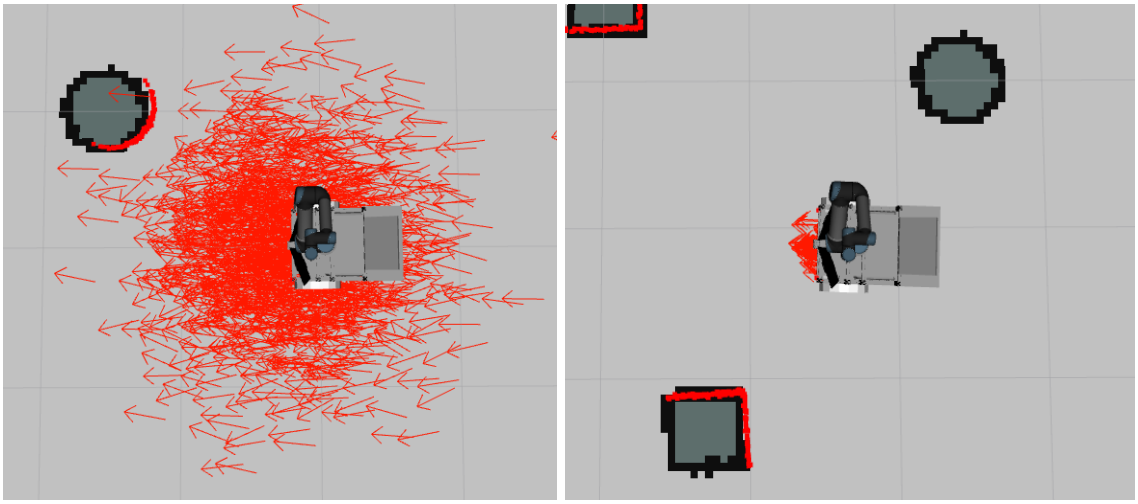
Bij gebruik van een a priori gekende kaart vereist de navigation stack een algoritme voor de lokalisatie van de robot in de kaart. De meest courante methode voor de navigation stack is de *Adaptive (of Augmented) Monte Carlo localization (AMCL)* aanpak, alsook *KLD-sampling* genoemd, met behulp van het *AMCL-package* [63]. Dit is een probabilistisch, samplegebaseerd lokalisatiealgoritme dat alle mogelijke poses in een kaart bemonstert en aanpast met een particle filter na een gekende nieuwe beweging van de odometrie in de kaart en de correctie ervan op basis van sensordata [64]. Deze adaptieve versie van de Monte Carlo lokalisatie zorgt ervoor dat de nieuwe bemonstering in elke iteratie zich beperkt tot een gebied rond de positie die tijdens de vorige iteratie de hoogste waarschijnlijkheid had. Op deze manier convergeert het algoritme sneller en is het minder computationeel zwaar dan de niet-adaptieve versie. Bijkomend is de adaptieve versie in staat om lokalisatiefalen te herstellen door willekeurig nieuwe mogelijke punten toe te voegen aan de verzameling van bemonsterde punten.

De *amcl* node van het *amcl* package maakt gebruik van laserscandata in de vorm van sensor_msgs/LaserScan berichten van het topic /hokuyo/laser/scan, van de transformaties tussen assenstelsels van het topic tf (dat de transformatie van de odometrie bevat) en van de opgegeven kaart op het topic map. Hiermee publisht het enerzijds de geschatte robotpose op het topic *amcl_pose*. Anderzijds publisht het de transformatie tussen het assenstelsel van de odometrie en de map op het topic tf om zodanig de robotpose te koppelen met het assenstelsel van de kaart. Onderstaande Figuur 3.15 toont schematisch de functie van het lokalisatiealgoritme waarbij frame staat voor een assenstelsel. De transformatie tussen het *base_frame* van de robot en het odometrieassenstelsel gebeurt met odometrieberekeningen (*dead reckoning* = gegist bestek). Hierop zitten fouten (*odometry drift*) ten opzichte van de werkelijke positie van de robot in de kaart. AMCL schat de benaderende positie van de robot in de kaart zodat de verdere planningsalgoritmes geen gebruik moeten maken van de foute odometriepositie.



Figuur 3.15: Schematische voorstelling van de AMCL-aanpak dat de robotpositie schat in een gegeven kaart om zodanig de fouten afkomstig van de odometrie niet te hoeven gebruiken.

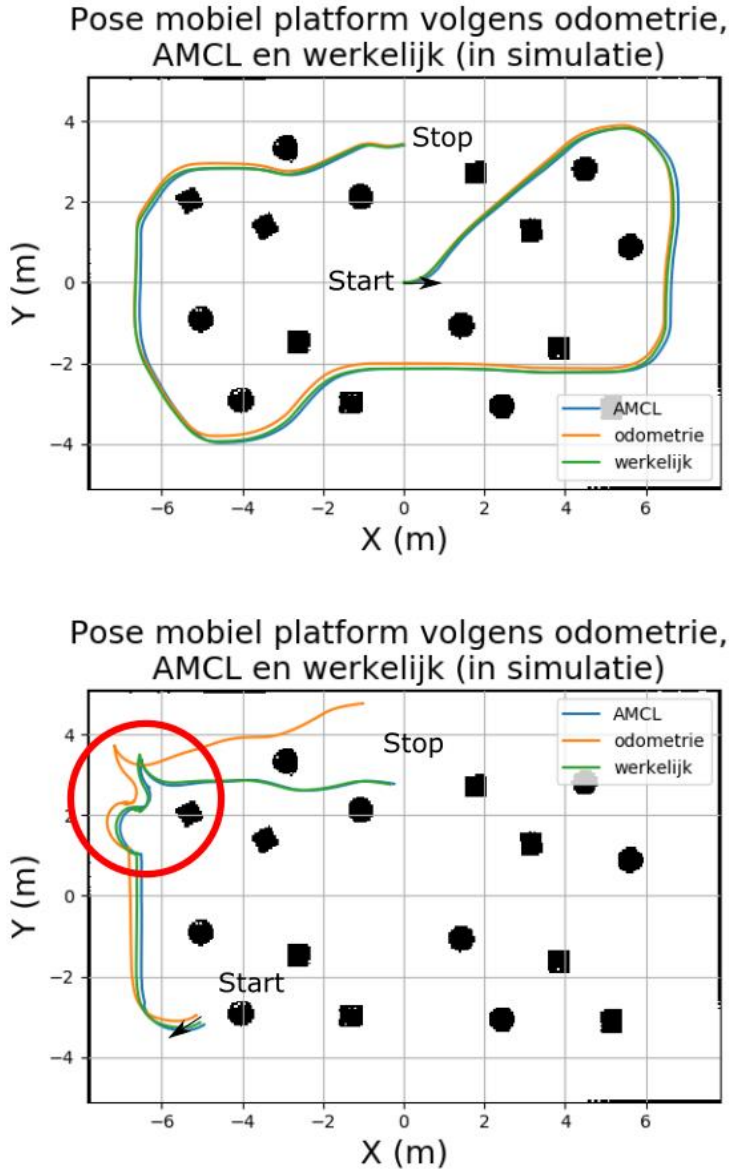
Bovendien tonen simulatietesten met AMBER dat de lokalisatie van de robot in een gegeven kaart met AMCL aanzienlijk sneller en accurater gebeurt dan de methode waarmee de gmapping node de kaart opbouwt tijdens het rondrijden. Onderstaande Figuur 3.16 toont enerzijds de visualisatie van AMBER in RViz met de geschatte initiële beginpositie van de mobiele basis door AMCL in de vorm van een verdeling van pijlen die de mogelijke poses aanduiden. Anderzijds toont het de door AMCL geschatte positie van de mobiele basis na bewegingen door de kaart en opname van nieuwe sensordata.



Figuur 3.16: (links) Visualisatie van AMBER in RViz met de geschatte initiële beginpositie met grote onzekerheid van de mobiele basis door AMCL in de vorm van een verdeling van pijlen die de mogelijke poses aanduiden. (rechts) De door AMCL geschatte positie van de mobiele basis na bewegingen door de kaart en opname van nieuwe sensordata waardoor de onzekerheid verkleint en de pose convergeert naar de werkelijke positie.

Onderstaande Figuur 3.17 toont twee voorbeelden van een uitgevoerd padplanningstraject van de mobiele basis dat de odometriepose en de AMCL-pose vergelijkt met de werkelijke pose in de simulatieomgeving Gazebo. Het toont enerzijds een eenvoudig traject met afwisseling tussen rotaties en rechte stukken waarbij de odometriepose en de AMCL-pose beide goed de werkelijke pose volgen. Anderzijds toont het een complexer traject met gevraagde tussenposes waarvoor verschillende rotaties noodzakelijk zijn omcirkelt in het rood. De odometrie toont grote fouten na de rotaties en rekent hiermee verder waardoor het richting het einde de werkelijke pose niet meer volgt. De AMCL-pose benadert de werkelijke pose relatief goed in het begin, heeft geen last van de rotaties en benadert de werkelijke pose het best naarmate het einde van het traject aangezien de

robot navigeert tussen verschillende objecten waarbij het meerdere bewegingen uitvoert en zijn odometrieberekeningen corrigeert met voldoende herkenningspunten.



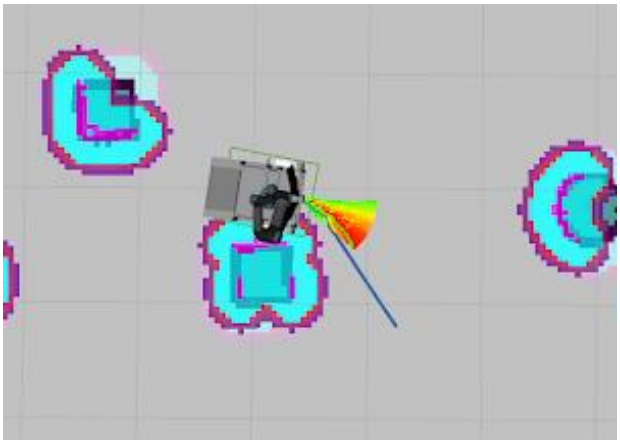
Figuur 3.17: Vergelijking tussen odometriepose, AMCL-pose en werkelijke pose in simulatie tijdens een padplanningstraject doorheen de simulatieomgeving. (boven) Een eenvoudig traject met afwisseling tussen rotaties en rechte stukken waarbij de odometriepose en de AMCL-pose beide goed de werkelijke pose volgen. Hierbij toont de odometrie lichte fouten na rotaties die in dit geval doorheen het traject gecompenseerd worden door rotaties naar de andere kant. (beneden) Een complexer traject. Hierbij zijn gevraagde tussenposes omcirkelt in het rood waarvoor verschillende rotaties noodzakelijk zijn. De odometrie toont grote fouten na de rotaties en rekent hiermee verder waardoor het richting het einde de werkelijke pose niet meer volgt. De AMCL-pose benadert de werkelijke pose relatief goed in het begin en heeft geen last van de rotaties. Naarmate het einde van het traject benadert de AMCL-pose de werkelijke positie het beste aangezien de robot navigeert tussen verschillende objecten waarbij het meerdere bewegingen uitvoert en zijn odometrieberekeningen corrigeert met voldoende herkenningspunten.

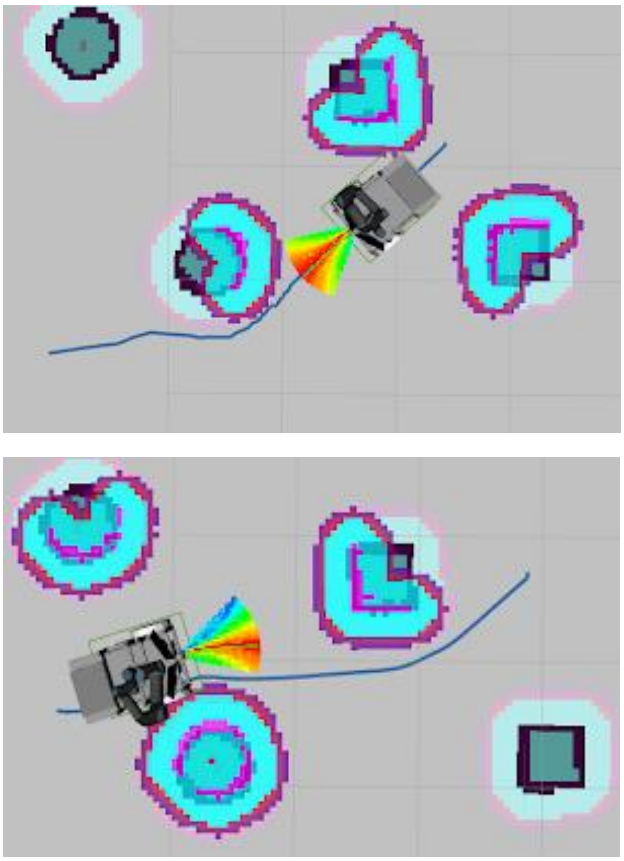
De *move_base* node van de navigation stack is het hoofdproces voor de padplanning van de mobiele basis. Het voorziet de configuratie en uitvoering van de navigation stack en voorziet een implementatie van ROS-acties om met de navigation stack te communiceren [65]. Dit betreft het

bereiken van een opgegeven doel voor de mobiele basis doorheen de kaart met behulp van de combinatie van een globale en lokale planner. De node onderhoudt de twee kostkaarten gebruikt voor de navigatie van de robot waarin informatie over obstakels zitten. De kostkaart voor de globale planner is afgeleid van de opgegeven 2D-kaart en de kostkaart voor de lokale planner van recente sensordata. Beide kaarten zorgen ervoor dat de planners rond obstakels plannen door een hoge kost rond de obstakels [66]. De move_base node luistert naar het actietopic *move_base/goal* of naar het topic *move_base_simple/goal* (geeft geen info over uitvoering van het proces) om een doel te ontvangen. Hierna laat het de padplanningsalgoritmes paden plannen en stuurt het twist berichten naar het topic cmd_vel om zodanig het doel te bereiken.

De move_base parameter *base_global_planner* geeft aan welke globale plannerplugin de navigation stack gebruikt die instaat voor het plannen van een optimaal pad tussen start- en doelpositie. De parameter *base_local_planner* duidt op de lokale plannerplugin die bepaalt hoe de robot zich gedraagt doorheen het geplande pad door de juiste commando's te sturen naar de wielcontroller.

Deze masterproef maakt gebruik van A* als globale planner met behulp van het *global_planner* package met parameter *use_dijkstra = False* [67] en van de DWA-methode als lokale planner met het package *dwa_local_planner* [68]. Onderstaande figuren tonen bovenaanzichten van padplanningsuitvoeringen van de mobiele basis van AMBER met de navigation stack in RViz. RViz laat toe om doelen te definiëren met visualisatietools maar dit is ook mogelijk door gebruik te maken van actions in aparte code. Op deze figuren zijn in de kaart tweedimensionale obstakels te zien als donkere gebieden met kleuren eromheen van de globale kostkaart en in de buurt van de robot fellere kleuren rond obstakels afkomstig van de lokale kostkaart. Verder stelt de blauwe lijn het globaal pad voor bekomen door A*, het kleurenveld voor de basis de kosten van de mogelijke aanneembare snelheden volgens de DWA lokale planner waarin de groene lijn de huidig gekozen volgende snelheid weergeeft. De lokale planner tracht zo goed mogelijk het globale pad te volgen rekening houdend met de hoge planningskost rond obstakels.





Figuur 3.18: Bovenaanzichten van padplanningsuitvoeringen van de mobiele basis van AMBER met de navigation stack in RViz. Hierin stelt de blauwe lijn het A-pad voor en het kleurenveld voor AMBER de kostenverdeling van de aanneembare snelheden volgens de DWA.*

Onderstaande Figuur 3.19 toont het opstartbestand van de navigation stack. Hierin laadt allereerst de *map_server* node de kaart. Vervolgens roept het bestand een ander opstartbestand op dat de amcl node start voor een mobiele basis met differentiële wielaandrijving. Tot slot start het de move_base node met bijhorende padplanners en kostkaarten.

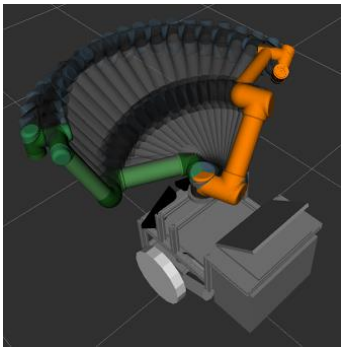
src > amber_2dnav > launch > move_base.launch

```
<launch>
    <master auto="start"/>
    <!-- Run the map server to load a pre built map (using slam_gmapping) -->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find amber_description)/maps/vierkante_kamer.yaml"/>

    <!--- Run AMCL localisation in a pre built map-->
    <include file="$(find amber_2dnav)/launch/amcl_diff.launch"/>

    <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

        <!-- Remap to correct cmd_vel of mobile base controller of AMBER -->
        <remap from="cmd_vel" to="/acro_mobile_manipulator/mobile_base_controller/cmd_vel" />

        <!-- Global and local planners with their parameters -->
        <param name="base_global_planner" value="global_planner/GlobalPlanner"/>
        <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS"/>
        <rosparam file="$(find amber_2dnav)/config/global_planner_params.yaml" command="load" />
        <rosparam file="$(find amber_2dnav)/config/dwa_local_planner_params.yaml" command="load" />

        <!-- Global and local costmaps with their parameters -->
        <rosparam file="$(find amber_2dnav)/config/costmap_common_params.yaml" command="load" ns="global_costmap" />
        <rosparam file="$(find amber_2dnav)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
        <rosparam file="$(find amber_2dnav)/config/local_costmap_params.yaml" command="load" />
        <rosparam file="$(find amber_2dnav)/config/global_costmap_params.yaml" command="load" />
    </node>
</launch>
```

Figuur 3.19: Het opstartbestand van de navigation stack dat de kaart inlaadt, de AMCL start, de kostkaarten inlaadt en de padplanningsalgoritmes initialiseert.

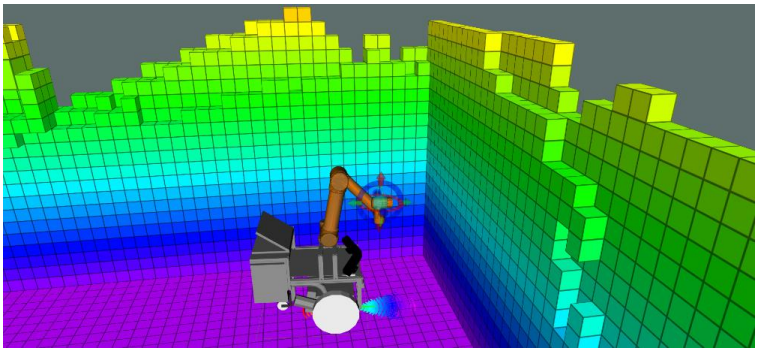
3.4.2 Manipulator: planning group en OMPL in MoveIt!

De Setup Assistant van MoveIt! laat toe om een *planning group* van de manipulator te definiëren in de SRDF [52]. Deze planning group beschrijft de onderlinge relaties tussen de links en joints van de arm en is noodzakelijk om padplanning en obstakelvermijding uit te voeren. Bijkomend bepaalt enerzijds de keuze van de planner (of verzameling van planners) de manier waarop een traject van de eindeffector gevonden wordt doorheen de ruimte om een bepaald doel te bereiken. Anderzijds bepaalt de keuze van de *kinematic solver* de manier waarop met behulp van inverse kinematica de individuele joints doorheen de kinematische ketting van de arm poses aannemen om de eindeffectorpose te volgen doorheen het traject van de padplanning. Deze masterproef maakt gebruik van de OMPL-padplanners voor de manipulator en van de standaard KDL-kinematische solver van de *kdl_kinematics_plugin*. Deze instellingen zijn vereist om begin- en startpose voor een bepaalde padplanning van de manipulator op te geven in de MoveIt! plugin van RViz of door gebruik te maken van actions geschreven in aparte code. Onderstaande Figuur 3.20 toont een visualisatie in RViz van een padplanningsuitvoering met MoveIt! van de UR5 manipulator van AMBER met oranje startpose en groene doelpose. Dit gebeurt met de bidirectionele, RRT-Connect planner uit de OMPL-bibliotheek [69].



Figuur 3.20: Visualisatie in RViz van een trajectgeneratie van de UR5 manipulator van AMBER met de RRT-Connect OMPL-planner gebruik makend van MoveIt! met oranje startpose en groene doelpose.

MoveIt! vormt de sensordata van de Kinect 3D-camera op het PointCloud topic om naar een *octomap.* Dit verandert de puntenwolk naar *voxels* of driemdemensionale cellen als obstakels in de planningsomgeving. MoveIt! houdt bijgevolg rekening met de obstakels tijdens het plannen zodanig dat de manipulator er niet mee in botsing komt doorheen een gepland traject. Onderstaande Figuur 3.21 toont AMBER in RViz voor een muur voorgesteld door de octomap van MoveIt!.



Figuur 3.21: Visualisatie van AMBER in RViz met muren als octomap van MoveIt! om 3D-obstakels te vermijden tijdens de planning van de manipulator met MoveIt!.

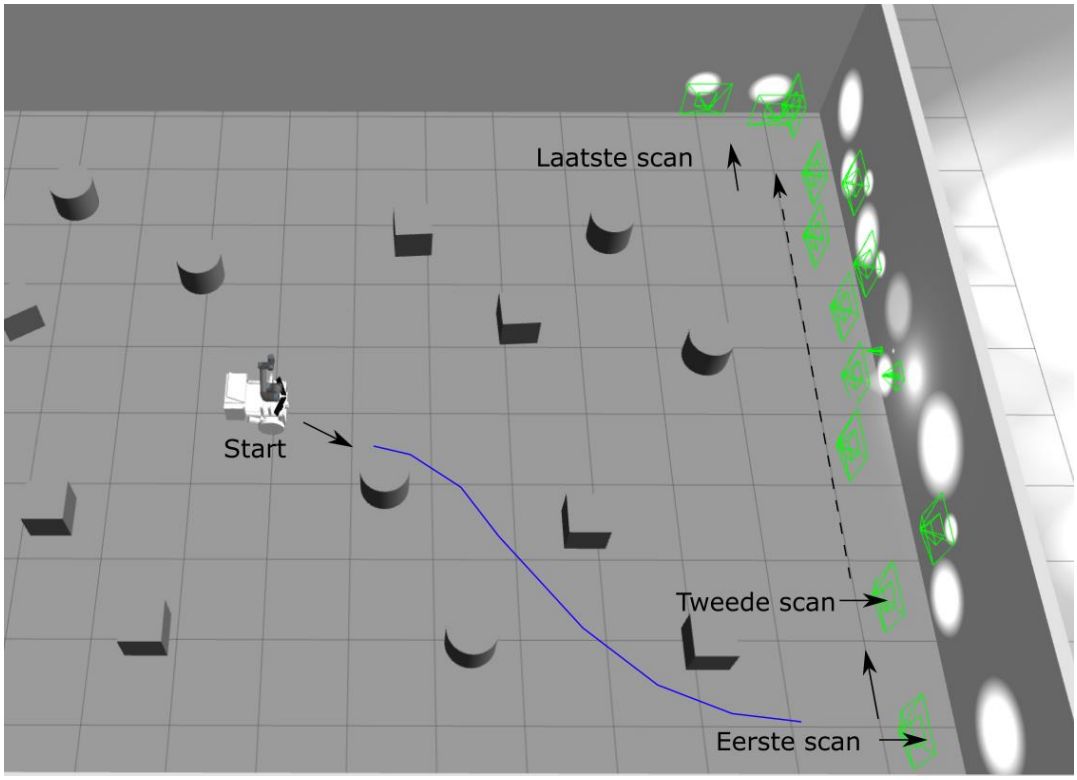
3.5 Conclusie

Dit hoofdstuk besprak de ondernomen stappen om het robotmodel van AMBER te simuleren gebruik makend van Gazebo en RViz in het ROS-raamwerk. Hierbij zorgt ROS voor de communicatie tussen verschillende processen en software met behulp van nodes, topics en messages. Verder waren aanpassingen aan het URDF-bestand vereist voor de simulatie in Gazebo en voor het gebruik van sensoren. Verschillende controllers zorgen voor de modelering van actuatoren om zodanig het robotmodel in Gazebo aan te sturen. Tot slot zorgt enerzijds de navigation stack met een kaart, AMCL en de move_base node met kostenkaarten en padplanningsalgoritmes voor de configuratie en uitvoering van padplanning van de mobiele basis. Deze masterproef maakt gebruik van een A* globale planner en de DWA als lokale planner voor obstakelvermijding. Anderzijds staat MoveIt! in voor de padplanning van de manipulator doorheen de driedimensionale ruimte. Dit gebeurt met de RRT-Connect uit de OMPL-bibliotheek. Hoewel specifieke aanpassingen vereist zijn afhankelijk van de robot, is het dankzij de open-source aard van ROS en met enige basiskennis van het ROS-raamwerk mogelijk om een volledige mobiele manipulator te modelleren en simuleren in het ROS-raamwerk.

4 Use case: aftasten van een muuroppervlak in simulatie

Dit hoofdstuk beschrijft een use case van de mobiele manipulator AMBER om de geïmplementeerde padplanningsmethode te testen in simulatie. Hierbij moet de robot, in kader van het ARCHER-project, muren aftasten in een simulatieomgeving om geïmiteerde radiologische bronnen in kaart te brengen. Zoals reeds eerder beschreven in hoofdstuk 3 gebeurt dit door een ontkoppelde padplanningsaanpak met enerzijds een A*- en DWA-padplanning van de mobiele basis en anderzijds een RRT-Connect planning van de manipulator.

De taak bestaat uit het autonoom navigeren van AMBER naar een muur vanuit het midden van een kamer met obstakels zoals te zien op Figuur 4.1. Hierbij moet de mobiele basis obstakels vermijden en blijft de manipulator in een rusttoestand aangezien de arm geen obstakels tegenkomt tijdens het traject van de basis. Hierna plant de manipulator, met stilstaande basis, een reeks van scanpunten op de muur om geïmiteerde radiologische bronnen in kaart te brengen. Tot slot navigeert AMBER naar het volgende gedeelte van de muur om de volgende scan uit te voeren. Op deze manier kan een volledig gedeelte van een muur of een volledige kamer onderzocht worden op de locaties van radiologische bronnen.

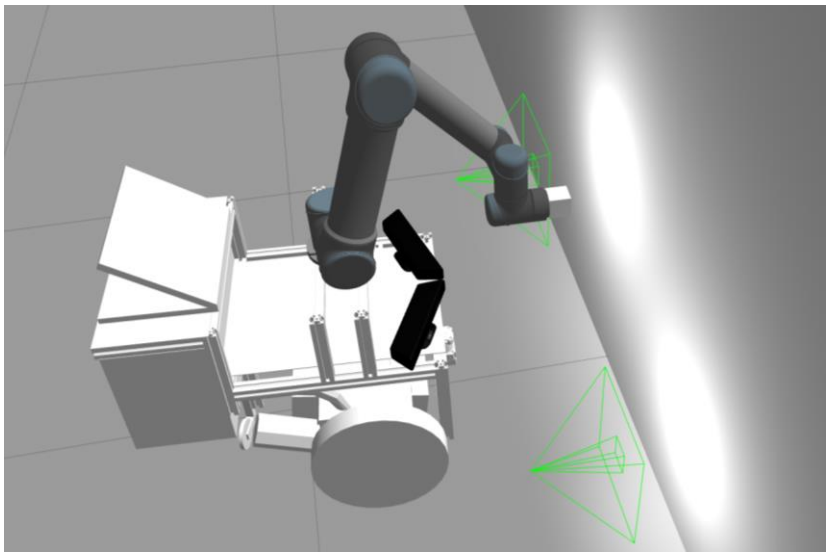


Figuur 4.1: Voorstelling van de muurscanprocedure in Gazebo met verschillende bronnen op een muur.

De nadruk van deze masterproef ligt echter op de combinatie van padplanners voor mobiele manipulatoren en niet op de simulatie van radiologische bronnen en sensoren. Aangezien deze radiologische simulatie in Gazebo binnen het kader van de masterproef te uitgebreid is, worden radiologische puntbronnen voorgesteld door lichtbronnen en meet een camera de verlichtingssterkte ter vervanging van een stralingsmeting.

4.1 Simulatie en detectie van licht als geïmiteerde radiologische bronnen

In Gazebo is het mogelijk om eenvoudig licht in de vorm van puntbronnen toe te voegen aan de simulatieomgeving. Door dergelijke lichtbron te richten naar een muur, toont het een gelijkaardig intensiteitsverloop. Zoals te zien op Figuur 4.2 imiteren verschillende lichtbronnen op deze manier de in kaart te brengen radioactieve concentraties in de muren van de simulatieomgeving.



Figuur 4.2: AMBER in Gazebo met twee naar de muur gerichte lichtbronnen toegevoegd aan de simulatieomgeving.

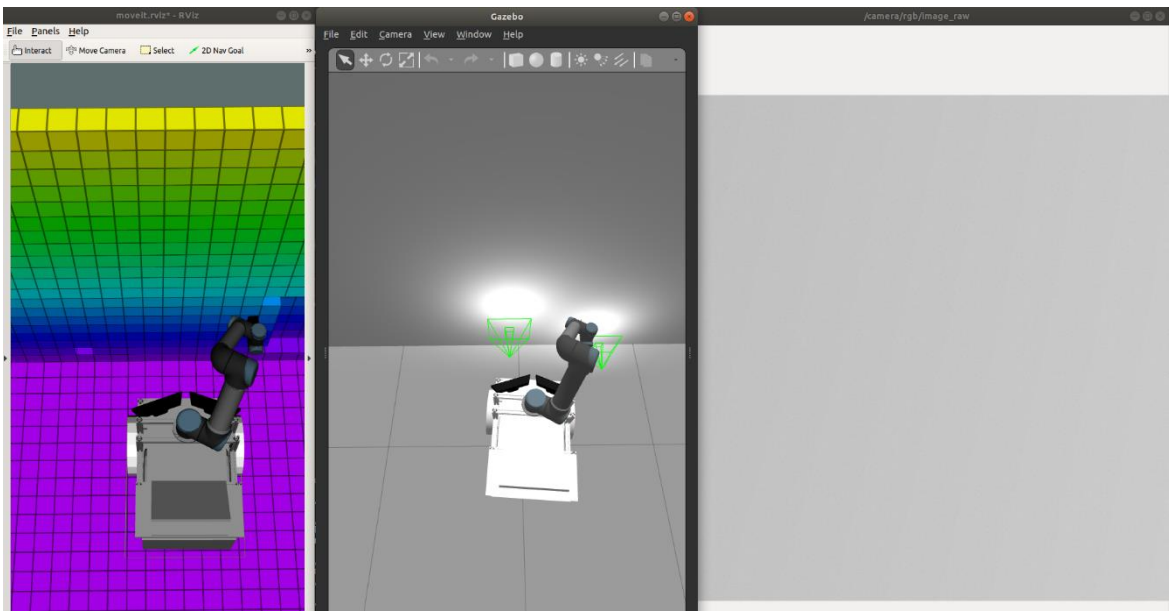
Een camera op de eindeffector kan vervolgens de lichtsterkte op de muur waarnemen om de lichtconcentratie in kaart te brengen. Dit gebeurt met een aangepaste cameraplugin voor Gazebo [70]. Aangezien dergelijke cameraplugins enkel licht op objecten waarnemen, zijn de lichtbronnen gericht op de muur en niet naar de robot toe. De *gazebo_light_sensor_plugin* is gekoppeld aan de *camera_link* op de eindeffector. Deze link is een blokje bevestigd aan de eindeffector dat de camera voorstelt, te zien op de vorige Figuur 4.2 en op Figuur 4.3. De plugin publisht het RGB-camerabeeld vanuit die linkpose op het topic *rgb/image_raw.* Hierna start het een node dat de gemiddelde verlichtingssterkte berekent van alle pixels van het RGB-beeld, gevisualiseerd met het *image_view* package [71]. Tot slot publisht de plugin deze waarde op het topic */light_sensor_plugin/lightSensor*. Door deze gemiddelde verlichtingssterkte op verschillende punten te meten, worden de locaties en groottes van de lichtbronnen in kaart gebracht.

src > amber_description > urdf > acro_mobile_manipulator.urdf.xacro

```
796     <!-- Camera -->
797     <gazebo reference="camera_link">
798       <sensor type="camera" name="camera1">
799         <update_rate>30.0</update_rate>
800         <camera name="head">
801           ...
802         </camera>
803         <plugin name="gazebo_light_sensor_plugin" filename="libgazebo_light_sensor_plugin.so">
804           ...
805           <imageTopicName>rgb/image_raw</imageTopicName>
806           <frameName>camera_link</frameName>
807           ...
808         </plugin>
809       </sensor>
810     </gazebo>
811
812     <link name="camera_link">
813     ...
814     </link>
815
816     <joint name="camera_joint" type="fixed">
817       <axis xyz="0 1 0" />
818       <origin xyz="0 0 -0.025" rpy="1.57 1.57 0"/>
819       <parent link="end_effector"/>
820       <child link="camera_link"/>
821     </joint>
```

Figuur 4.3: Deel van het URDF-bestand dat de aangepaste cameraplugin beschrijft en koppelt aan de camera_link dat bevestigd is aan de eindeffector van de robotarm.

Onderstaande Figuur 4.4 toont AMBER tijdens het scannen van een muur naar de lichtsterktes. Links is AMBER te zien in RViz met de MoveIt! plugin waarbij de door MoveIt! gegenereerde octomap de muur als obstakel weergeeft. De padplanning van de manipulator houdt rekening met de octomap als obstakel zodanig dat de manipulator tijdens een gepland traject niet botst met de octomap en dus ook niet met de muur in Gazebo. In het midden is AMBER te zien in Gazebo waarbij het de muur scant naar lichtbronnen. Het rechterdeel toont het RGB-beeld op het *rgb/image_raw* topic waarvan de node van de plugin in elk scanpunt de gemiddelde verlichtingssterkte berekent.



Figuur 4.4: Werkingsprincipe van de samenwerking tussen MoveIt! en Gazebo voor een muurscan. (links) AMBER in RViz en MoveIt! waarbij de octomap instaat voor de botsingsvermijding met de muur tijdens een padplanning. (midden)

AMBER in Gazebo tijdens een muurscan naar de lichtbronnen. (rechts) Het RGB-beeld op het rgb/image_raw topic waarvan de plugin in elk scanpunt de gemiddelde lichtsterkte berekent.

4.2 Structuur van het muurscanprogramma

Onderstaande Figuur 4.5 toont de pseudocode waarin meerdere processen samenwerken om een muurscanprocedure uit te voeren. Verschillende functies zijn gedefinieerd in aparte *Pythonscripts* die met elkaar data uitwisselen. In deze Pythonprogramma's gebeurt enerzijds de communicatie met de manipulator met de *Move Group Python Interface* van MoveIt! en anderzijds de communicatie met de move_base node voor de basis met ROS-acties. De code vereist als input een doelpositie voor de basis in de gekende kaart. Dit manueel ingegeven doel moet voldoende dicht bij de muur zijn om zodanig de muur te kunnen scannen. De code geeft een lijst terug van de achtereenvolgende scans met daarin de x-, y- en z-coördinaten en oriëntatie van de scanpunten in het assenstelsel van de kaart alsook de gemeten intensiteit in elk scanpunt.

Algorithm 1: Muurscanprocedure

Input: goal(x, y, θ)
Output: combined_scan_data

```
1  move_base_goal ← pose_from_goal(goal)
2  move_base_action_server.send(move_base_goal)
3  if move_base_action_server.result() = true then
4      scan_points_arm ← get_scan_points()
5      scan_data_map ← arm_scan(scan_points_arm)
6      combined_scan_data.extend(scan_data_map)
7      for Aantal bijkomende scans do
8          move_base_goal ← goal_from_scan_point(scan_points_arm[n])
9          move_base_action_server.send(move_base_goal)
10         if move_base_action_server.result() = true then
11             scan_points_arm ← get_scan_points()
12             scan_data_map ← arm_scan(scan_points_arm)
13             combined_scan_data.extend(scan_data_map)
14 return combined_scan_data
```

Figuur 4.5: Stappenplan van de muurscanprocedure met functies geïmplementeerd in Python en ROS-acties.

De goalinput bestaat uit een x- en y-waarde die de positie van de robot beschrijven in de kaart. Verder stelt θ, de hoek rond de z-as vertrekkende vanaf de x-as, de oriëntatie van de robot voor. De move_base node vereist een doel als input in de vorm een *target pose.* Dit is een combinatie van een x-, y-, z-positie en een x-, y-, z-, w-oriëntatie in de vorm van eenheidsquaternionen. Daarom zet de functie *pose_from_goal,* in lijn 1, de opgegeven goal om in een *move_base_goal* als target pose. Deze stelt, de juiste vorm voor de move_base, dezelfde doelpositie van de mobiele basis voor als de initiële goal. Vervolgens wordt de move_base_goal naar de actieserver van de move_base gestuurd waarna de mobiele basis navigeert richting het doel, zoals vermeld in hoofdstuk 3.

De actieserver geeft als resultaat *true* wanneer de mobiele basis het doel bereikt heeft. Indien dit geldt, zorgt lijn 4 tot en met 6 voor de accumulatie van data uit scanpunten. Allereerst geeft de

functie *get_scan_points()* een lijst $scan\_points_{arm}$ van scanpunten terug. Hoofdstuk 4.2.2 beschrijft uitgebreid op welke manier deze functie scanpunten berekent uit een 2D-laserscan van de muur. Hierna zorgt de functie *arm_scan* voor de opeenvolgende padplanning van de manipulator naar alle scanpunten. In elk scanpunt slaat de functie de gemiddelde lichtsterkte op van het topic */light_sensor_plugin/lightSensor*. Hierna geeft de functie een lijst $scan\_data_{map}$ terug met alle scanpunten gedefinieerd in het assenstelsel van de kaart met de bijhorende gemiddelde verlichtingssterkte in elk punt. Tot slot wordt deze lijst in lijn 6 toegevoegd aan een globale lijst *combined_scan_data* die alle punten van opeenvolgende scanopdrachten combineert.

Het scannen van een volledige muur gebeurt in verschillende scans. Voor het aantal bijkomende scans om een gewenst gedeelte van een muur te scannen, wordt na het tot nu toe doorlopen programma voor de eerste scan, de code van lijn 8 tot en met 13 herhaald. Hierbij berekent eerst de functie *goal_from_scan_point* de nieuwe pose voor de mobiele basis op basis van het laatste scanpunt van de vorige scan. Verder verloopt lijn 9 tot en met 13 analoog aan lijn 2 tot en met 6 voor de volgende scans. Tot slot resulteert de muurscanprocedure in een lijst van data over alle scanpunten van de opeenvolgende scan.

4.2.1 Bepaling van scanpunten uit laserscandata

Dit deel bespreekt hoe de functie get_scan_points scanpunten berekent uit een 2D-laserscan van een muur. Zoals reeds beschreven bij de algemene pseudocode, gebeurt dit elke keer dat AMBER een positie aanneemt voor een muur. Aangezien er kleine afwijkingen kunnen zitten op de oriëntatie van de mobiele basis bij het bereiken van de doelpositie, is het belangrijk dat de oriëntatie van de muur ten opzichte van de manipulator exact gekend is. Op deze manier volgt de eindeffector tijdens de scans de oriëntatie van de muur. Daarom moet de rechte gekend zijn die de muuroriëntatie beschrijft.

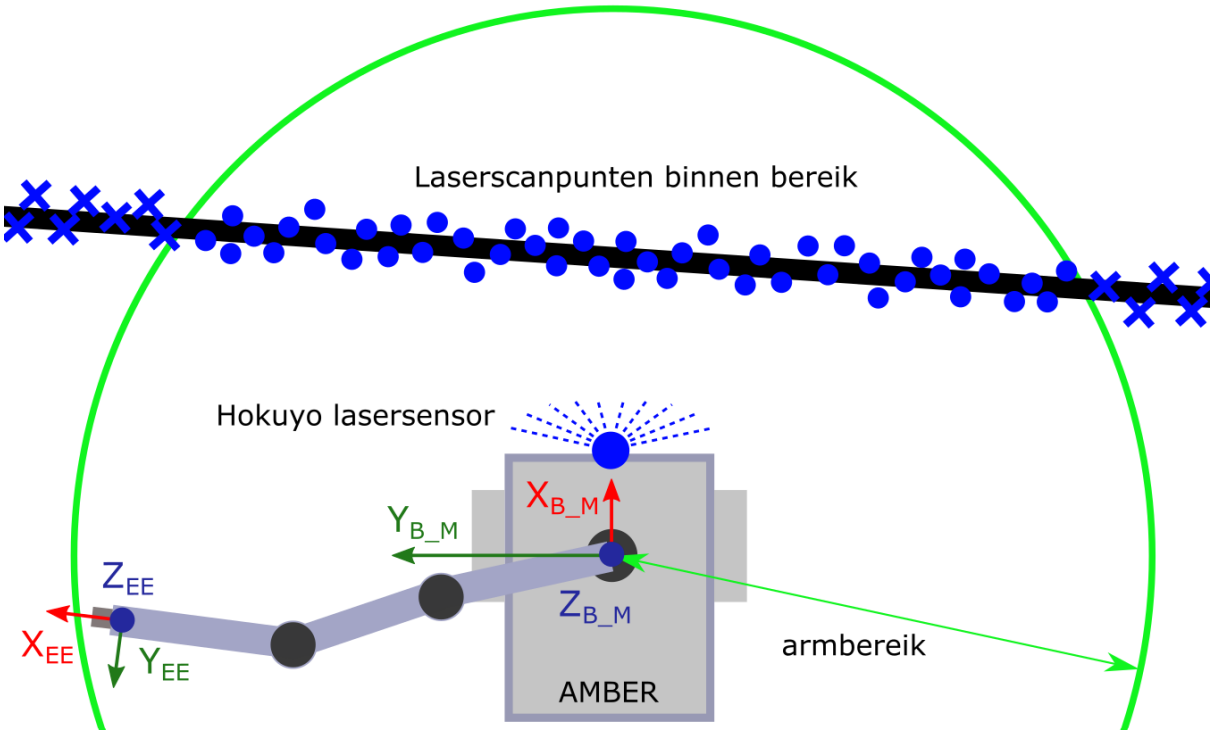
De Hokuyo 2D-laserscanner geeft een verzameling van punten weer waarop de laserstralen op een obstakel botsen. Deze punten zijn gedefinieerd door een hoek rond de sensor en door een afstand tot het obstakel. Allereerst converteert de functie get_scan_points de polaire coördinaten van de 2D-laserscanpunten naar Cartesiaanse coördinaten aan de hand van onderstaande formule.

$$\begin{cases} x_n = l_n * \sin(\delta_n) \\ y_n = -l_n * \cos(\delta_n) \end{cases} \tag{4.1}$$

Waarbij:

- l de afstand gemeten door laserscanner is;
- δ de hoek waaronder deze afstand gemeten is.

Uit deze omzetting volgt een lijst met x-coördinaten en een lijst met y-coördinaten van de 2D-laserscanpunten. Deze lijsten beschrijven een laserscanpunt door een x- en y-coördinaat met dezelfde index in de lijst. Vervolgens transformeert de functie de punten naar het assenstelsel van de manipulatorbasis. Dit gebeurt enkel voor de punten die binnen het armbereik liggen. Verdere berekeningen houden bijgevolg enkel rekening met de punten waarvan de afstand tot de manipulatorbasis kleiner is dan het armbereik. Figuur 4.6 toont deze selectie van laserscandata binnen het bereik van de manipulator.

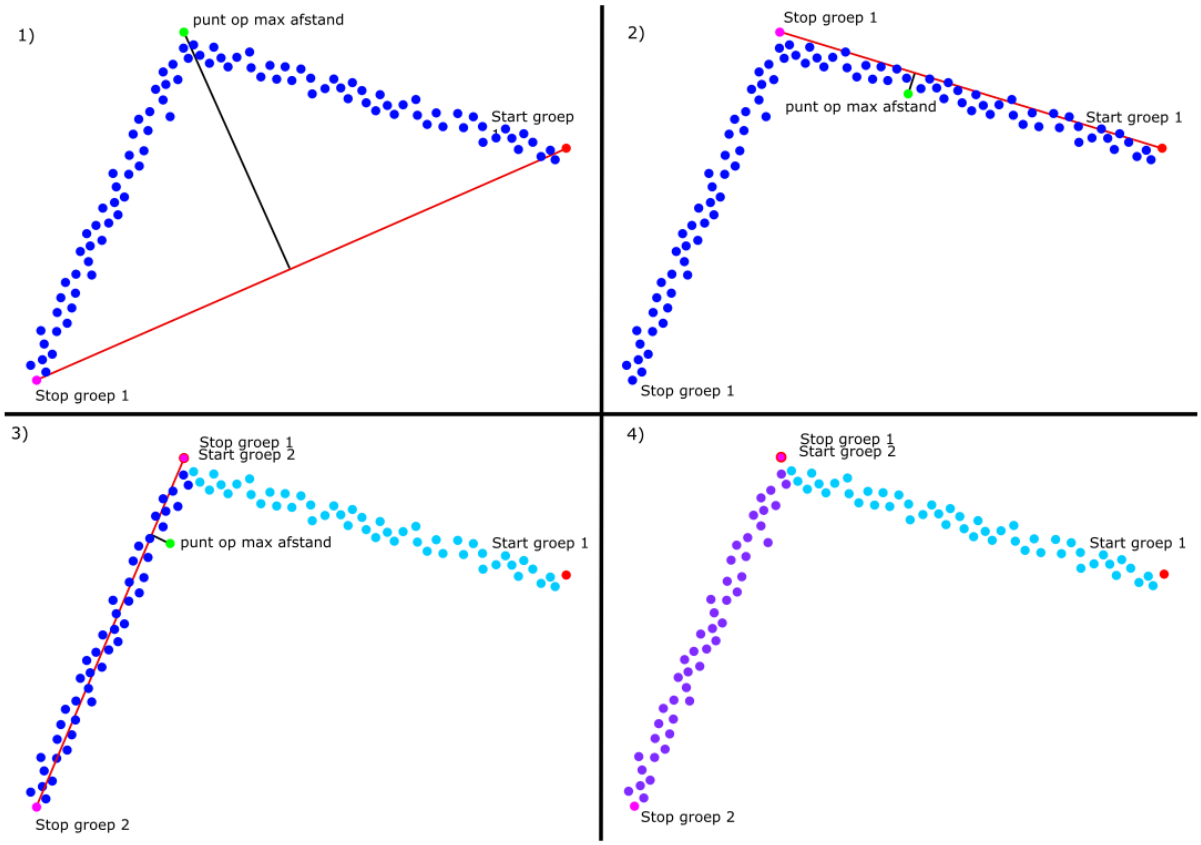


Figuur 4.6: Schematische voorstelling van AMBER voor een muur waarbij de filter enkel de 2D-laserscanpunten binnen armbereik behoudt voor verdere berekeningen. De blauwe bolletjes op de zwarte muur tonen deze punten. Hierin stelt B_M (base_manipulator) het assenstelsel van de manipulatorbasis voor en EE dat van de eindeffector.

Vervolgens deelt een *split-and-merge-algoritme* deze punten verder op in groepen om, indien aanwezig, verschillende muuronderdelen te herkennen. Elke groep wordt gekenmerkt door een start- en stoppunt. Een punt behoort tot een bepaalde groep wanneer het binnen de start- en stoppunten van die groep ligt. Bij aanvang van het split-and-merge-algoritme is er slechts één groep. Het eerste en laatste laserscanpunt zijn respectievelijk het start- en stoppunt van deze eerste groep. Hierna bepaalt het algoritme de vergelijking van de lijn die het start- en stoppunt verbindt. Vervolgens berekent onderstaande formule van elk punt, behorende tot die groep, de loodrechte afstand tot de lijn.

$$d_n = \frac{\left|(x_{stop} - x_{start}) * y_n + (y_{start} - y_{stop}) * x_n - (x_{stop}y_{start} - x_{start}y_{stop})\right|}{\sqrt{(x_{stop} - x_{start})^2 + (y_{start} - y_{stop})^2}} \quad (4.2)$$

Indien de maximaal berekende afstand van alle punten onder een bepaalde drempelwaarde ligt, stopt het algoritme. Wanneer de maximale afstand echter boven de drempelwaarde ligt, is het punt behorende tot deze maximale afstand het nieuw stoppunt van de huidige groep. Het algoritme blijft deze stappen herhalen totdat de maximale afstand onder de drempelwaarde ligt. Indien hieraan voldaan is, deelt het algoritme de overige laserscanpunten op in groepen. Hierbij is het startpunt van deze iteratie het stoppunt van de vorige en het laatste laserscanpunt opnieuw het stoppunt. Het algoritme herhaalt deze stappen totdat alle punten in groepen opgedeeld zijn. Figuur 4.7 toont een schematische weergave van de werking van dit algoritme [5].



Figuur 4.7: Verloop van het split-and-merge-algoritme van laserscanpunten voor een hoek in de muur. (1) Start- en stoppunt zijn respectievelijk eerste en laatste laserscanpunt. De maximale afstand van een punt tot de rechte die start- en eindpunt verbindt ligt boven de drempelwaarde waardoor dit punt het nieuwe stoppunt wordt van de groep. (2) Ditmaal ligt de maximale afstand onder de drempelwaarde. De punten tussen start en stop behoren bijgevolg tot dezelfde groep. (3) De overige punten worden verder gegroepeerd door het stoppunt van groep 1 gelijk te stellen aan het startpunt van groep 2 en het laatste laserscanpunt gelijk te stellen aan het stoppunt van groep 2. Opnieuw liggen alle afstanden tot de rechte onder de drempelwaarde. (4) Het split-and-merge-algoritme heeft alle punten ingedeeld in groepen. Deze groepen zijn gekenmerkt door hun start- en stoppunt.

Vervolgens bepaalt de functie *get_scan_points* de best passende rechte doorheen elke puntengroep op basis van de kleinste-kwadratenmethode. Deze rechte is van volgende vorm:

$$a' * y + b' * x - c' = 0 \tag{4.3}$$

waarbij:

- $a' = -n * \cos(\theta)$
- $b' = n * \sin(\theta)$
- $c' = n * c$

Hierbij is c de loodrechte afstand tussen de oorsprong van het assenstelsel en de rechte. θ vormt de hoek tussen de negatieve y-as en deze loodrechte afstand, c en θ zijn zichtbaar op Figuur 4.9. De parameters a', b' en c' die uit de kleinste-kwadratenmethode volgen, zijn echter een veelvoud van de gewenste waarden. Aangezien de effectieve afstand tot de muur gekend moet zijn, moeten a', b' en c' zo aangepast worden zodat c' de effectieve afstand tot de muur voorstelt. Dit kan door c' eerst door zichzelf te delen en nadien te vermenigvuldigen met de afstand tussen de oorsprong en de

rechte. De vermenigvuldiging met deze factor moet voor alle parameters gebeuren zodat deze nog steeds dezelfde rechte beschrijven. Dit resulteert in:

$$c = \frac{c'}{c'} * \frac{|a' * y_0 + b' * x_0 - c'|}{\sqrt{a'^2 + b'^2}} = \frac{|a' * 0 + b' * 0 - c'|}{\sqrt{a'^2 + b'^2}} = c' * \frac{1}{\sqrt{a'^2 + b'^2}} \tag{4.4}$$

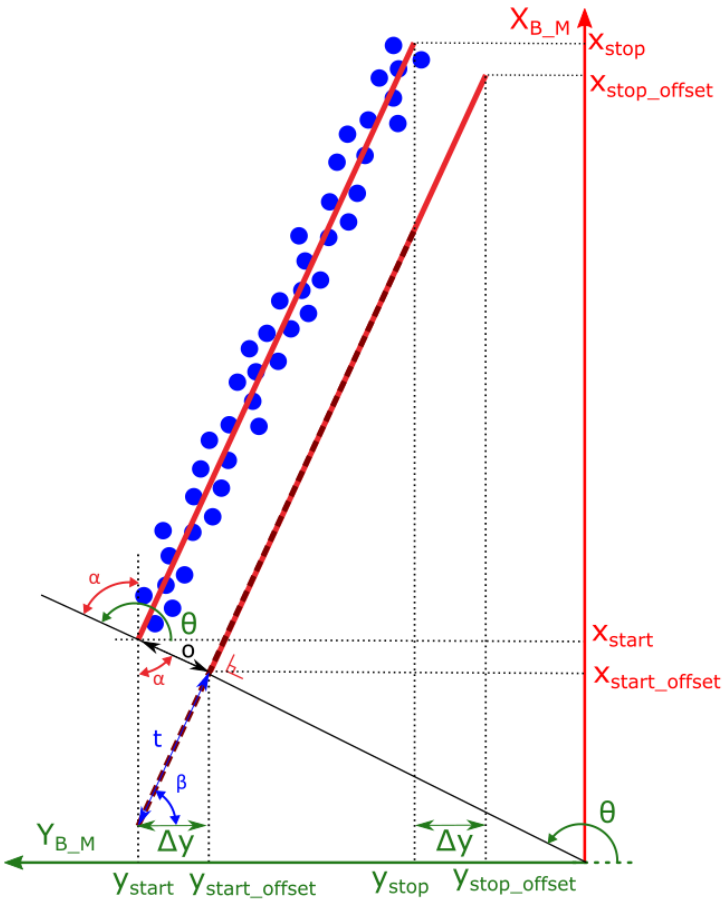
$$b = \frac{b'}{c'} * \frac{|a' * y_0 + b' * x_0 - c'|}{\sqrt{a'^2 + b'^2}} = b' * \frac{1}{\sqrt{a'^2 + b'^2}} = \sin(\theta) \tag{4.5}$$

$$a = \frac{a'}{c'} * \frac{|a' * y_0 + b' * x_0 - c'|}{\sqrt{a'^2 + b'^2}} = a' * \frac{1}{\sqrt{a'^2 + b'^2}} = -\cos(\theta) \tag{4.6}$$

Terug invullen in de originele vergelijking (4.3) geeft:

$$a * y + b * x - c = 0 \tag{4.7}$$

De functie heeft tot nu toe uit 2D-laserscandata de rechten bepaald die de oriëntatie van de muren beschrijven in het assenstelsel van de manipulatorbasis. De meting van bronnen in de muur neemt plaats op een bepaalde afstand van de muur. Dit gebeurt door de rechte die de muur beschrijft, met een bepaalde afstand loodrecht op de rechte te verschuiven richting de robot. Dit kan door c in bovenstaande vergelijking met een bepaalde offset waarde te verminderen. Hieruit volgt een lijnstuk die dichter bij de robot ligt en evenwijdig is met de muurrechte. Om dit lijnstuk loodrecht te verschuiven moeten nieuwe y_start- en y_stoppunten, respectievelijk y_{start_offset} en y_{stop_offset}, berekend worden zoals te zien in onderstaande Figuur 4.8.



Figuur 4.8: Berekening van y-start en -stop voor de verschoven lijn op basis van de hoek van de muur θ en de afstand o waarover deze verschoven is.

Onderstaande formules berekenen y_{start_offset} en y_{stop_offset}.

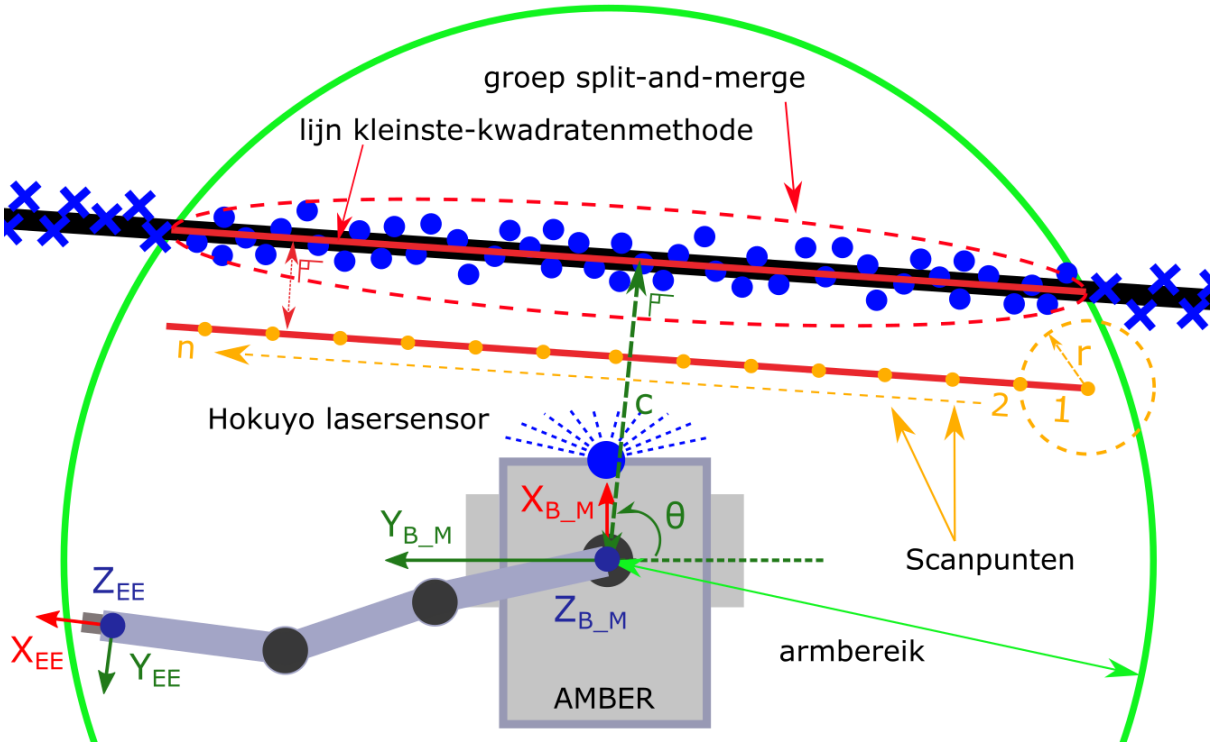
$$\begin{aligned} y_{start\_offset} &= y_{start} - \Delta y && (4.8)\\ &= y_{start} - \cos(\beta) * t \\ &= y_{start} - \cos(\beta) * \cos(\alpha) * o \\ &= y_{start} - \cos(\theta - {}^{\pi}/_{2}) * \cos(\theta - {}^{\pi}/_{2}) * o \\ &= y_{start} - \sin(\theta)^2 \\ &= y_{start} - \mathrm{b}^2 * o && (4.9)\\ y_{stop\_offset} &= y_{stop} - \Delta y && (4.10)\\ &= y_{stop} - \mathrm{b}^2 * o && (4.11) \end{aligned}$$

De x_{start_offset} en x_{stop_offset} volgen uit het invullen van respectievelijk y_{start_offset} en y_{stop_offset} in de vergelijking van de verschoven rechte (4.7).

Vervolgens genereert de functie op dit nieuw lijnstuk een aantal equidistante scanpunten door, vertrekkende van het startpunt op het lijnstuk, constructiecirkels met straal r te vormen die het lijnstuk snijden. Het snijpunt van deze cirkel met het lijnstuk resulteert in een nieuw scanpunt van waaruit een nieuwe constructiecirkel met dezelfde straal opnieuw het lijnstuk snijdt op de positie van het volgend scanpunt. Elk punt wordt toegevoegd aan de lijst $scan\_points_{arm}$. Deze groei van punten op het lijnstuk herhaalt zich totdat het laatste punt op het lijnstuk geen cirkel heeft die een

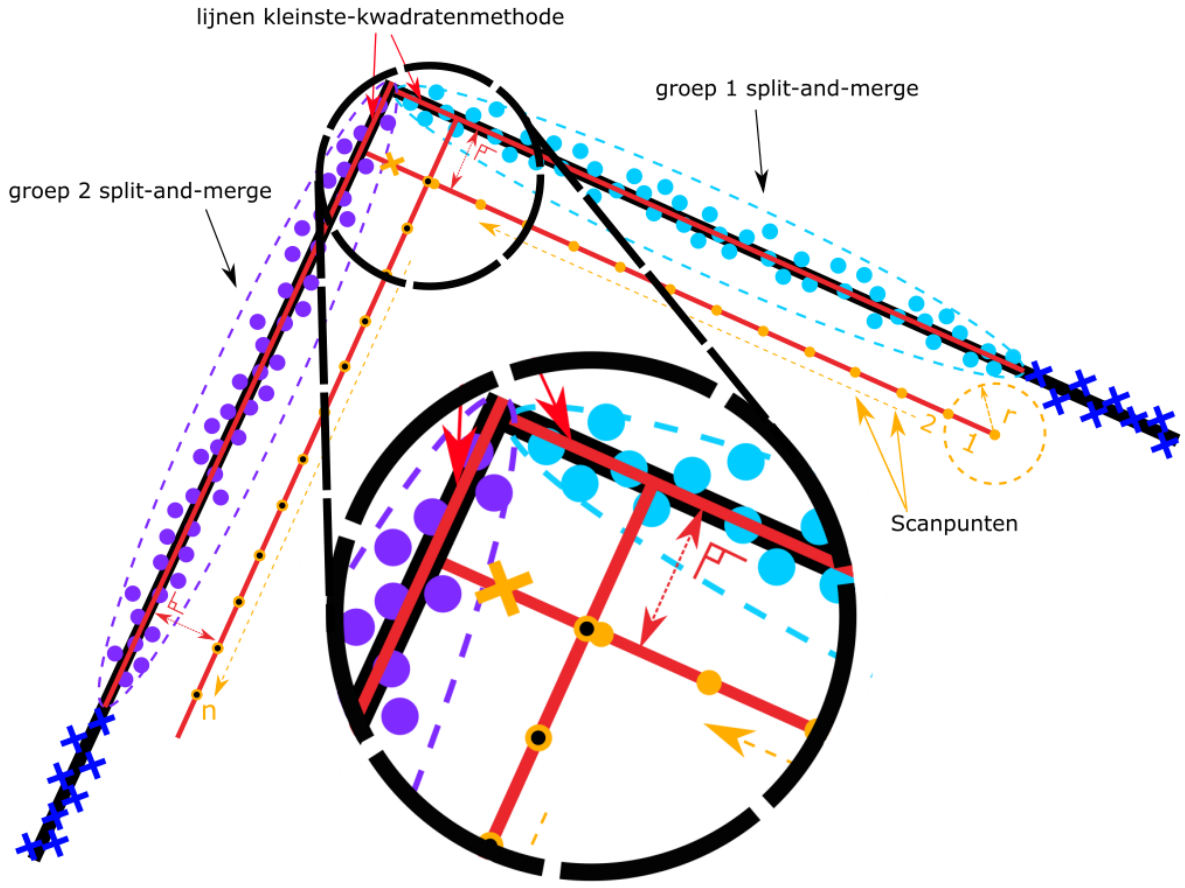
snijpunt vormt met het lijnstuk. De functie *get_scan_points* resulteert nu in een lijst $scan\_points_{arm}$ met de te doorlopen scanpunten voor de eindeffector gedefinieerd in het assenstelsel van de manipulatorbasis.

Onderstaande Figuur 4.9 toont schematisch het verloop van de functie *get_scan_points* die uitgaand van 2D-laserscanpunten binnen armbereik opeenvolgend puntgroepen vormt, een best passende rechte genereert voor elke groep, lijnstukken evenwijdig hiermee plaatst en scanpunten genereert op elk lijnstuk. In dit voorbeeld van een recht stuk muur is er maar één groep, één best passende rechte, één lijnstuk en bestaat de lijst $scan\_points_{arm}$ uit scanpunten van hetzelfde lijnstuk.



Figuur 4.9: Schematische voorstelling van AMBER voor een rechte muur waarbij de functie get_scan_points equidistante scanpunten genereert op een lijnstuk evenwijdig aan de best passende rechte doorheen een groep 2D-laserscanpunten binnen het armbereik.

Bij het scannen van een hoek in de muur, vormt het split-and-merge-algoritme twee groepen uit de 2D-laserscandata. Vervolgens ontstaan twee best passende rechten en twee lijnstukken, waarbij elk lijnstuk evenwijdig is met de overeenkomstige rechte op een muurzijde. Dit resulteert in twee lijnstukken die elkaar snijden ter hoogte van de hoek in de muur. Dit snijpunt is vervolgens gekend in x- en y-coördinaat. De functie *get_scan_points* voegt wederom vanuit de rechterkant van het eerste lijnstuk scanpunten toe aan de lijst $scan\_points_{arm}$. Bovendien wordt ook de hoek θ van de muur, waarbij het scanpunt tot behoort, aan de lijst toegevoegd. Wanneer de punten van het eerste lijnstuk het snijpunt naderen, is het laatste scanpunt op dat lijnstuk het punt waarvan de y-waarde nog net kleiner is dan de y-waarde van het snijpunt. Vervolgens is het snijpunt tussen de lijnstukken het eerste scanpunt op het tweede lijnstuk. Hierna verloopt de generatie van scanpunten analoog op het tweede lijnstuk. Onderstaande Figuur 4.10 geeft dit schematisch weer waarbij het "X" scanpunt na het snijpunt niet meer behoort tot de lijst $scan\_points_{arm}$. De lijst bevat nu alle punten van beide lijnstukken.



Figuur 4.10: Schematische voorstelling van de functie get_scan_points voor een 2D-laserscan van een hoek in de muur. Hierbij deelt de functie de punten in armbereik op in twee groepen waarna het de best passende rechte bepaalt en twee lijnstukken evenwijdig plaatst met de rechten. Vervolgens genereert het vertrekkende van het rechterlijnstuk de equidistante scanpunten tot aan het snijpunt met het tweede lijnstuk. Tot slot begint deze generatie op het tweede lijnstuk vanaf het snijpunt tussen de lijnstukken.

De methode om vanuit 2D-laserscandata scanpunten te genereren in het assenstelsel van de kaart is niet de enige manier om een muur te scannen. Het zou ook mogelijk zijn om met behulp van de *Octomap API* de eindeffector te plannen naar bepaalde cellen van de Octomap in MoveIt! [72]. Deze methode, die Pythoncode gebruikt om de muuroriëntatie te bepalen op basis van sensordata uit Gazebo en de Octomap in MoveIt! enkel gebruikt om de muur te ontwijken tijdens padplanning van de arm, bleek in het kader van deze masterproef echter meer toepasbaar. Dit aangezien het te tijdrovend zou zijn om in latere stadia van de masterproef de C++ *Octomap API* onder de knie te krijgen. Een nadeel van de huidige aanpak is dat het enkel toepasbaar is voor rechte muren aangezien er een rechte gefit wordt op de sensordata. Hierdoor kan deze methode niet toegepast worden in een buis of in ruimtes met bochten of krommingen in muren.

4.2.2 Padplanning tijdens muurscanprocedure

Zoals eerder beschreven gebeurt de padplanning tijdens het in kaart brengen van bronnen met een ontkoppelde padplanning tussen de mobiele basis en de manipulator van AMBER. Hierbij navigeert de basis naar een positie voor een muur, doorloopt de manipulator de planning tussen scanpunten en navigeert de basis naar een volgende positie voor de muur om een nieuwe scan uit te voeren.

Deze planning gebeurt met een A* globale planner en een DWA lokale planner als obstakelvermijding.

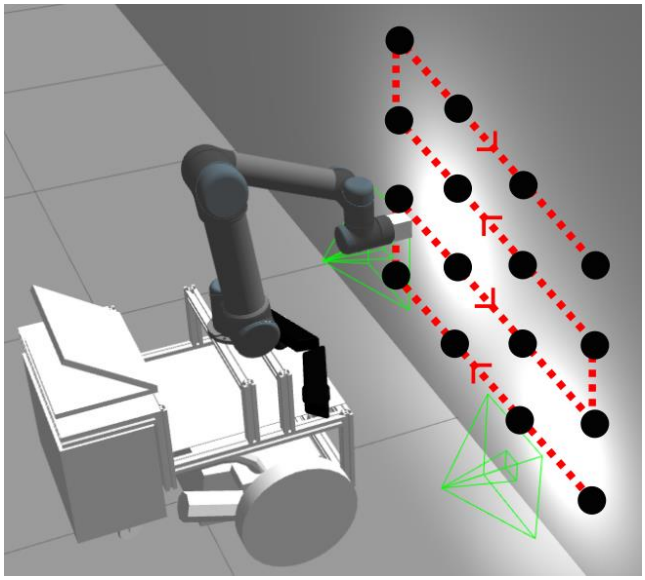
Met behulp van de *rospy* bibliotheek is het mogelijk om vanuit Pythoncode verschillende ROS-processen uit te voeren [73]. Zo is het mogelijk om een actionlib client-server koppeling te starten dat communiceert met de ROS-actie gebaseerde move_base om de mobiele basis aan te sturen. Hiermee ontvangt de move_base node via de service doelen voor de basis afkomstig van de client en kan de client het resultaat van de uitvoering ervan opvragen van de server. De move_base node krijgt via deze ROS-acties een doelpositie uit de Pythoncode, plant een pad ernaartoe en stuurt snelheidscommando's om het doel te bereiken. Bij aankomst start de rest van de muurscanprocedure.

Nadat de lijst $scan\_points_{arm}$ volledig is, maakt de functie *arm_scan* hiervan gebruik om de manipulator doorheen een zigzagpatroon naar de opeenvolgende punten in het assenstelsel van de manipulatorbasis te plannen. De communicatie vanuit de Pythoncode met de manipulator in MoveIt! gebeurt met behulp van de *Move Group Python Interface* [74]. Deze bibliotheek laat toe om gewenste poses van een bepaalde move group in te stellen als doel, een traject te plannen naar dit doel en het traject uit te voeren. Deze planning maakt gebruik van de OMPL RRT-Connect planner en van de octomap van de muur in MoveIt! om botsingsvrije paden te plannen.

Na de functie *get_scan_points* plant de functie *arm_scan* allereerst naar een bekende armconfiguratie en voert het de planning uit. Deze configuratie van de move group is op voorhand gedefinieerd in het SRDF-bestand om de scans te beginnen met een gewenste oriëntatie van de eindeffector. Zoals te zien op Figuur 4.11 botst het uiteinde van de eindeffector op deze manier nooit met een gedeelte van een manipulatorarm tijdens een traject tussen twee opeenvolgende scanpunten. Dit zou echter voor een overbodig lang omwegtraject van het manipulatorpad zorgen tussen twee punten.

De functie *arm_scan* doorloopt hierna een lijst met manueel ingegeven hoogtepunten in het assenstelsel van de manipulatorbasis. Bij een even rij, bijvoorbeeld op 0.2 m hoogte in het assenstelsel van de manipulatorbasis, doorloopt de functie beurtelings de punten uit de lijst $scan\_points_{arm}$ om telkens naar een volgend punt te plannen en het traject uit te voeren. Op deze manier volgt de eindeffector het muurvlak in de ruimte. In elk punt past de functie de oriëntatie van de eindeffector aan om loodrecht op de muur te meten. Dit gebeurt door de eindeffector te oriënteren volgens de juiste omzetting van de hoek θ, horend bij elk punt uit de lijst $scan\_points_{arm}$, die de hoek van de muur beschrijft. Vervolgens slaat de functie in elk punt de gemiddelde verlichtingssterkte op. Hierna transformeert de functie in elk punt de pose van het punt in het assenstelsel van de manipulatorbasis naar de pose van het punt in het wereldassenstelsel, namelijk die van de kaart. Dit gebeurt met behulp van functies uit het *tf2*-package [75]. Op oneven rijen doorloopt de functie de lijst $scan\_points_{arm}$ achterwaarts om van een volledige scan een zigzagpatroon te vormen op een mobiele basispositie. Daarbuiten gebeuren de bewerkingen op elk punt analoog aan de bewerkingen op een even rij. Deze twee uitvoeringen in elk punt resulteren in een lijst $scan\_data_{map}$ die de x-, y-, z-positie en oriëntatie in het wereldassenstelsel en de verlichtingssterkte van alle scanpunten van het zigzagpatroon bevat.

Onderstaande Figuur 4.11 toont AMBER in Gazebo voor een muur met lichtbronnen tijdens het doorlopen van het zigzagpatroon met bijhorende scanpunten.



Figuur 4.11: AMBER in Gazebo voor een muur met lichtbronnen tijdens het doorlopen van het zigzagpatroon met bijhorende scanpunten.

Aangezien het zigzagpatroon van de scan op elke hoogte doorlopen wordt op basis van dezelfde rechte afkomstig van 2D-laserscandata vlak boven de grond, is deze methode van scannen enkel toepasbaar op muren die in de hoogte een constante verticale vorm behouden.

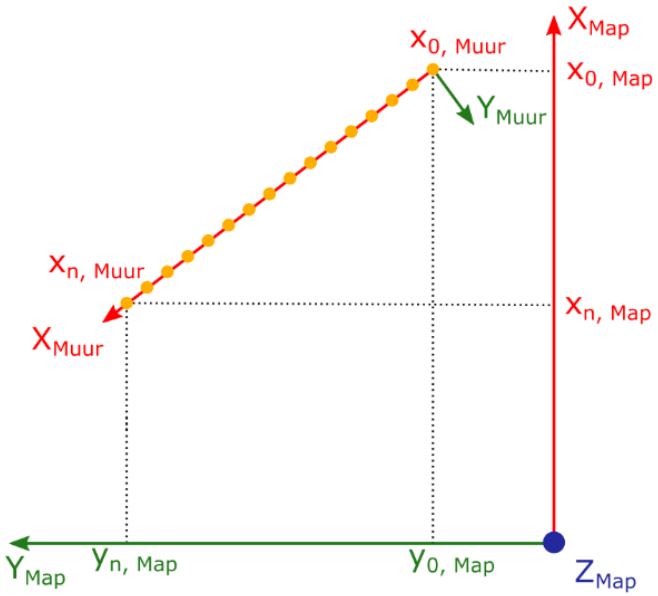
Bij een volgende scan van een muurgedeelte berekent de functie *goal_from_scan_point* de nieuwe doelpositie voor de move_base node vanuit de positie van het laatste scanpunt uit de vorige scan. Dit gebeurt door allereerst een hulprechte, evenwijdig aan de muur, te plaatsen op een vaste afstand van de muur richting de robot. Vervolgens maakt de functie een constructiecirkel met het armbereik als straal vanuit het laatste scanpunt van de vorige scan. Het punt waar de hulprechte en constructiecirkel elkaar snijden, is de nieuwe doelpositie voor de basis. De basis navigeert vervolgens hiernaartoe om de volgende scan te beginnen. Hoewel de nieuwe positie berekend wordt op basis van het armbereik met als oorsprong het assenstelsel van de manipulatorbasis en de oorsprong van de basis dus navigeert naar de positie van de manipulatorbasis, blijft deze nieuwe positie correct aangezien de assenstelsels van de manipulator en basis vlak boven elkaar liggen.

4.2.3 Visualisatie gescande radioactieve bronnen

Een muurscanprocedure resulteert in een lijst *combined_scan_data* die de informatie over de coördinaten, verlichtingssterkte en pose van de eindeffector bevat van alle scanpunten uit de opeenvolgende scans. Dergelijke procedure kan een afzonderlijk stuk van een muur scannen, een stuk van een muur scannen vertrekkend van een hoek en stoppend aan de volgende hoek of meerdere muurzijden scannen waarbij het verschillende hoeken passeert. Om een volledig afzonderlijke muur met bronnen tweedimensionaal in kaart te brengen, wordt de lijst *combined_scan_data* allereerst manueel opgesplitst in lijsten van afzonderlijke muurzijden. Dit gebeurt op basis van de oriëntatie van de eindeffectorpose in het wereldassenstelsel van elk

scanpunt. Deze verschilt echter afhankelijk van de gescande muur aangezien de eindeffector loodrecht op een muur scant.

Een lijst van scanpunten definieert nu alle scanpunten van een afzonderlijke muurzijde als x-, y- en z-coördinaten in het wereldassenstelsel met bijhorende verlichtingssterkte. Om het verloop van de verlichtingssterkte te tonen op een tweedimensionale muur, heeft de definitie van deze punten een te hoge dimensie. Het y-coördinaat, dat de diepte voorstelt, is hierbij overbodig. Daarom zorgt een afzonderlijk Pythonprogramma voor de projectie van alle scanpunten op een vlak met de muuroriëntatie. Zo zijn de punten gedefinieerd als een x- en z-coördinaat in een lokaal muurassenstelsel met in elk punt van het vlak een waarde voor de verlichtingssterkte. Hierbij zijn de z-coördinaten in het lokaal assenstelsel hetzelfde als in het wereldassenstelsel. Onderstaande Figuur 4.12 toont deze voorstelling van de scanpunten in het lokaal assenstelsel.



Figuur 4.12: Transformatie van de scanpuntcoördinaten van het wereldassenstelsel naar het lokaal assenstelsel met de muuroriëntatie. Hierbij verliest de positiedefinitie een dimensie door de overbodigheid van de y-coördinaten.

Voor elk punt gebeurt de omzetting van de x- en y-coördinaten in het wereldassenstelsel naar x-coördinaten op het lokaal assenstelsel van de muurprojectie volgens de stelling van Pythagoras met de afstanden in het wereldassenstelsel van het punt tot het eerste scanpunt. Hierbij begint het eerste scanpunt op x_0 van het lokaal assenstelsel.

$$x_{n,muur} = \sqrt{\left(y_{n,map} - y_{0,map}\right)^2 + \left(x_{n,map} - x_{0,map}\right)^2} \tag{4.10}$$

$$x_{0,muur} = 0 \tag{4.11}$$

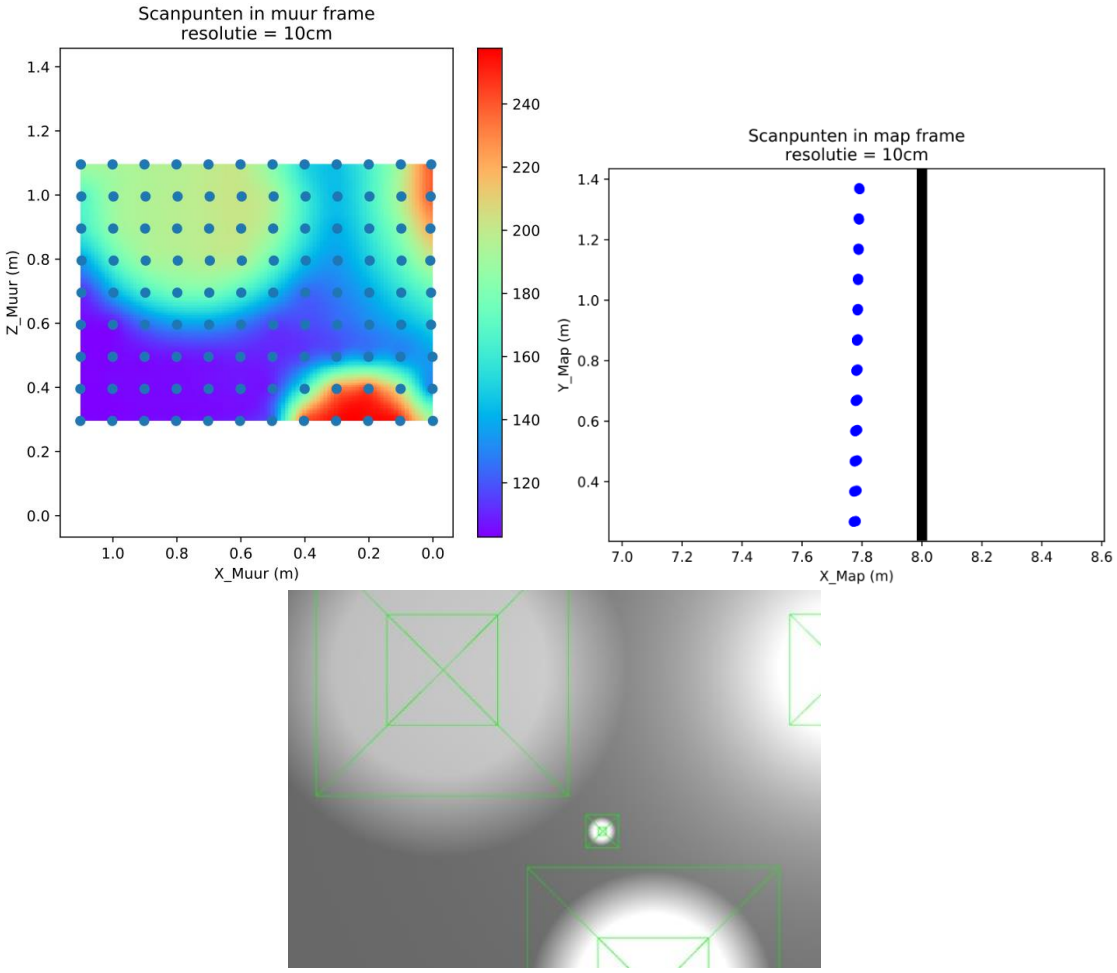
Het programma vormt hierna de lijst met x- en z-coördinaten en verlichtingssterktes om in een *dataframe*. Dit dataframe is een tabel die de scandata op de muur voorstelt met op elk coördinaat een waarde voor de verlichtingssterkte. Vervolgens genereert het programma een tweedimensionale *heatmap* van de verlichtingswaarden aan de hand van de *Pcolor* functie uit de *matplotlib plottingsbibliotheek* [76]. Bijkomend worden de verlichtingssterktes in de gebieden tussen scanpunten geïnterpoleerd om een verloop ervan te tonen. Dit gebeurt in Python met behulp

van de *scipy.interpolate.Rbf* functie [77]. Op deze manier brengt de heatmap de locaties van de bronnen op de muur in kaart. Hieruit zijn de coördinaten in het wereldassenstelsel van verschillende bronnen niet direct af te leiden. Daarom toont een bijkomend bovenaanzicht de locatie van de scans in de ruimte. Op deze manier zijn beide voorstellen gekoppeld aan elkaar om de bronnen op het vlak van de muuroriëntatie en de positie ervan in de ruimte in kaart te brengen.

4.3 Resultaten en bespreking

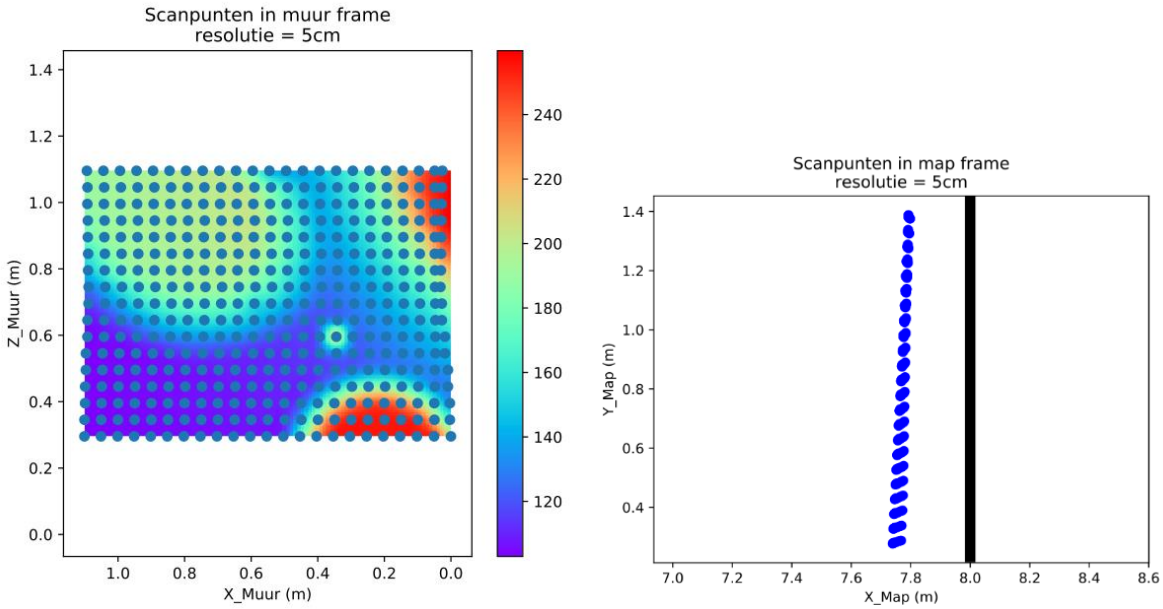
4.3.1 Scan van afzonderlijk gedeelte van een muur

De eerste simulatietest die bestaat uit een enkele scan van een afzonderlijk muurstuk toont de aanwezigheid en sterkte van de bronnen in de muur zoals te zien op Figuur 4.13. Deze scan gebeurde met een tussenafstand van 10 cm tussen opeenvolgende scanpunten en tussen opeenvolgende rijen.



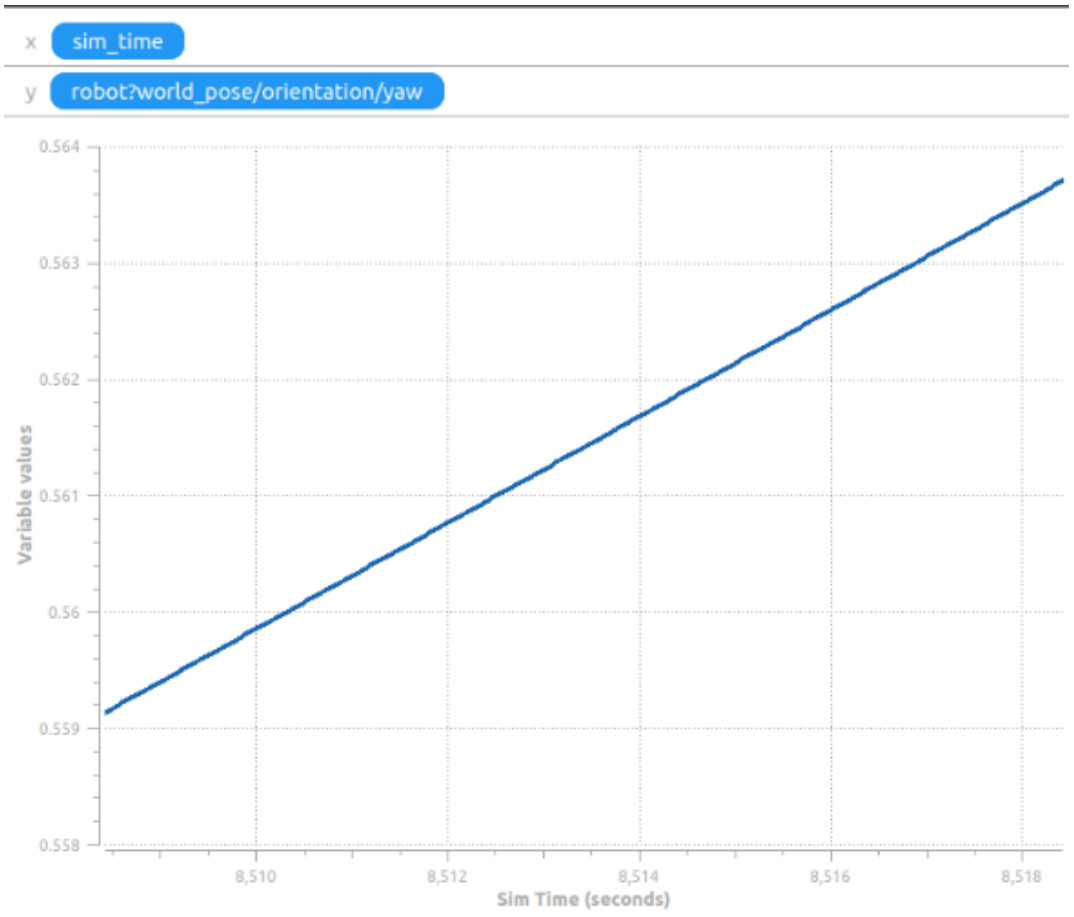
Figuur 4.13: Scan van afzonderlijk muurstuk met 10 cm tussen ieder scanpunt. (links) Visualisatie van de bronnen op het vlak van de muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur. (onderaan) De gescande lichtbronnen in Gazebo.

Deze scan toont duidelijk de verschillende onderlinge locaties en sterktes van de individuele bronnen in lux, de eenheid van verlichtingssterkte. Door de kleine resolutie, is de kleine bron tussen de drie grote bronnen niet zichtbaar. Onderstaande Figuur 4.14 toont een scan met grotere resolutie.



Figuur 4.14: Scan van afzonderlijk muurstuk met 5 cm tussen ieder scanpunt. (links) Visualisatie van de bronnen op het vlak van de muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur.

Op deze scan met grotere resolutie is de kleine bron wel individueel zichtbaar tussen de drie grote bronnen. Hoewel de interpolatie tussen scanpunten dit deels maskeert, is op sommige plaatsen duidelijk variatie zichtbaar op de uniforme verdeling tussen sommige scanpunten. Dit komt door een drift van het robotmodel in Gazebo. Deze drift zorgt voor een kleine verandering van de basispose in de kaart tijdens een scan. De translatiedrift wordt reeds geminimaliseerd door de massa van het robotmodel voldoende laag in te stellen. Dit zorgt soms echter voor het bijkomend driftprobleem waarbij de basis, door de kleine massa (en dus kleine inertie), meebeweegt wanneer de manipulator een snelle en grote beweging maakt. Verder update de lokalisatie van de AMCL enkel bij de uitvoering van een nieuwe beweging. Wanneer de robotbasis stilstaat en er een kleine drift optreedt (hoofdzakelijk in de rotatie van de robot), beweegt de robot zonder bewegingscommando's te ontvangen. Hierdoor update AMCL niet en is er geen nieuwe lokalisatie van de robotoriëntatie in de ruimte. De positie van het robotassenstelsel in de ruimte is, met behulp van tf en de robot_state_publisher, ondertussen wel aangepast. Deze ongewenst aangepaste positie wordt wel nog steeds gebruikt voor de berekeningen. Hieruit volgt een fout op de positie van de robot aangezien er een beweging blijkt te zijn die enkel afkomstig is van de drift en niet gecorrigeerd werd door AMCL. Onderstaande Figuur 4.15 toont deze drift op de *yaw*-hoek of de hoek rond de z-as van de robot, in functie van de simulatietijd. Gedurende deze tijdsperiode werd geen enkel snelheidscommando gestuurd naar de controller maar is er toch een variatie op de rotatie.



Figuur 4.15: Drift in Gazebo op de yaw-hoek of oriëntatie rond de z-as van de robot in functie van de simulatietijd.

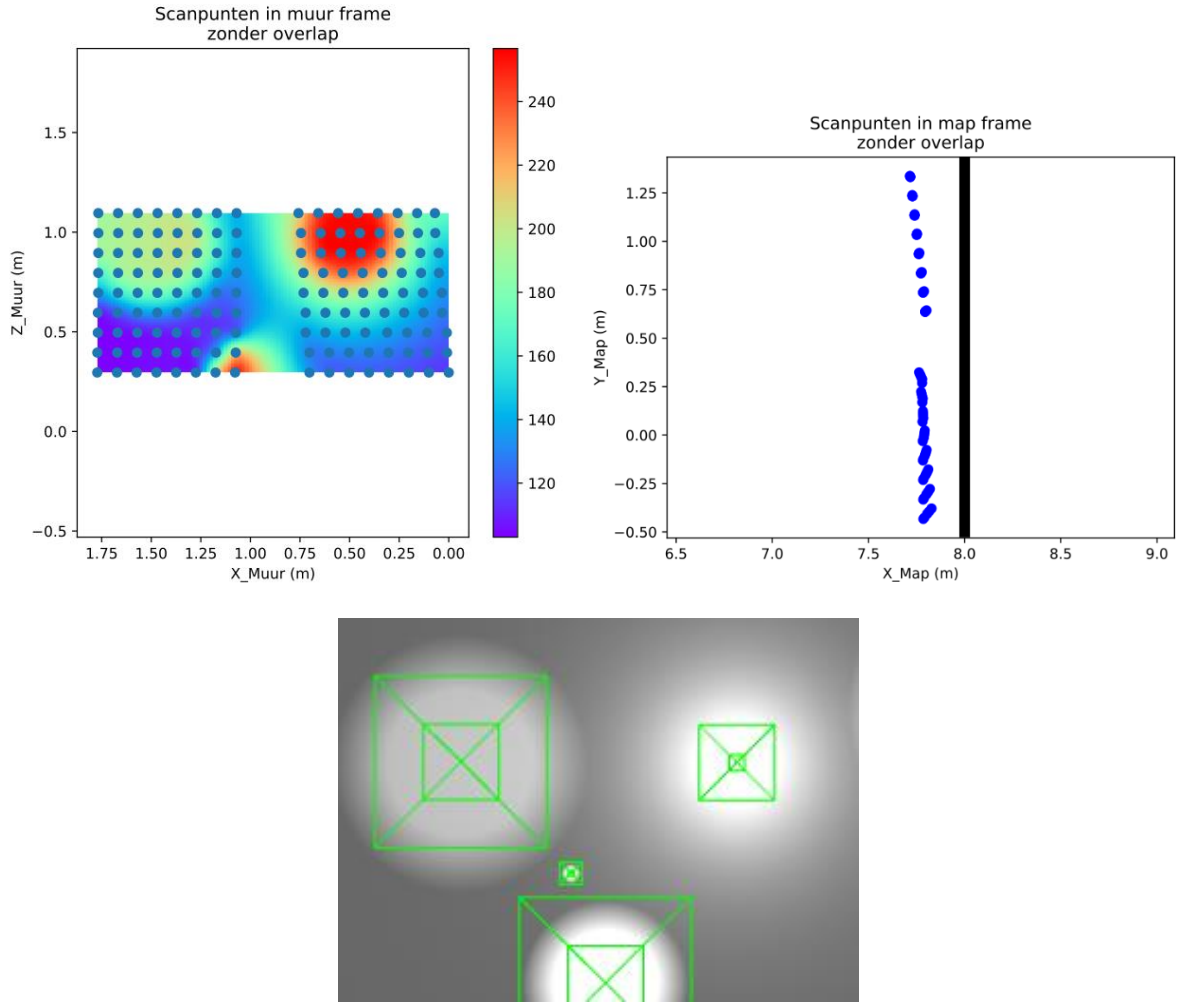
Deze drift is zowel zichtbaar op de heatmap als op het bovenaanzicht van de scan. Naarmate de scantijd oploopt en de manipulator steeds hogere rijen doorloopt, blijft de drift aanwezig en neemt de fout op de oriëntatie toe.

Een grotere resolutie door een kleinere afstand tussen opeenvolgende scanpunten en tussen opeenvolgende rijen, betekent echter dat de scan langer duurt. Bijkomend zorgt een langere scan ervoor dat de drift in Gazebo een groter effect heeft op de robotoriëntatie. Hierdoor is de drift beter zichtbaar bij langere scans en ontstaat er een grotere fout. Vandaar dat deze drift weinig zichtbaar is bij de scan met een lagere resolutie. Bijgevolg moet er steeds een afweging gemaakt worden tussen een grotere resolutie en een langere scantijd die gepaard gaat met grotere driftfouten in Gazebo.

De drift op het robotmodel blijkt uit de literatuur een vaak voorkomend probleem. Een mogelijke oorzaak voor een grotere invloed hiervan zou een foute inertie van een bepaalde link in het robotmodel kunnen zijn. Bijkomend werden alle processen en simulaties gelijktijdig uitgevoerd op relatief eenvoudige laptops, namelijk een *HP Elitebook 850 G4* (Intel® Core™ i5-7200U CPU @ 2.50GHz en 16 GB RAM) en een *Xiaomi Mi Notebook Pro* (Intel® Core™ i5-8250U @ 1.60GHz en 16GB RAM). Deze zijn niet specifiek bedoeld voor dergelijke computationeel intensieve taken. Het is mogelijk dat de drift hierdoor hoger is.

4.3.2 Vergelijking twee scans met en zonder overlap

Onderstaande Figuur 4.16 toont de bronnen van de tweede simulatie waarbij twee opeenvolgende muurdelen gescand werden.

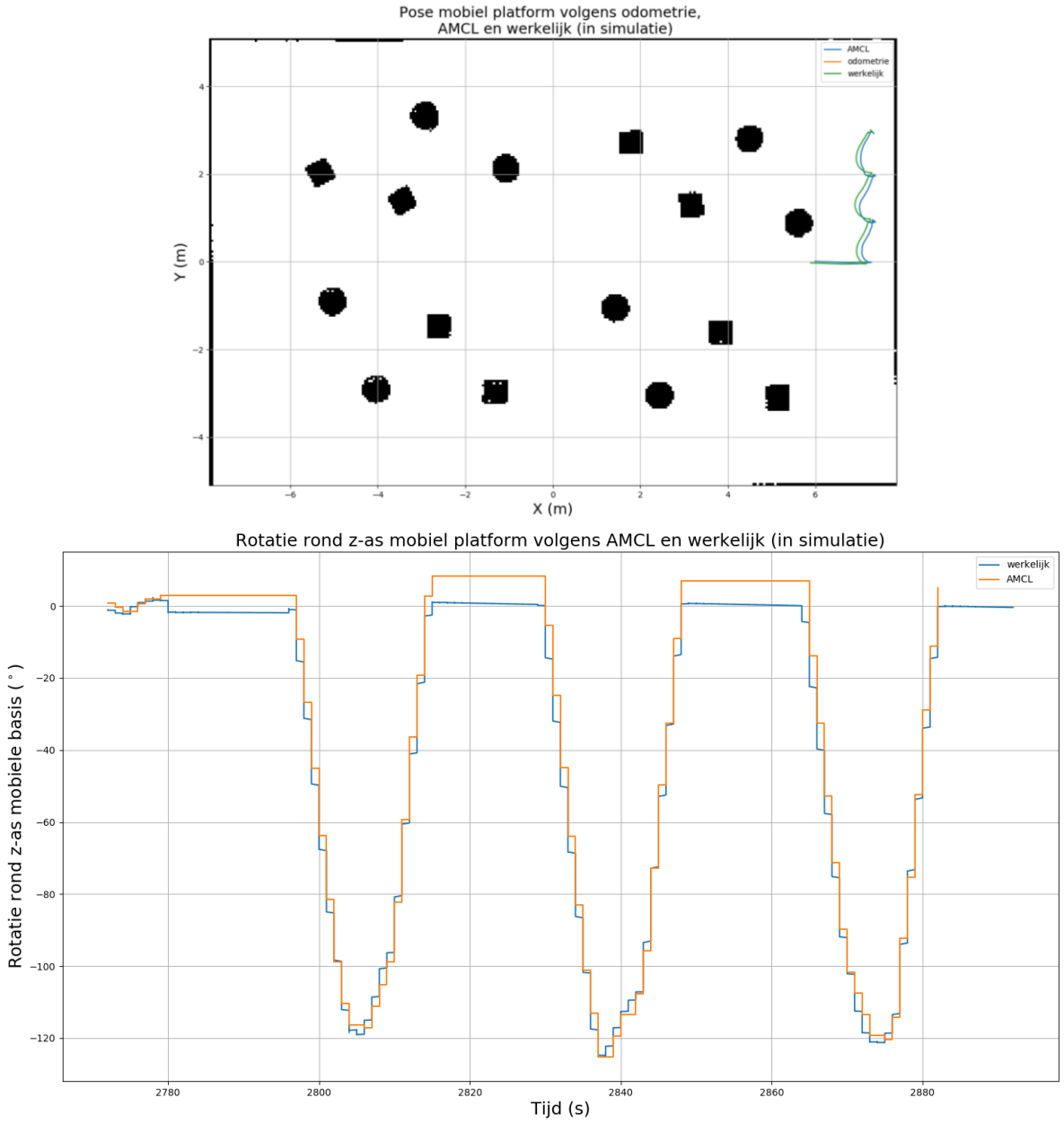


Figuur 4.16: Twee opeenvolgende scans, zonder overlap, van een muurstuk met 10 cm tussen ieder scanpunt. (links) Visualisatie van de bronnen op het vlak van de muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur. (onderaan) De gescande lichtbronnen in Gazebo.

Ook hier is te zien dat er variatie zit op de uniforme verdeling tussen opeenvolgende scanpunten. Zo staan de scanpunten van de rechterscan schuin boven elkaar. Dit komt wederom door de reeds vermelde driftfouten tijdens een individuele scan.

Vervolgens is er een verschil in oriëntatie tussen de twee individuele scans, zoals te zien op het bovenaanzicht. Dit komt door een fout op de oriëntatie van de bereikte doelpositie van de move_base. Nadat de robot de doelpositie en juiste oriëntatie heeft bereikt, roteert het verder dan de gewenste hoek zonder zich na deze laatste rotatie te lokaliseren. Hoewel de robot nu schuiner staat ten opzichte van de muur, wordt dit nooit opnieuw gelokaliseerd en gecorrigeerd. Door deze foute lokalisatie, worden de scanpunten foutief getransformeerd vanuit het schuine (maar recht veronderstelde) manipulatorassenstelsel naar het assenstelsel van de map. Hierdoor hebben, zoals te zien op het bovenaanzicht, de twee opeenvolgende scans van een muurgedeelte niet perfect dezelfde

oriëntatie in de ruimte aangezien de fout op de oriëntatie niet steeds hetzelfde is. Onderstaande Figuur 4.17 toont deze lokalisatiefout.

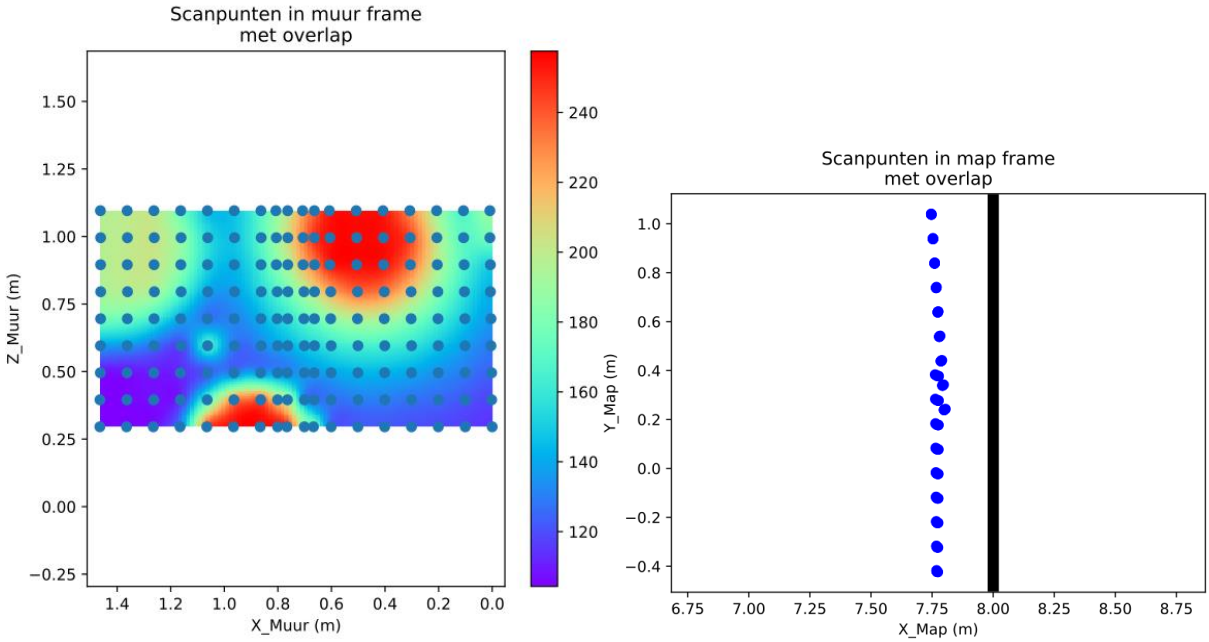


Figuur 4.17: (boven) Gevolgd pad van de mobiele basis om vier scanposities te bereiken. (beneden) Vergelijking van de rotatie rond de z-as van de robot volgens de AMCL en volgens de werkelijke rotatie (in simulatie). Hierbij zijn de horizontale stukken rond 0° de scanposes waarbij een fout van ± 8° te zien tussen de AMCL oriëntatie en de werkelijke.

In bovenstaande vergelijking tussen de AMCL en werkelijke oriëntatie zijn de horizontale stukken de scanposes (recht voor de muur) en de dalen de rotaties om te navigeren naar het volgend scanpunt. Tijdens de scanposes is de drift op de rotatie van de robot in het blauw te zien aangezien de lijn daar niet horizontaal is (en dit wel zou moeten zijn aangezien er geen snelheidscommando's ontvangen worden). Tijdens deze poses is ook te zien dat er een fout van ±8° zit op de AMCL-rotatie ten opzichte van de werkelijke. De AMCL update nog op het eerste punt van een recht stuk van een scanpose en update vervolgens pas wanneer de robot opnieuw beweegt bij het begin van een nieuw dal, terwijl de werkelijke pose (door drift) wel nog verandert.

Door de berekening van de nieuwe doelpositie van de basis uit het laatste scanpunt, is er een spatie tussen twee opeenvolgende scans waardoor er net geen overlap is. De grote spatie zou het gevolg kunnen zijn van de tolerantie op het bereiken van de berekende nieuwe doelpositie of van een foutieve berekening van deze nieuwe doelpositie. Een kleinere tolerantie-instelling van de planner zorgt echter voor oscillaties van het robotmodel bij het bereiken van het doel aangezien de wielcontroller dergelijke nauwkeurigheid niet kan bereiken. Onderstaande simulatie tracht de spatie op te lossen door met opzet voor overlap tussen twee opeenvolgende scans te zorgen. Dit kan door de nieuwe doelpositie niet met het laatste scanpunt te berekenen maar door middel van een eerder punt.

Onderstaande Figuur 4.18 toont de bronnen van de derde simulatie waarbij twee opeenvolgende muurdelen gescand werden met overlap tussen de twee scans.



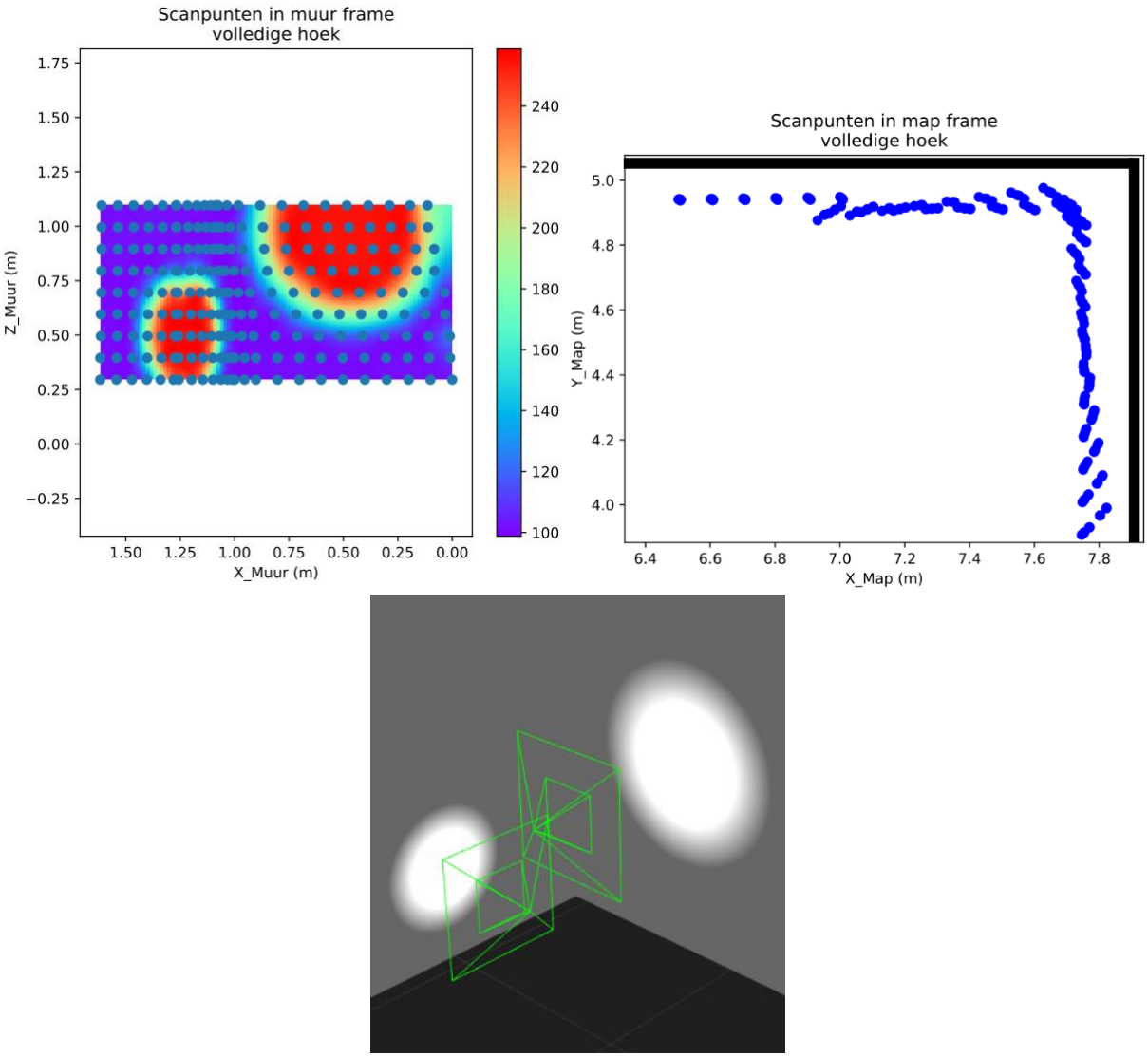
Figuur 4.18: Twee opeenvolgende scans, met overlap, van een muurstuk met 10 cm tussen ieder scanpunt. (links) Visualisatie van de bronnen op het vlak van de muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur.

Hier is op het bovenaanzicht wederom een verschil in oriëntatie tussen de opeenvolgende scans te zien door de vermelde fout op de lokalisatie. De spatie is echter niet meer zichtbaar door voor een overlap te zorgen. Hierdoor wordt het verloop van de bronnen dat eerder in de spatie lag, correcter in kaart gebracht. Deze overlap heeft echter als nadeel dat er meer opeenvolgende scans nodig zijn om een bepaalde muurlengte te scannen.

Bovendien is op de heatmap van deze scan de kleine bron wel zichtbaar ondanks de afstand tussen twee scanpunten 10 cm is. Dit komt doordat het berekende scanpunt toevallig gelijk valt met deze bron.

4.3.3 Meerdere opeenvolgende scans

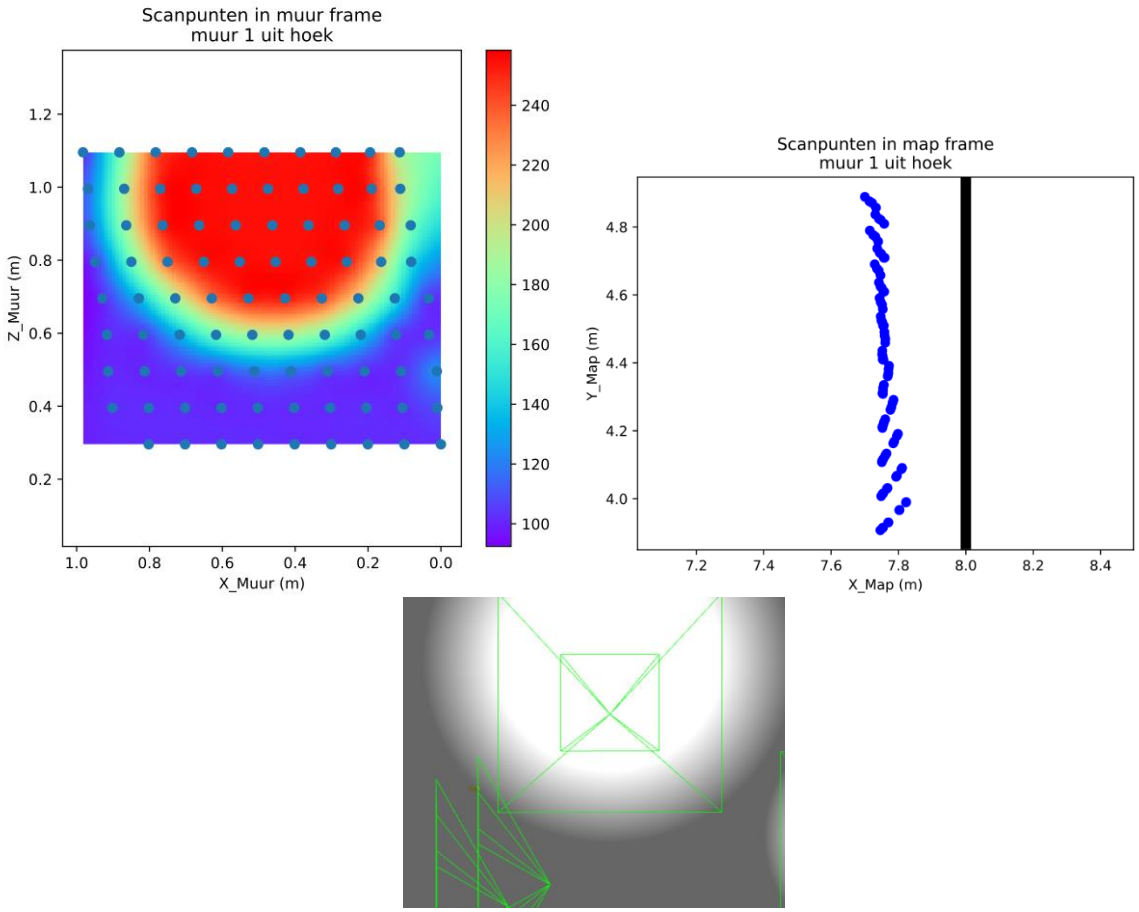
Vervolgens toont onderstaande Figuur 4.19 de simulatie waarbij een hoek gescand werd.



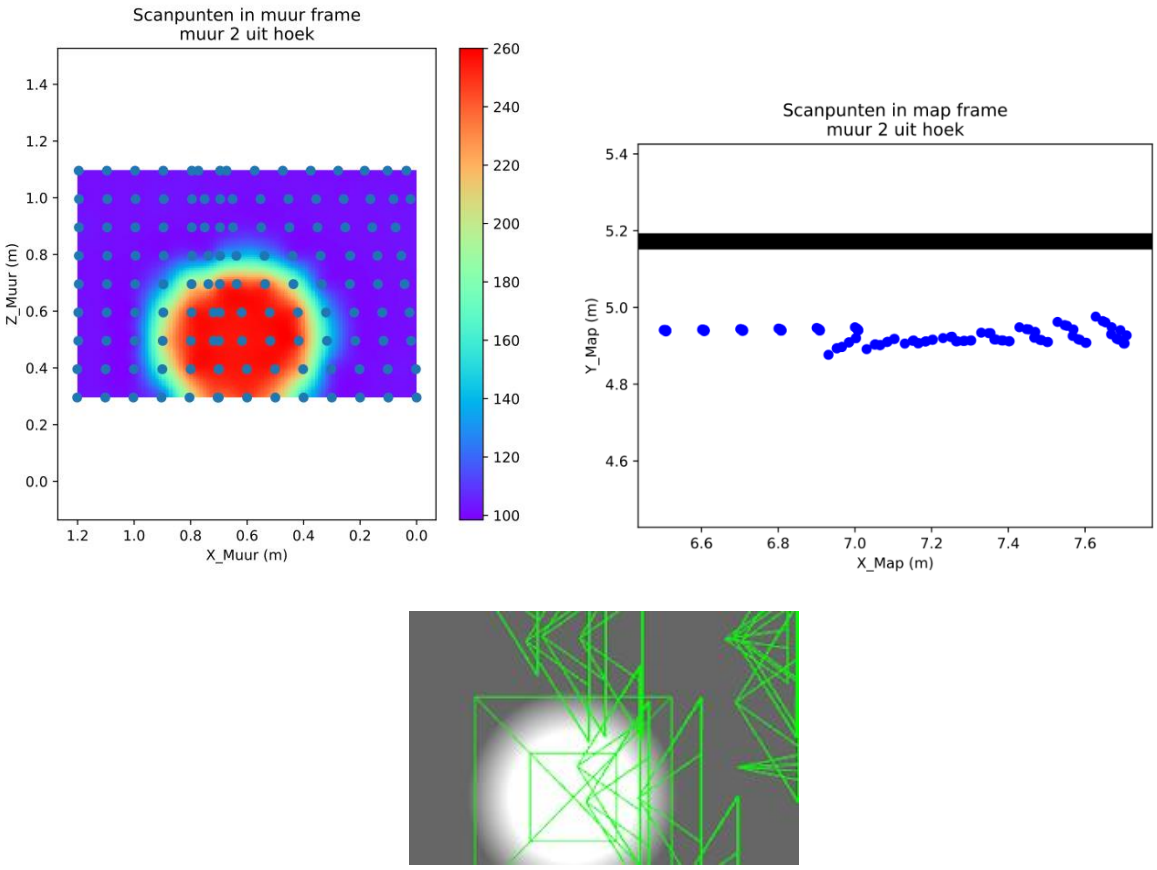
Figuur 4.19: Twee opeenvolgende scans in een hoek met overlap en 10 cm tussen ieder scanpunt. (links) Visualisatie van de bronnen op het vlak van de eerste muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur. (onderaan) De gescande lichtbronnen in Gazebo.

Zoals reeds beschreven bevat de lijst *combined_scan_data* oorspronkelijk alle punten van de scans van de twee muurzijden. Zoals te zien aan het verschil in drift op het bovenaanzicht, bestond deze scan uit twee opeenvolgende scans van eerst de ene muurzijde inclusief de hoek en van vervolgens een stuk van de andere muurzijde. De drift in de scan van de hoek is opvallend groot. Dit komt door het reeds besproken fenomeen waarbij de basis door een lage massa beweegt bij de beweging van de manipulator van de ene zijde naar de andere.

Aangezien deze heatmap een projectie is van een hoek op een vlak, zijn de bronnen vervormd. Door vervolgens deze globale lijst in het CSV-bestand manueel op te splitsen in een lijst van elke muurzijde, wordt een correcter beeld verkregen van de bronnen op elke muurzijde. Deze opsplitsing gebeurt op basis van de eindeffector pose in het scanpunt. Figuur 4.20 Figuur 4.21 tonen deze scans voor elke muurzijde afzonderlijk.



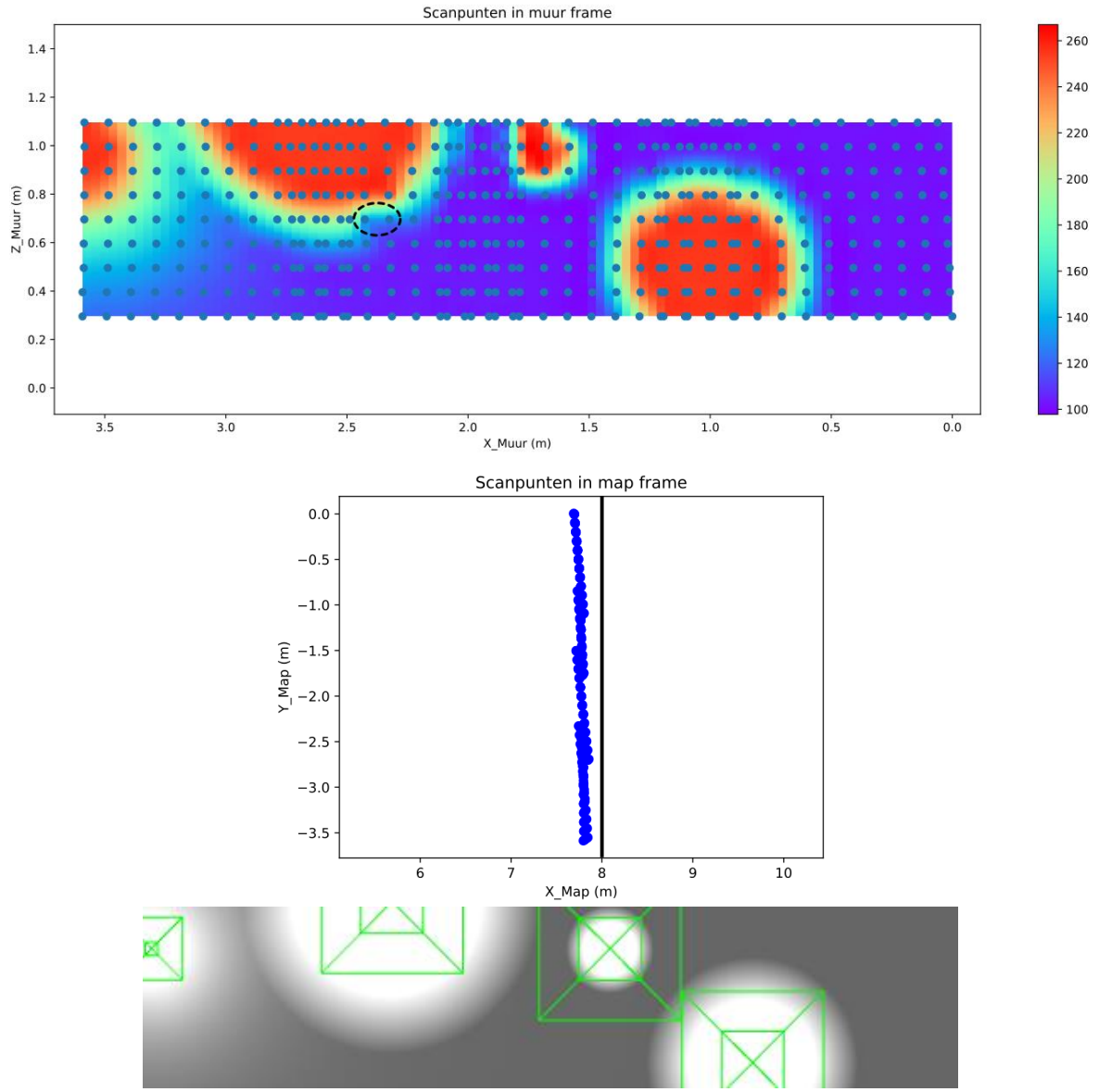
Figuur 4.20: Scan van de eerste muurzijde van een hoek met 10 cm tussen ieder scanpunt. (links) Visualisatie van de bronnen op het vlak van de muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur. (onderaan) De gescande lichtbronnen in Gazebo.



Figuur 4.21: Scan van de tweede muurzijde van een hoek met 10 cm tussen ieder scanpunt. (links) Visualisatie van de bronnen op het vlak van de muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur. (onderaan) De gescande lichtbronnen in Gazebo.

Door de visualisatie van de bronnen op te splitsen per muurzijde, tonen de heatmaps geen vervorming van de bronnen.

Bij het scannen van een langere muur moeten er aantal opeenvolgende scans uitgevoerd worden. Figuur 4.22 toont het resultaat van een scanopdracht die zoveel mogelijk scans achterelkaar uitvoert, om zo een grotere sectie van een muur in kaart te brengen.



Figuur 4.22: Vier opeenvolgende scans met overlap en 10 cm tussen de scanpunten. (links) Visualisatie van de bronnen op het vlak van de eerste muuroriëntatie. (rechts) Bovenaanzicht van de scanpunten voor de muur.

Op de heatmap zijn twee opeenvolgende scanpunten in het zwart omcirkeld. Hierbij heeft de planning van de RRT-Connect planner van het rechterpunt naar het linkerpunt gefaald (geen pad gevonden binnen de toegelaten planningstijd). Hierdoor bereikt de eindeffector het linkerpunt niet en meet de sensor de lichtsterkte in dit punt niet. Bijgevolg slaat de planner dit punt over en plant het rechtstreeks naar het volgend punt. Het overgeslagen punt krijgt dan automatisch dezelfde verlichtingssterkte toegekend als het vorige punt.

Bij een toenemend aantal bijkomende scans stijgt de kans op falen van het programma van de muurscanprocedure. Zo blijkt vier opeenvolgende scans meestal wel te lukken. Hierdoor is het niet mogelijk om een volledig stuk muur te scannen in dezelfde uitvoering van de procedure. De meest voorkomende fout is een oscillatie rond de doelpositie van de mobiele basis. Hierdoor bereikt de basis de doelpositie niet en bijgevolg kan er geen volgende scan uitgevoerd worden. Een andere mogelijke fout komt van de tf2-transformaties van de scanpunten naar het wereldassenstelsel.

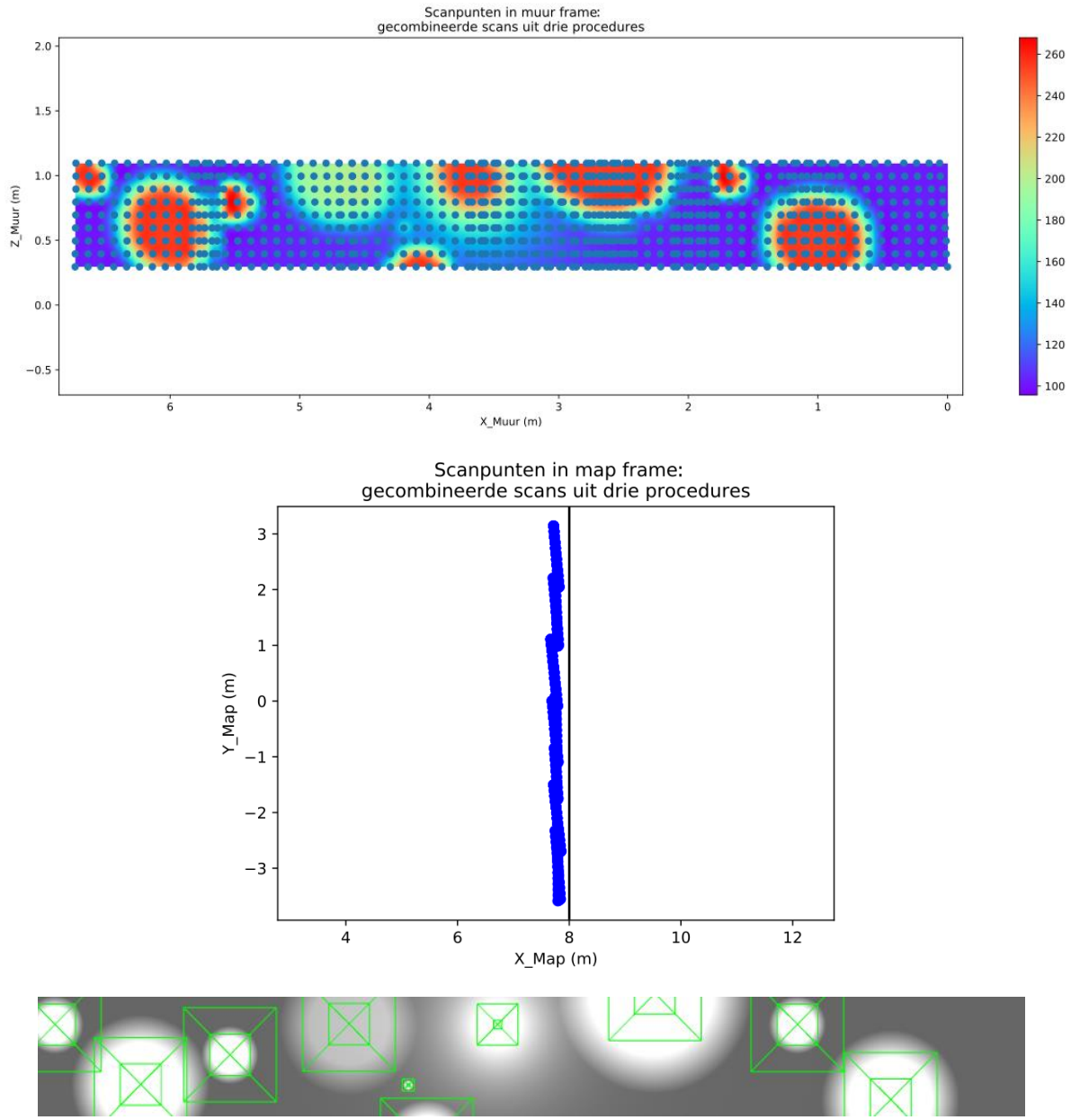
Onderstaande Figuur 4.23 toont deze foutmelding die voorkomt wanneer de tijdsindicaties van bepaalde assenstelsels niet overeenkomt.

```
Traceback (most recent call last):
  File "/home/thijsmang/master_ws/src/wall_scan/scan_wall.py", line
 30, in <module>
    listener.waitForTransform("base_manipulator", "map", rospy.Time
.now(), rospy.Duration(15.0))
  File "/opt/ros/melodic/lib/python2.7/dist-packages/tf/listener.py
", line 76, in waitForTransform
    raise tf2_ros.TransformException(error_msg or "no such transfor
mation: \"{}\" -> \"{}\"".format(source_frame, target_frame))
tf2.TransformException: Lookup would require extrapolation into the
 future.  Requested time 4320.444000000 but the latest data is at t
ime 4250.251000000, when looking up transform from frame [map] to f
rame [base_manipulator]
```

Figuur 4.23: Foutmelding omtrent de tijdsindicaties van assenstelsels bij tf2-transformaties na meerdere opeenvolgende scans.

Het is echter wel mogelijk om verschillende muurprocedures van telkens een zo lang mogelijk stuk muur achtereenvolgens uit te voeren. Hierbij loopt een muurscanprocedure zo lang mogelijk, ofwel gedurende 4 opeenvolgende scans, en start men manueel de volgende procedure met de allereerste doelpositie van de basis op het laatste bereikte punt van de vorige procedure. Aangezien elke procedure de totale lijst van scanpunten opslaat, is het mogelijk om verschillende lijsten van opeenvolgende stukken muur in een CSV-bestand achtereen te plaatsen.

Onderstaande Figuur 4.24 toont de combinatie van de scans uit drie aparte muurscanprocedures met 10 cm tussen de scanpunten om een volledige muurzijde te visualiseren. Deze zijn manueel samengevoegd in een CSV-bestand.



Figuur 4.24: Combinatie van de scans uit drie aparte muurscanprocedures met 10 cm tussen de scanpunten om een volledige muurzijde te visualiseren. (boven) Visualisatie van de bronnen op het vlak van de muuroriëntatie. (midden) Bovenaanzicht van de scanpunten voor de muur. (beneden) De gescande lichtbronnen in Gazebo.

Dit toont aan dat het mogelijk is om een muur te scannen met lengte naar wens. Aangezien de scanpunten allen gedefinieerd zijn in hetzelfde wereldassenstelsel, is het mogelijk om ze allemaal te transformeren naar hetzelfde lokaal assenstelsel.

4.3.4 Algemene bespreking padplanning

De A*-planner van de mobiele basis heeft tijdens geen enkele test uit de use case of tijdens voorgaande testritten van de navigation stack problemen getoond. Deze plant voldoende rond obstakels en voert zelfs een herplanning van het originele pad uit wanneer de DWA-planner het geplande pad niet kan volgen. De DWA-planner vereiste zeer veel *finetuning* van de parameters om een aanvaardbaar gedrag te bekomen in relatie met de wielcontrollers. Aangezien het door de COVID-19 maatregelen niet meer mogelijk was om de werkelijke snelheids- en

versnellingsparameters te achterhalen van de werkelijke wielmotoren van AMBER, werden de parameters van de DWA-planner zo ingesteld om een zo gunstig mogelijk gedrag te vertonen in combinatie met de wielcontrollers.

De RRT-Connect planner van de manipulator plant in sommige gevallen trajecten met een lange omweg door grote bewegingen. Dit werd voor de use case reeds geminimaliseerd door de manipulator voor aanvang van een scanuitvoering reeds naar een configuratie te plannen waarbij dergelijke omwegen niet voorkomen tussen openvolgende scanpunten. Deze lange omwegen van trajecten komen bijkomend meer voor bij scans in een hoek van de muur aangezien de arm daar minder plaats heeft. Dit gedrag is typisch voor dergelijke samplegebaseerde planners en zou eventueel vermeden kunnen worden door gebruik te maken van een Cartesiaanse planner van de manipulator. In sommige gevallen faalt de planning naar een volgend scanpunt van de RRT-Connect planner. Wanneer de voorziene tijd om een plan te vinden overschreden wordt doordat de planner geen plan vindt, slaat de planner het punt over en plant het rechtstreeks naar het volgend scanpunt.

4.4 Conclusie

Dit hoofdstuk besprak allereerst de opbouw van een use case om de ontkoppelde padplanningsmethode te testen. Hierbij moet AMBER autonoom bepaalde stukken muren scannen om geïmiteerde radiologische bronnen in kaart te brengen. Vervolgens besprak het de manier waarop lichtbronnen de radiologische bronnen nabootsen en de manier waarop een aangepaste Gazebo cameraplugin deze lichtbronnen kan meten.

Hierna overloopt een pseudocode de volgorde van en de samenwerking tussen verschillende programma's om een muurscanprocedure uit te voeren. Hierbij navigeert AMBER naar een muur, haalt een programma uit 2D-laserscandata de muuroriëntatie om hieruit een reeks te scannen punten te definiëren op een bepaalde afstand van de muur. Dit gebeurt met een split-and-merge-algoritme en een reeks van analytische berekeningen. Hierna doorloopt de manipulator de padplanning tussen opeenvolgende scanpunten in een zigzagpatroon dat de muur aftast. De positie en verlichtingswaarden worden bijgehouden in elk punt. Uit het laatste scanpunt berekent een programma de nieuwe doelpositie van een volgende scan van de muur. Vervolgens navigeert AMBER naar deze volgende doelpositie en herhaalt het stappenplan zich om zo met behulp van opeenvolgende scans een stuk muur te scannen. Na de scans, visualiseert een heatmap tot slot de lichtsterktes van de scanpunten op een vlak van een lokaal assenstelsel volgens de muuroriëntatie.

De gebruikte methode toont dat het mogelijk is om met een ontkoppelde padplanningsmethode geïmiteerde radiologische bronnen in kaart te brengen door muren af te tasten. Dit gaat goed voor een beperkt gedeelte van de muur of voor enkele opeenvolgende scans. Hierin zijn de locaties en groottes van de bronnen duidelijk te zien. Dit gebeurt echter niet optimaal voor langere stukken van een muur en vereist verschillende aparte, aan elkaar gekoppelde muurscanprocedures.

Uit verschillende simulaties blijkt dat de lokalisatie van de basis in de kaart en het bereikte doel van de basis niet optimaal nauwkeurig is. Drift op het robotmodel in Gazebo en onzekerheid op de bereikte oriëntatie van de mobiele basis zijn hiervan de grootste oorzaak. Dit heeft bijgevolg ook invloed op de interactie van de manipulator met de omgeving. De simulatieproblemen afkomstig van de drift zouden zich echter niet in werkelijkheid kunnen voordoen. Wegens de COVID-19 maatregelen was het niet mogelijk om dit te testen.

De afzonderlijke padplanners werken zonder grote problemen. Door gebruik van een ontkoppelde aanpak, gaat de padplanningsmethode echter niet optimaal om met de redundantie in de kinematische structuur van AMBER. Door de reeds beschreven onnauwkeurigheden is er zelfs nood aan verdere ontkoppeling tussen basis en manipulator. Deze verdere ontkoppeling gebeurt doordat de manipulatorplanning gebruikt maakt van 2D-laserscandata van de muur in plaats van de reeds gekende lokalisatie van de basis in de ruimte.

5 Algemene conclusie

Dit hoofdstuk vormt een algemene conclusie omtrent de masterproef. Dit gebeurt door allereerst na te gaan indien de onderzoeksvragen voldoende beantwoord zijn en op welke manier dit gebeurde door de beoogde doelstellingen te voltooien. De daaropvolgende sectie werpt een kritische reflectie op de resultaten, zowel goede zaken als zaken die verbeterd kunnen worden. Tot slot stelt de laatste sectie onderwerpen voor van volgend onderzoek om de padplanning van mobiele manipulatoren verder te optimaliseren.

5.1 Antwoorden op onderzoeksvragen en behaalde doelstellingen

De opdracht van de masterproef bestond uit de implementatie van hardware (i.e. sensoren) en software (i.e. padplanners) voor een mobiele manipulator dat autonoom kan rondrijden en taken kan uitvoeren in een statische omgeving. Dit betreft de autonome lokalisatie en navigatie van de robot zoals het in kaart brengen van de omgeving en de padplanning doorheen de omgeving om een taak uit te voeren. Als use case bestaat deze taak uit het autonoom rondrijden en aftasten van een muuroppervlak in de omgeving om geïmiteerde radiologische bronnen in kaart te brengen. Wegens de maatregelen betreffende COVID-19 gebeurde dit enkel in simulatie voor een robotmodel in plaats van ook op een werkelijke mobiele manipulator, zoals origineel gepland.

Om deze opdracht te voltooien, moet allereerst volgende hoofdonderzoeksvraag beantwoord zijn:

- Welke mogelijke padplanners bestaan er en welke mogelijkheden zijn er om zowel het mobiel platform als de robotarm aan te sturen?

Uit een literatuurstudie naar bestaande padplanningsmethoden volgt allereerst kennis omtrent de verder gebruikte technieken zoals de meest bekende padplanningsalgoritmes namelijk het Dijkstra en A*-algoritme, kaarttransformaties zoals variabele celdecompositie, RDT en RRT, lokale obstakelvermijdende planners zoals de DWA. Verder volgt uit de literatuurstudie kennis over de twee verschillende denkwijzen voor het controleren van mobiele manipulatoren met een groot aantal en vooral redundante vrijheidsgraden. Dit betreft enerzijds de beredeneerde, vooropgeplande modelgebaseerde aanpak die nood heeft aan een exacte weergave van de omgeving en van de te manipuleren objecten. Deze aanpak bestaat uit twee soorten strategieën waarbij de ene soort een opsplitsing of aanpassing van de volledige configuratieruimte hanteert terwijl de andere soort de volledige configuratieruimte van de mobiele manipulator in zijn geheel beschouwt. Anderzijds heeft de reactieve behavior-based aanpak minder nood aan dergelijke exacte modellen door gebruik te maken van zogenaamde constraints om het robotgedrag te controleren.

Uit de literatuur blijkt echter een gebrek aan open-source, praktische implementaties van gecombineerde padplanningsmethoden. Verder zit de meerderheid van de methoden nog in onderzoeksfasen. Dit toont dat mobiele manipulatoren nog niet industrieel toepasbaar zijn wegens hun complexe besturing.

Uit een vergelijking van verschillende varianten van de padplanningsmethoden blijken de BI²RRT*-methode, die in de volledige configuratieruimte plant, en de traditionele, ontkoppelde methode die de arm en basis apart controleert als het best toepasbaar op AMBER. Dit volgt vooral uit het feit dat de overige methoden nog in onderzoeksfasen zitten, weinig open-source zijn en weinig praktische voorbeelden hebben van hun implementatie. Echter blijkt de ontkoppelde methode het meest haalbaar binnen de masterproef wegens de eenvoudige ROS-integratie en de complexe aanpassingen

vereist aan de code van de BI²RRT*-methode. Deze code is namelijk bedoeld voor holonomische robots, terwijl AMBER niet-holonomisch is. De aanpassing van deze code is, in kader van deze masterproef, te complex en uitgebreid. Vandaar maakt deze masterproef gebruik van een ontkoppelde padplanningsmethode met het A*-algoritme als globale planner van de basis, de DWA als lokale planner van de basis en de OMPL RRT-Connect planner als planner voor de manipulator. Op deze manier worden mobiele basis en manipulator afzonderlijk gecontroleerd.

Verder helpen bijkomende antwoorden op enkele hulponderzoeksfasen om de opdracht te voltooien:

- Op welke manier is dergelijke padplanner toepasbaar op AMBER?

De implementatie van de ontkoppelde padplanningsmethode gebeurt in het ROS-raamwerk dat instaat voor de communicatie tussen verschillende processen. Dit raamwerk zorgt bijkomend voor de koppeling tussen de simulatieomgeving Gazebo en de visualisatieomgeving RViz. Het URDF-bestand werd aangepast en uitgebreid om met behulp van zogenaamde ROS-controllers het robotmodel te besturen in Gazebo. Verder zorgt de navigation stack van ROS voor de configuratie en uitvoering van 2D-padplanning van de mobiele basis. Dit maakt gebruik van een a priori gekende kaart, lokalisatie met AMCL en de move_base node met kostenkaarten en padplanningsalgoritmes. Hierin zijn verschillende algoritmes in te stellen, waaronder een A* globale planner en de DWA als lokale planner voor obstakelvermijding. Anderzijds staat MoveIt! in voor de padplanning van de manipulator doorheen de driedimensionale ruimte. Ook hierin zijn verschillende algoritmes te kiezen, waaronder de RRT-Connect planner uit de OMPL-bibliotheek. Hoewel specifieke aanpassingen vereist zijn afhankelijk van de robot, is het dankzij de open-source aard van ROS en met enige basiskennis van het ROS-raamwerk mogelijk om een volledige mobiele manipulator te modelleren en te simuleren in het ROS-raamwerk. Hierbij is de gekozen padplanningsmethode makkelijk toepasbaar op het robotmodel.

- Op welke manier functioneert AMBER met dergelijke planner bij het autonoom uitvoeren van taken in een statische omgeving?

Een use case zorgt voor een praktische validatie van de geïmplementeerde planners voor AMBER. Dit betreft het autonoom rondrijden en aftasten van een muuroppervlak in een simulatieomgeving om geïmiteerde radiologische bronnen in kaart te brengen. De afzonderlijke planners, namelijk de A*- en DWA-planner en de RRT-Connect planner, functioneren zonder problemen. De lokalisatie van de basis in de kaart en het bereikte doel van de basis is echter niet optimaal nauwkeurig. Uit de simulaties blijkt drift op het robotmodel in Gazebo en een fout op de bereikte oriëntatie van de mobiele basis als grootste oorzaak hiervoor. Dit heeft bijgevolg ook invloed op de interactie van de manipulator met de omgeving. Hoewel de padplanningsmethode bruikbaar is om dergelijke taken in simulatie uit te voeren, is het niet optimaal en is het niet mogelijk om een volledige ruimte in één keer te scannen.

- Hoe gaat de planner in een efficiënte manier om met redundantie in de kinematische structuur van AMBER?

Door gebruik te maken van een ontkoppelde aanpak, gaat de padplanningsmethode niet optimaal om met de redundantie in de kinematische structuur van AMBER. Door de reeds beschreven

onnauwkeurigheden is er zelfs nood aan verdere ontkoppeling tussen basis en manipulator. Dit gebeurt doordat de manipulatorplanning gebruikt maakt van 2D-laserscandata van de muur in plaats van te vertrouwen op de reeds gekende lokalisatie van de basis in de ruimte.

Met behulp van antwoorden op deze onderzoeksvragen, wordt voldaan aan volgende doelstellingen. Enerzijds de hoofddoelstelling, namelijk de implementatie van hardware (i.e. sensoren in simulatie) en software (i.e. padplanners) voor een mobiele manipulator dat autonoom kan rondrijden en taken kan uitvoeren in een statische simulatieomgeving. Dit gebeurt volledig in het ROS-raamwerk met RViz en Gazebo.

- het gebruik van sensoren, zoals laserscanners of camera's, voor de besturing (waaronder kaartopbouw) van de mobiele manipulator;

Dit gebeurt met de 2D-lasersensor van het merk Hokuyo voor de 2D-navigatie van de mobiele basis en met de Microsoft Kinect 3D-camera voor de 3D-obstakelvermijding van de manipulator.

- het plannen van een botsingsvrij pad om van een startpositie naar een eindpositie te gaan (deterministisch en/of stochastisch);

Dit gebeurt enerzijds met de navigation stack voor de mobiele basis en anderzijds met de MoveIt! plugin van RViz voor de manipulator.

- het besturen van meerdere vrijheidsgraden (> 6), waarbij een deel van de vrijheidsgraden mobiel zijn (i.e. een mobiel platform dat beschikt over drie DoF) en een deel statisch zijn (i.e. een industriële robotarm dat beschikt over zes DoF).

Dit gebeurt door een ontkoppelde padplanningsmethode die de mobiele basis en de manipulator apart controleert. De toepassing van gecombineerde methoden vereist uitgebreider onderzoek.

Deze implementaties werden allen getest en aangetoond met behulp van de use case in Gazebo.

Deze masterproef draagt enerzijds bij aan het onderzoek naar mobiele manipulatoren van het onderzoekscentrum ACRO en anderzijds aan het ARCHER-project. Dit gebeurt allereerst door de categorisering van verschillende padplanningsmethoden en door te staven dat de meerderheid van de gecombineerde methoden nog niet industrieel toepasbaar is. Verder zorgt de ontwikkeling van een robotmodel van de bestaande mobiele manipulator AMBER voor de mogelijkheid om verschillende soorten padplanningstesten uit te voeren in simulatie. Deze simulaties tonen dat het mogelijk is om met een ontkoppelde padplanningsmethode geïmiteerde radiologische bronnen in kaart te brengen door muren af te tasten. Dit gaat goed voor een beperkt stuk van de muur waarvan de locaties en groottes van de bronnen duidelijk te zien zijn. Dit gebeurt echter niet optimaal voor langere stukken van een muur en vereist verschillende aparte, aan elkaar gekoppelde muurscanprocedures. Deze simulaties blijken niet optimaal om grootschalige testen uit te voeren maar tonen wel hun nut om een robotmodel op te bouwen en planners te integreren en te testen die daarna op de werkelijke robot toegepast worden.

5.2 Reflectie

Door gebrek aan praktische voorbeelden van open-source padplanningsmethoden die in de volledige configuratieruimte plannen, was het binnen de masterproef helaas niet mogelijk om dergelijke planner toe te passen op AMBER. Zulke planner zou echter nuttiger geweest zijn voor het onderzoek van ACRO. Dergelijke soort planner maakt meer gebruik van de integratie van een manipulator op een mobiele basis om gelijktijdig en in samenwerking tussen elkaar de arm en basis te besturen.

Door gebrek aan ervaring met ROS en Gazebo duurde het langer dan voorzien om het volledig werkend robotmodel op te bouwen. Hierdoor was er binnen de masterproef geen tijd om andere padplanningsmethoden te integreren en te testen op het robotmodel. Zo konden bijvoorbeeld andere planners ingesteld worden in de navigation stack of in de MoveIt! padplanning van de manipulator. Het zou verder ook mogelijk geweest zijn om bijkomend toch een gecombineerde padplanner, zoals de BI²RRT*-planner, proberen te integreren. Het zou ook interessant geweest zijn om andere use cases uit te werken, zoals bijvoorbeeld een industriële pick-and-place opdracht, om na te gaan hoe de planners omgaan met redundantie in de kinematische structuur van AMBER.

De use case vereist, buiten de muur, niet veel obstakelvermijding tussen opeenvolgende scans. Dit zou de simulatietijd echter onnodig vergroten aangezien de padplanning van de basis reeds getest wordt tijdens de navigatie naar de eerste scan en obstakels tussen scans niet veel bijbrengen. Indien gekozen wordt voor een gecombineerde planning, zou het wel interessant zijn om te testen hoe de planning verloopt tussen opeenvolgende scans waarbij de eindeffector tijdens het bewegen van de basis nog steeds de muur volgt. In dit geval zou een complexere planning noodzakelijk zijn om ervoor te zorgen dat de eindeffector de muur volgt terwijl de basis obstakels vermijdt.

Een andere aanpak voor het scannen van de muur zou zijn om zijdelings lang de muur te rijden terwijl de manipulator de muur aftast. Hiervoor moet het robotmodel, meer bepaald de sensoren, aangepast worden om de muur zijdelings te herkennen. Vervolgens zou ook de code aangepast moeten worden voor het plannen van de arm aan de zijkant van de robot in plaats van recht voor de basis. Door tijdgebrek was het niet mogelijk om de gebruikte methode hiernaar uit te breiden.

Helaas konden geen testen uitgevoerd worden op AMBER in werkelijkheid wegens de maatregelen omtrent COVID-19. Het was oorspronkelijk wel de bedoeling om na de simulaties het gedrag van de geïntegreerde planner in werkelijkheid te testen. Dit toont hoe nuttig het robotmodel van AMBER is om padplanners te testen in simulatie. Dankzij het robotmodel en de simulatieomgevingen was het toch mogelijk om zaken te testen en resultaten te bekomen ondanks het gebrek aan de werkelijke robot. Bijkomend is dit robotmodel nuttig voor verder onderzoek om simulatietesten op uit te voeren.

5.3 Toekomstig onderzoek

Toekomstig onderzoek kan allereerst de voorgestelde ontkoppelde padplanningsmethode integreren op de werkelijke AMBER en dit testen met de use case uit de simulatie (of andere use cases) nagebouwd in werkelijkheid. Dit gebeurt wederom met ROS maar vereist een minder complex robotmodel. Hierbij is er geen nood aan controllers voor de aansturing in simulatie en geen nood aan simulatie van sensordata.

Verder kan toekomstig onderzoek zich focussen op gecombineerde padplanningsmethoden om optimaal gebruik te maken van de redundante vrijheidsgraden. Voor taken zoals het aftasten van een muur lijkt een behavior-based aanpak, zoals het eTaSL-raamwerk, zeer interessant [78]. Hierbij zorgen constraints ervoor dat de eindeffector de muur aftast en volgt de mobiele basis, terwijl het obstakels vermijdt, naargelang het opschuiven van de eindeffector op de muur.

Literatuurlijst

[1] KU Leuven, “ACRO.” [Online]. Available: https://iiw.kuleuven.be/onderzoek/acro.

[2] UHasselt, “Nuclear Technology Centre (NuTeC) : project R-9397.” [Online]. Available: https://www.uhasselt.be/UH/Research-groups/en-projecten_DOC/en-project_details.html?pid=16459&t=en&oid=1929. [Accessed: 20-Sep-2010].

[3] T. Cardeynaels and S. Del Gallo, “Ontwerpen, kalibreren en aansturen van een autonome mobiele gidsrobot [eindwerk],” Diepenbeek: Gezamenlijke opleiding Industriële Ingenieurswetenschappen UHasselt & KU Leuven, 2019.

[4] Morgan Quigley *et al.*, “ROS: an open-source Robot Operating System,” *ICRA Work. Open Source Softw.*, vol. 3, p. 5, 2009.

[5] E. Demeester, “Autonomous mobile robots locomotion, kinematics, perception, localisation, navigation [cursus],” Diepenbeek: Gezamenlijke opleiding Industriële Ingenieurswetenschappen UHasselt & KU Leuven, 2019.

[6] S. M. Lavalle, *PLANNING ALGORITHMS*. Illinois: Cambridge University Press, 2006.

[7] W. H. Wilson, “Methods of Search,” 2001. [Online]. Available: http://www.cse.unsw.edu.au/~billw/Justsearch.html. [Accessed: 25-Nov-2019].

[8] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” in *Numerische Mathematik 1*, 1959, pp. 269–271.

[9] K. Wang, “Compare A* with Dijkstra algorithm,” 2015. [Online]. Available: http://plainaslife.blogspot.com/2015/02/compare-with-dijkstra-algorithm.html. [Accessed: 26-Nov-2019].

[10] N. Correl, “Introduction to Robotics #4: Path-Planning,” 2011. [Online]. Available: http://correll.cs.colorado.edu/?p=965. [Accessed: 27-Nov-2019].

[11] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge: The MIT Press, 2011.

[12] J. Pan and D. Manocha, “Efficient Configuration Space Construction and Optimization for Motion Planning,” *Engineering*, vol. 1, no. 1, pp. 046–057, Mar. 2015.

[13] I. A. Şucan, M. Moll, and L. Kavraki, “The open motion planning library,” *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, 2012.

[14] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, “Path planning and trajectory planning algorithms: A general overview,” in *Mechanisms and Machine Science*, vol. 29, Carbone Giuseppe and F. and Gomez-Bravo, Eds. Cham: Springer International Publishing, 2015, pp. 3–27.

[15] S. Temizer, “Comparison of the Human Model and Potential Field Method for Navigation,” 2001. [Online]. Available: http://people.csail.mit.edu/lpk/mars/temizer_2001/Method_Comparison/index.html. [Accessed: 27-Nov-2019].

[16] J. Borenstein and Y. Koren, “the Vector Field Histogram -,” *IEEE J. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, 1991.

[17] D. Fox, W. Burgard, and S. Thrun, “The Dynamic Window Approach to Collision Avoidance,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 137–146, 1997.

[18] E. Marder-Eppstein, “dwa_local_planner,” 2018. [Online]. Available: http://wiki.ros.org/dwa_local_planner. [Accessed: 07-Dec-2019].

[19] E. Demeester, M. Nuttin, D. Vanhooydonck, G. Vanacker, and H. Van Brussel, “Global dynamic window approach for holonomic and non-holonomic mobile robots with arbitrary cross-section,” *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, pp. 2694–2699, 2005.

[20] V. K. Pilania, “Safe Motion Planning under Uncertainty for Mobile Manipulators in Unknown Environments,” SIMON FRASER UNIVERSITY, 2015.

[21] K. M. Lynch and C. L. Park, *Modern Robotics. Mechanics, Planning, and Control*. 2017.

[22] V. Pilania and K. Gupta, “A hierarchical and adaptive mobile manipulator planner,” *IEEE-RAS Int. Conf. Humanoid Robot.*, vol. 2015-Febru, pp. 45–51, 2015.

[23] V. Pilania and K. Gupta, “Mobile manipulator planning under uncertainty in unknown environments,” *Int. J. Rob. Res.*, vol. 37, no. 2–3, pp. 316–339, 2018.

[24] S. Chitta, B. Cohen, and M. Likhachev, “Planning for autonomous door opening with a mobile manipulator,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1799–1806, 2010.

[25] M. Likhachev, G. Gordon, and S. Thrun, “ARA *: Anytime A * with Provable Bounds on,” *Science (80-.).*, pp. 767–774, 2014.

[26] K. Gochev, A. Safonova, and M. Likhachev, “Planning with adaptive dimensionality for mobile manipulation,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 2944–2951, 2012.

[27] F. Burget, M. Bennewitz, and W. Burgard, “BI 2 RRT*: An Efficient Sampling-Based Path Planning Framework for Task-Constrained Mobile Manipulation,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 3714–3721, 2016.

[28] D. Berenson, J. Kuffner, and H. Choset, “An optimization approach to planning for mobile manipulation,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1187–1192, 2008.

[29] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed RRT∗: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” *IEEE Int. Conf. Intell. Robot. Syst.*, no. Iros, pp. 2997–3004, 2014.

[30] P. Lehner and A. Albu-Schaffer, “The Repetition Roadmap for Repetitive Constrained Motion Planning,” *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3884–3891, 2018.

[31] S. Huang, “Task Learning and Execution for Behavior-based Mobile Manipulation,” Leuven: Department of Mechanical Engineering KU Leuven, 2011.

[32] B. J. W. Waarsing, M. Nuttin, and H. Van Brussel, “Behavior-based mobile manipulation inspired by the human example,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 1, pp. 268–273, 2003.

[33] J. De Schutter and H. Van Brussel, “Compliant Robot Motion I. A Formalism for Specifying Compliant Motion Tasks,” *Int. J. Rob. Res.*, vol. 7, no. 4, pp. 3–17, 1988.

[34] H. Bruynickx and J. De Schutter, “Specification of Force-Controlled Actions in Task Space,”

KU Leuven, 1999.

[35] J. De Schutter *et al.*, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *Int. J. Rob. Res.*, vol. 26, no. 5, pp. 433–455, 2007.

[36] P. Soetens, "A software framework for real-time and distributed robot and machine control." K.U.Leuven. Faculteit Ingenieurswetenschappen, Leuven, 2006.

[37] P. Soetens and H. Bruyninckx, "Realtime hybrid task-based control for robots and machine tools," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2005, no. April, pp. 259–264, 2005.

[38] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the OROCOS project," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2003, vol. 2, pp. 2766–2771.

[39] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots in cluttered environments," *IEEE Int. Conf. Robot. Autom.*, pp. 572–577, 1990.

[40] A. M. Romero, "ROS/ Concepts," 2014. [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed: 10-May-2020].

[41] E. Marder-Eppstein, V. Pradeep, and M. Arguedas, "actionlib," 2018. [Online]. Available: http://wiki.ros.org/actionlib. [Accessed: 10-May-2020].

[42] D. Hershberger, D. Gossow, and J. Faust, "Rviz," 2018. [Online]. Available: http://wiki.ros.org/rviz. [Accessed: 10-May-2020].

[43] MoveIt, "MoveIt." [Online]. Available: https://moveit.ros.org/. [Accessed: 10-May-2020].

[44] Gazebo, "Gazebo - Robot simulation made easy." [Online]. Available: http://gazebosim.org/. [Accessed: 10-May-2020].

[45] H. Deng, J. Xiong, and Z. Xia, "Mobile manipulation task simulation using ROS with MoveIt," *2017 IEEE Int. Conf. Real-Time Comput. Robot. RCAR 2017*, vol. 2017-July, pp. 612–616, 2018.

[46] I. Sucan and J. Kay, "urdf," 2019. [Online]. Available: https://wiki.ros.org/urdf. [Accessed: 10-May-2020].

[47] R. Ventura, "urdf/XML/Link," 2011. [Online]. Available: http://library.isr.ist.utl.pt/docs/roswiki/urdf(2f)XML(2f)Link.html. [Accessed: 23-Apr-2020].

[48] Gazebo, "Tutorial: Using a URDF in Gazebo," 2014. [Online]. Available: http://gazebosim.org/tutorials/?tut=ros_urdf. [Accessed: 10-May-2020].

[49] V. D. Lu and J. Kay, "joint_state_publisher," 2020. [Online]. Available: http://wiki.ros.org/joint_state_publisher. [Accessed: 10-May-2020].

[50] I. Sucan, J. Kay, and W. Meeussen, "robot_state_publisher," 2020. [Online]. Available: http://wiki.ros.org/robot_state_publisher. [Accessed: 10-May-2020].

[51] T. Foote, E. Marder-Eppstein, and W. Meeussen, "tf," 2017. [Online]. Available: http://wiki.ros.org/tf.

[52] MoveIt, "MoveIt! Setup Assistant," 2018. [Online]. Available:

http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html.

[53] I. Sucan, “srdf,” 2019. [Online]. Available: http://wiki.ros.org/srdf. [Accessed: 09-May-2020].

[54] P. F, “Hardware Interfaces,” 2020. [Online]. Available: https://github.com/ros-controls/ros_control/wiki/hardware_interface. [Accessed: 30-Apr-2020].

[55] Gazebo, “Tutorial: Ros Control,” 2020. [Online]. Available: http://gazebosim.org/tutorials/?tut=ros_control. [Accessed: 04-May-2020].

[56] W. Meeussen, “ros_control,” 2020. [Online]. Available: https://wiki.ros.org/ros_control. [Accessed: 03-May-2020].

[57] S. Chitta *et al.*, “ros_control: A generic and simple control framework for ROS,” *J. Open Source Softw.*, vol. 2, no. 20, p. 456, 2017.

[58] B. Magyar, “diff_drive_controller,” 2019. [Online]. Available: http://wiki.ros.org/diff_drive_controller. [Accessed: 06-May-2020].

[59] A. R. Tsouroukdissian, “joint_trajectory_controller,” 2018. [Online]. Available: http://wiki.ros.org/joint_trajectory_controller. [Accessed: 30-Apr-2020].

[60] E. Marder-Eppstein, “navigation,” 2019. [Online]. Available: http://wiki.ros.org/navigation. [Accessed: 04-May-2020].

[61] B. Gerkey, “gmapping,” 2019. [Online]. Available: http://wiki.ros.org/gmapping?distro=melodic. [Accessed: 04-May-2020].

[62] C. Stachniss, G. Grisetti, and W. Burgard, “Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters,” *IEEE Trans. Robot.*, vol. 23(1), pp. 34–46, 2007.

[63] B. Gerkey, “amcl,” 2019. [Online]. Available: https://wiki.ros.org/amcl. [Accessed: 06-May-2020].

[64] D. Fox, “KLD-sampling: Adaptive particle filters,” *Adv. Neural Inf. Process. Syst.*, 2002.

[65] E. Marder-Eppstein, “move_base,” 2018. [Online]. Available: http://wiki.ros.org/move_base. [Accessed: 10-May-2020].

[66] E. Marder-Eppstein, “costmap_2d,” 2018. [Online]. Available: http://wiki.ros.org/costmap_2d. [Accessed: 10-May-2020].

[67] V. D. Lu, “global_planner,” 2019. [Online]. Available: http://wiki.ros.org/global_planner?distro=melodic. [Accessed: 10-May-2020].

[68] E. Marder-Eppstein, “dwa_local_planner,” 2018. [Online]. Available: http://wiki.ros.org/dwa_local_planner?distro=melodic. [Accessed: 10-May-2020].

[69] J. J. Kuffner and S. M. La Valle, “RRT-connect: an efficient approach to single-query path planning,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, no. Icra, pp. 995–1001, 2000.

[70] R. Parasuraman, “Sec. 4: Creating a light sensor plugin,” 2018. [Online]. Available: https://github.com/SMARTlab-Purdue/ros-tutorial-gazebo-simulation/wiki/Sec.-4:-Creating-a-light-sensor-plugin. [Accessed: 15-May-2020].

[71] M. Patrick, “image_view,” 2016. [Online]. Available: http://wiki.ros.org/image_view. [Accessed: 21-May-2020].

[72] Octomap, “Octomap,” 2016. [Online]. Available: http://octomap.github.io/octomap/doc/. [Accessed: 24-May-2020].

[73] K. Conley, “rospy,” 2017. [Online]. Available: http://wiki.ros.org/rospy. [Accessed: 24-May-2020].

[74] P. So, D. Coleman, and M. Lautman, “Move Group Python Interface,” 2018. [Online]. Available: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/move_group_python_interface/move_group_python_interface_tutorial.html. [Accessed: 24-May-2020].

[75] T. Foote, E. Marder-Eppstein, and W. Meeussen, “tf2,” 2019. [Online]. Available: http://wiki.ros.org/tf2. [Accessed: 20-May-2020].

[76] J. Hunter, D. Dale, E. Firing, and M. Droettboom, “matplotlib.axes.Axes.pcolor,” 2020. [Online]. Available: https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.pcolor.html#matplotlib.axes.Axes.pcolor. [Accessed: 21-May-2020].

[77] The SciPy Community, “scipy.interpolate.Rbf,” 2019. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.Rbf.html. [Accessed: 23-May-2020].

[78] E. Aertbeliën and J. De Schutter, “Etasl/eTC: A Constraint-Based Task Specification Language and Robot Controller Using Expression Graphs,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1540–1546.