2014•2015
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: energie

Masterproef

Experimentele evaluatie van botsingsvrije trajectgeneratie voor 3D random bin picking

Promotor :
Prof. dr. ir. Eric DEMEESTER
dr. ir. Johan BAETEN

Joris Beuls
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

2014•2015

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: energie

Masterproef

Experimentele evaluatie van botsingsvrije trajectgeneratie voor 3D random bin picking

Promotor :
Prof. dr. ir. Eric DEMEESTER
dr. ir. Johan BAETEN

Joris Beuls

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

universiteit hasselt | KU LEUVEN

Woord vooraf

In september 2014 kreeg ik de mogelijkheid om een onderzoekstopic voor mijn masterproef te kiezen. Het onderwerp dat voor mij meteen boven de andere uitstak was dat rond *3D random bin picking*. Dit onderwerp trok mijn aandacht omdat er robotica en visietechnologie aan te pas komt. Het is hier dat mijn interesse ligt en waar ik meer van wou leren. Door deze masterproef heb ik geleerd wat onderzoek inhoudt en kijk ik terug op een fijne en leerzame periode.

Door de brede toepasbaarheid van de padplanning en volumetrische voorstelling van een omgeving heb ik ook interesse ontwikkeld voor mobiele robotica. Het is in dit gebied dat ik later ook meer onderzoek zou willen verrichten.

Graag wil ik nog enkele personen bedanken die mij door de hele periode goed begeleid hebben. Zonder hen had ik deze scriptie niet op deze manier kunnen volbrengen. Prof. dr. ir. Eric Demeester wil ik bedanken voor zijn begeleiding doorheen het hele jaar, zijn visie, zijn inzicht en de gesprekken doorheen de masterproef. Prof. dr. ir. Johan Baeten bedank ik voor zijn steun doorheen het jaar. Dr. Jeroen Lievens wil ik bedanken voor de hulp bij het schrijven van deze scriptie. Ook wil ik Geert en Maarten bedanken voor het mee opvolgen van de masterproef, de samenwerking, hun kennis en de leuke sfeer. Ten slotte bedank ik nog mijn ouders om me de mogelijkheid te bieden om verder te studeren en me gedurende deze periode te steunen en mijn vriendin om er steeds voor me te zijn.

Inhoudsopgave

Lijst van figuren

Abstract

ACRO in Diepenbeek onderzoekt de mogelijkheid tot *3D random bin picking* m.b.v. de open bron codes ROS en MoveIt!. Om grote mechanische omsteltijden en –kosten te vermijden, neemt een robot idealiter zelf producten uit een bak die kortbij staat, en plaatst die producten in de productiemachine, het zogenaamde *random bin picking*. In de huidige industriële context is manuele programmatie van robots economisch niet interessant. In plaats daarvan is automatische botsingsvrije trajectgeneratie vereist. Deze masterproef heeft als doel een robotplanner zelf een botsingsvrij pad te laten zoeken, een object te grijpen uit een doos en dit vervolgens botsingsvrij te verplaatsen. De *bin picking* wordt eerst in simulatie getest en vervolgens uitgevoerd op een Epson 6-assige robot.

De simulatie wordt uitgevoerd in rviz, de visualisatieomgeving van ROS. Ten eerste wordt hier getest hoe de omgeving van de robot kan worden voorgesteld. Ten tweede wordt de padplanning uitgevoerd m.b.v. de “MoveIt!” softwarebibliotheek. Het padplanningsalgoritme voor het bepalen van een botsingsvrij pad is RRT-Connect en is gekozen op basis van de resultaten van een eerdere masterproef. De padplanning wordt uiteindelijk getest op de robot.

M.b.v. MoveIt! is het gelukt om in simulatie aan botsingsvrije *bin picking* te doen. Er worden padplanningstijden behaald van 7s. Daarna is ook communicatie met de robot gelegd en is de robot aangestuurd volgens het berekende pad. De praktische haalbaarheid van *bin picking* met botsingsvrije trajectgeneratie is hiermee bewezen.

Abstract in English

ACRO at Diepenbeek researches the possibility of 3D random bin picking by adopting ROS and MoveIt!. To reduce high mechanical changeover time and cost, the robot ideally picks the products from a bin that's near it, and places these products into a production machine, the so called random bin picking. In the current industrial context, manual programming of robots is economically not interesting. Instead, automatic collision-free trajectory generation is required. This master's thesis researches the possibility to let a robot find a collision-free path on its own, grab an object from a bin and move it without colliding with the environment. The bin picking is first tested in simulation and then on an Epson 6-axis robot.

The simulation is executed in rviz, the visualization environment of ROS. First is tested how the environment of the robot can be represented. Secondly the path planning is executed with the help of the MoveIt! software library. The path planning algorithm to calculate the collision-free path is RRT-Connect and is chosen based on the results of a previous master's thesis. The collision-free paths are eventually tested on the robot.

With the help of MoveIt!, collision-free bin picking was possible in the simulated environment. Path planning times of 7s were obtained. After that, communication with the robot was made and the robot was actuated to points from the planned path. The practical feasibility of *bin* picking based on collision-free trajectories is therefore proven.

1 Inleiding

Dit hoofdstuk licht het onderwerp van dit masterproefonderzoek toe. Allereerst situeert sectie 1.1 het onderzoek. Vervolgens formuleert sectie 1.2 de probleemstelling. Daarna beschrijft sectie 1.3 de doelstellingen en onderzoeksvragen van dit onderzoek. De gebruikte software en opstelling worden uitgelegd in sectie 1.4. Sectie 1.5 beschrijft de bijdragen en voornaamste besluiten van dit werk met tot slot eindigt sectie 1.6 met een overzicht van de volledige masterproef.

1.1 Situering

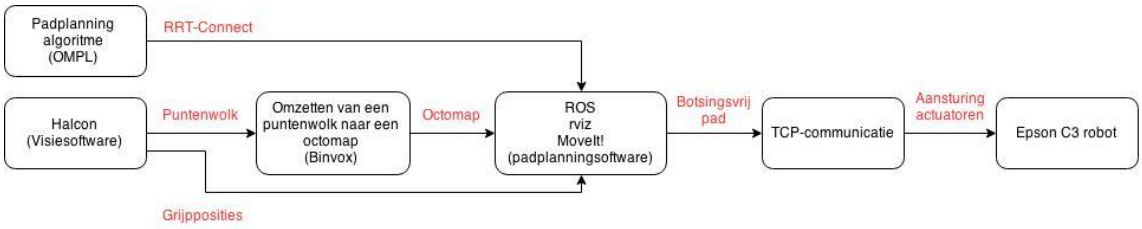
Deze masterproef vindt plaats in ACRO (AutomatiseringsCentrum Research en Opleiding), een onderzoeksgroep van de KULeuven en UC Leuven Limburg. ACRO volgt voortdurend de evoluties in machinevisie en robotica op en implementeert geïntegreerde robot- en visiesystemen in uiteenlopende projecten. Eén van de automatiseringstoepassingen waarnaar sterke vraag bestaat vanuit de industrie is *random bin picking*. Hierbij worden werkstukken niet op een heel nauwkeurige manier aan een industriële robot aangeboden voor verwerking, zoals momenteel typisch het geval is, maar wel in een bak (*bin*) waarin de producten op een niet gekende, willekeurige (*random*) manier liggen. De robot dient de locatie van de werkstukken in deze bak autonoom te bepalen, de stukken te grijpen (*picking*) en er vervolgens nuttige bewerkingen op toe te passen.

Een volledige bin picking cyclus bestaat uit de volgende stappen. Eerst wordt met een 2D of 3D visiesysteem een beeld genomen van de bak, waarna de op te nemen objecten worden herkend en gelocaliseerd. Deze stap resulteert typisch in een 3D puntenwolk en een set van mogelijke grijpposities voor de robotarm.

Vervolgens worden de grijpposities en puntenwolk naar de padplanningssoftware gestuurd die a.d.h.v. een padplanningsalgoritme een botsingsvrij pad zoekt naar de grijppositie. Zodra er een pad is gevonden wordt dit naar de robot gestuurd om de beweging te laten uitvoeren.

ACRO heeft in een PWO-project een eerste *random bin picking* demo-opstelling gerealiseerd, waarbij de nadruk vooral lag op het detecteren en lokaliseren van objecten in de bak met behulp van het computervisiepakket Halcon [1]. De aansturing van de robot om werkstukken daarna ook te grijpen, waarbij botsingen met andere werkstukken of de bak vermeden moeten worden, is echter nog niet in detail onderzocht.

De opdracht in deze masterproef bestaat enerzijds uit de evaluatie van het softwarepakket “MoveIt!” [2] voor botsingsvrije robottrajectgeneratie en anderzijds uit het opbouwen van communicatie tussen de visiesoftware Halcon, de robot en de trajectplanningsoftware MoveIt! die octomappen [3] gebruikt om padplanning en obstakelontwijking mogelijk te maken. Een octomap is een driedimensionale volumetrische voorstelling van de omgevingsruimte met behulp van *voxels*, wat kleine kubussen zijn. De padplanning wordt geïntegreerd op een 6-assige Epson robot. Meer concreet zal de 3D-puntenwolk die Halcon genereert omgevormd worden naar een octomap. Aan de hand van een octomap kan de padplanningsoftware via verschillende algoritmes een botsingsvrij pad plannen voor de robot om een object op te nemen en te verplaatsen. In figuur 1 is een overzicht van de communicatie tussen de verschillende systemen te zien, met de data die verstuurd wordt.



Figuur 1: Overzicht van de gebruikte systemen en hun data uitwisseling.

1.2 Probleemstelling

Het ontwerp van een robotcel gebeurt vandaag de dag typisch eerst door middel van offline-programmatie en simulatie, waarna het robotprogramma in een tweede fase verder aangepast wordt aan de echte robotcel. Bij het offline programmeren wordt typisch in simulatie de robotopstelling gemaakt en verbonden met de programmeersoftware. Zo kan de programmeur de robot programmeren en het gedrag controleren. Dit wordt gedaan om het robotprogramma klaar te maken voordat de robot en zijn periferie zijn gemonteerd. Het in bedrijf nemen van een robotcel kan dan sneller gebeuren. Hierbij komt naderhand nog veel calibratie- en parametreerwerk aan te pas om een robot een bepaalde handeling uit te laten voeren. Plotse veranderingen in de omgeving worden niet in rekening gebracht en aanpassingen kosten veel werk. Automatische botsingsvrije trajectgeneratie op basis van 3D modellen die bekomen zijn uit sensorinformatie over de omgeving bieden hiervoor een antwoord. M.b.v. de trajectgeneratie is de fijnafstelling en het parametreerwerk niet meer nodig.

De trajectgeneratiesoftware is in staat om zelf de aansturing van de robot te berekenen en uit te voeren door commando's naar de robotcontrollers te sturen. Dit is wat bij het *random bin picking* gebeurt. Door *random bin picking* kan ook bespaard worden op periferie die nodig is om voorwerpen aan te bieden aan de robot. Het plaatsen van een bak met voorwerpen onder een camera is voldoende. Dit kan typisch gedaan worden met behulp van een transpallet of een heftruck. Bijkomend is wel de nood aan visiesoftware voor de verwerking van het beeld en de berekening van de grijpposities. In deze masterproef is gekozen voor een 3D CAD model te vergelijken met het camerabeeld om de voorwerpen in de bak te detecteren. Deze masterproef kadert in de evolutie naar zulke automatische, sensorgebaseerde robotprogrammatie en wil hieraan bijdragen door bestaande open broncode padplanningstechnieken te evalueren op een industriële zesassige robot.

1.3 Doelstellingen en onderzoeksvragen

Het algemene doel van de masterproef bestaat erin de open broncode MoveIt! te evalueren voor *random bin picking* door deze te integreren in een voor handen zijnde *random bin picking* cel uitgerust met Halcon beeldverwerking en een Epson zesassige robot. Deze doelstelling bestaat uit verschillende deelvragen en eisen:

1. Ten eerste moet een manier gezocht worden om de 3D puntenwolk uit Halcon op een tijdsefficiënte manier om te zetten naar een octomap die bruikbaar is voor de padplanningssoftware MoveIt!. Ook de communicatie tussen Halcon en de padplanningssoftware moet worden onderzocht.
2. Halcon berekent per gedetecteerd object verschillende mogelijke grijpposities. Deze grijpposities moeten worden doorgestuurd naar de MoveIt! planner en gecontroleerd worden op bruikbaarheid.

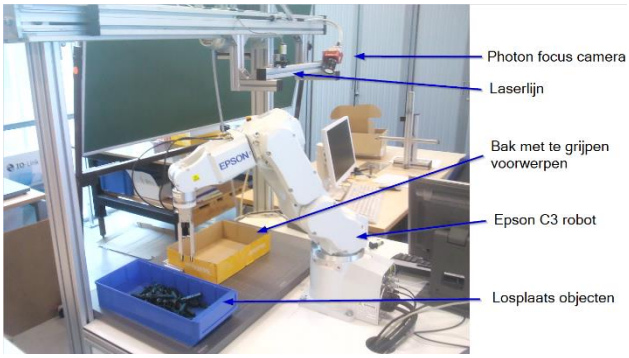
3. Vervolgens moet een keuze gemaakt worden omtrent het meest geschikte padplanningsalgoritme. Hiervoor is een eerdere studie gemaakt [4] die verschillende padplanningsalgoritmes evalueerde in simulatie. Gebruikte evaluatiecriteria zijn de snelheid van het berekenen van een pad, de kwaliteit van het pad zelf en het aantal keren dat het algoritme een goed pad vindt. Deze resultaten, die verworven zijn uit de simulatie, moeten gecontroleerd en vergeleken worden met de resultaten wanneer de padplanning wordt uitgevoerd op de robot.
4. Ten vierde moet de communicatie tussen de padplanningssoftware en de robot worden gemaakt.
5. Daarna moet de padplanningstijd en cyclustijd met de wensen van de industrie vergeleken worden.
6. Ten slotte wordt geëvalueerd of aan de volgende eisen voldaan is:
   - de robot treedt nooit met zichzelf of zijn statische omgeving in botsing;
   - de robot werkt volledig automatisch door gebruik te maken van een padplanningalgoritme;
   - de maximale cyclustijd in de industrie bedraagt typisch 4s, dit geeft een beperking van een padplanningstijd van <1s;
   - het algoritme moet minstens 95% van de keren een geldig botsingsvrij pad als resultaat geven binnen de 1s en in de andere 5% is een langere berekeningstijd toegestaan;
   - er mag nooit een foutief pad als resultaat gegeven worden.

Indien nodig worden ideeën voorgesteld om de padplanning te verbeteren en waar mogelijk ook daadwerkelijk verbeteringen in de C++-implementatie aangebracht.

1.4 Gebruikte software en opstelling

Er wordt een *sheet-of-light* opstelling gebruikt bestaande uit een Photon focus camera die een laserlijn waarneemt. Aan de hand van de vervorming van de laserlijn kan er in Halcon een 3D-puntenwolk gemaakt worden.

De robot die de padplanning zal uitvoeren is een Epson 6-assige robot die in het labo is opgesteld. Voor deze robot zijn meerdere grijpers ter beschikking. Figuur 2 toont de volledige opstelling.



Figuur 2: De bin picking opstelling van ACRO.

Een eerste softwarepakket dat gebruikt is in deze masterproef is Halcon. Dat is een programma dat onder andere in staat is om camerabeelden om te zetten naar een 3Dpuntenwolk in verschillende formaten. Dit programma is ook in staat een 3D-CAD-tekening in te laden en deze te passen op de gegenereerde puntenwolk. Het gevolg is dat de posities en oriëntaties van enkele

herkende objecten gekend zijn en verschillende grijpposities bepaald kunnen worden. Deze zijn nodig om later padplanning te kunnen toepassen.

ROS (Robot Operating System) [5] is een framework waarmee o.a. robotaansturing en visualisatie mogelijk is. ROS is gebaseerd op C++ en Python en wordt in deze masterproef gebruikt om de robotsoftware te schrijven en de simulatie uit te voeren. M.b.v. MoveIt!, een bibliotheek in ROS voor padplanning wordt de robot en de omgeving in de simulatie voorgesteld en de berekening van een botsingsvrij pad uitgevoerd. De OMPL bibliotheek bevat verschillende padplanningalgoritmes die MoveIt! kan gebruiken om een botsingsvrij pad te zoeken.

1.5 Bijdragen

Deze masterproef vult het onderzoek naar *random bin picking* aan. Allereerst is er een literatuurstudie gemaakt over verschillende opstelling bij andere onderzoeksinstellingen. Hieruit kan de padplanningtijd uit de deze masterproef met hun resultaat vergeleken worden.

Vervolgens is er een analyse en evaluatie gemaakt over de omzetting van een puntenwolk naar een octomap. Deze data moet verstuurd worden naar MoveIt! zodat dit geïntegreerd kan worden voor de padplanning. Hiervoor is een C++-programma geschreven dat de octomap inleest en omvormt naar een berichttype dat MoveIt! begrijpt. Daarna is ook een controle uitgevoerd over de juistheid van de voorstelling in rviz.

Ten derde is er op eigen initiatief een handleiding geschreven over het gebruik van ROS en MoveIt!. De handleiding legt alles uit aan de hand van de code die in deze masterproef is geschreven. Deze handleiding is terug te vinden in de bijlage op CD-ROM.

Daarna is een visualisatieomgeving in rviz aangemaakt die de omgeving van de robot benaderd. Hierdoor komt de padplanning in simulatie zo getrouw mogelijk overeen met de realiteit.

Ten vijfde zijn verschillende padplanningsimulaties uitgevoerd in de visualisatieomgeving om de snelheid en het traject te evalueren.

Door het gebrek aan ROS drivers voor de Epson robot, om de robot vanuit ROS aan te sturen, is er voor TCP-communicatie gekozen. Hiervoor is er een C++-programma geschreven dat een verbinding over TCP maakt tussen de laptop met de ROS software en de robot. Hierdoor kan de robot toch aangestuurd worden.

Ten slotte is een C++-programma geschreven om een berekend botsingsvrij pad naar de robot door te sturen en deze daarmee aan te sturen. De volledige kring van het *random bin picken* is hiermee gesloten.

1.6 Overzicht van de masterproef

Hoofdstuk 2 vergelijkt de stand van zaken rond random bin picking met de opstelling op ACRO. Vervolgens legt hoofdstuk 3 uit hoe de puntenwolk wordt omgezet naar de octomap. Daarna, in hoofdstuk 4, wordt de visualisatie en uitvoering van de padplanning in simulatie besproken. Daaropvolgend wordt in hoofdstuk 5 de communicatie tussen planner en robot besproken, eindigend met de resultaten van de padplanning in de realiteit. Ten slotte worden in hoofdstuk 6 de resultaten van de masterproef getoond.

2 Literatuurstudie

2.1 Inleiding

Bin picking wordt gezien als de heilige graal in de automatiseringswereld. De meest gebruikte *bin picking* opstellingen bestaan uit een 3D sensor die opgesteld staat boven een doos, een industriële robot met een grijper en een dataverwerkingseenheid. De opstelling op ACRO bestaat uit soortgelijke componenten.

2.2 Vergelijking van verschillende systemen

Een random bin picking opstelling voert een aantal standaard functies uit. Ten eerste wordt er een beeld gemaakt van de bak met voorwerpen. Vervolgens wordt dan via software het voorwerp in het beeld gedetecteerd (op verscheidene plaatsen omdat het object typisch op verscheidene plaatsen in de bak ligt). De positie van het voorwerp t.o.v. de camera kan dan berekend worden. Deze positie wordt gebruikt om de grijppositie te bepalen. Daarna kan dan een botsingsvrij pad gezocht worden.

In figuur 3 worden enkele systemen met elkaar vergeleken.[7] Hieruit is meteen duidelijk dat er verschillende soorten camera's gebruikt worden om objecten te detecteren. De snelheid waarmee de bin picking wordt uitgevoerd hangt af van de volgende factoren:

- de complexiteit van de vorm van het te grijpen voorwerp;
- de visiesoftware;
- het gebruikte padplanningsalgoritme.

Onderzoeker	Camera	Visiesoftware	Objectherkenning	Opstelling	Padplanningsalgoritme	Cyclustijd
ACRO	Photon focus	Halcon	CAD model	- 1 camera (sheet-of-light) - Laserlijn - Epson C3 robot - bin 400 x 240 x 130 mm	RRT-connect	
ISRA Vision		3D SHAPE	3D scan	- 2 camera's - laser scanner - bin 1200 x 1000 x 800 mm		<2 sec
Tordivel	Scorpion 3D stinger (stereovisie camera)	Scorpion vision software	CAD model	- 1 camera	3DMaMa	
University of Southern Denmark	standaard firewire Industriële camera's	Scape's bin picker Recognition software	SCAPE	- KUKA KR16 - euro pallet met 4 wanden	SCAPE	9,5 sec
Universiteit Leuven	SICK 3D Ranger	Vision++	BinPicker++	- 1 camera - 6 assige robot		4 sec

Figuur 3: Overzicht van verschillende bin picking opstellingen en de resultaten van het padplannen.

2.3 Octomappen

2.3.1 Inleiding

Verschillende robotapplicaties, zoals binnenhuisrobots of ruimtemissies, hebben een driedimensionaal model nodig van hun omgeving om een bepaalde weg te zoeken. Driedimensionale modellen geven een volumetrische voorstelling van de ruimte. Zo een volumetrische voorstelling kan bekomen worden met behulp van octomapping.

Octomapping is gebaseerd op octrees en schat de waarschijnlijkheid waarmee een deel van de ruimte bezet is. De waarschijnlijkheidsparameter heeft een waarde tussen 0 en 1 en geeft aan hoe zeker de meting is dat de voxel vrij of bezet is.

Een octree is een boomstructuur voor volumetrische voorstelling van de omgeving. De omgeving wordt opgesplitst in 8 kubussen van gelijke grootte, zogenaamde voxels. Iedere voxel bezit 8 "kinderen" en kan steeds verder opgesplitst worden (dit indien de voxel bezet is) tot de gewenste resolutie bereikt is.

3D mapping is duidelijk een belangrijke component in hedendaagse robotsystemen maar vormt nog steeds een bottleneck voor onderzoek. Veel basissoftware moet steeds herschreven en bedacht worden. Om dit te beperken hebben A. Hornung et al. een open-source 3D mapping systeem bedacht, namelijk octomap [3].

2.3.2 Vereisten van de volumetrische voorstelling

Het maken van de driedimensionale kaart moet rekening houden met verschillende eisen en problemen. In de volgende alinea's worden deze eisen besproken en hoe de problemen worden opgelost.

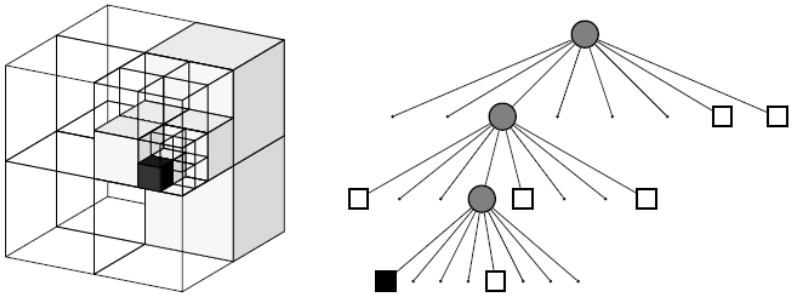
1. De eerste vereiste is dat de 3D kaarten een waarschijnlijkheidvoorstelling moeten bevatten. *Bin picking* opstellingen maken metingen van de omgeving m.b.v. externe sensoren. De metingen zijn niet altijd correct omdat er altijd onzekerheid is. De onzekerheid op de meting komt vaak door reflectie op materialen of bewegende delen. Hier moet rekening mee gehouden worden bij het bouwen van de kaart. Een verbetering van de metingen kan gebeuren door meerdere sensormetingen samen te verwerken.
2. De tweede vereiste is dat bij het modelleren van nieuwe gebieden de robot de onbekende gebieden moet vermijden. Het model moet dus zowel de gekende vrije omgeving, de bezette omgeving, als de onbekende omgeving in beeld brengen. De robot mag enkel een botsingsvrij pad zoeken door de gekende omgeving.
3. De laatste vereiste is dat de kaart snel genoeg moet kunnen geraadpleegd worden en niet te groot is in geheugenverbruik. De kaart is de centrale component van elk autonoom systeem.

Veel gebruikte methodes voor het voorstellen van de omgeving zijn puntenwolken, hoogtekaarten, multi-level oppervlaktekaarten en de volumetrische manier. Geen van deze methodes voldoet aan de bovengestelde eisen. Octomap kan wel aan alle eisen voldoen.

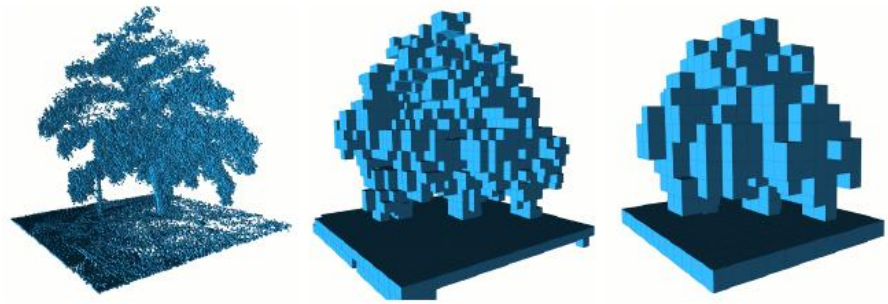
2.3.3 Voortstelling van de ruimte door octrees

Een octree is een manier om volume voor te stellen in een 3D ruimte. Elke node in een octree stelt de ruimte voor in een kubisch volume, een voxel genoemd. Zo een voxel wordt steeds verder onderverdeeld in 8 subvolumes tot een bepaalde voxelgrootte is bereikt. Figuur 4 toont aan hoe dit precies werkt. De voxelgrootte bepaalt ook de resolutie van de octree. Figuur 5 geeft verschillende resoluties van een kaart weer. De bezette ruimte wordt gevonden door het

eindpunt van een afstandssensor te nemen en deze voxel te initialiseren. De vrije ruimte is alle ruimte tussen de meting en het eindpunt en wordt als vrij geinitialiseerd. Alle niet geinitialiseerde voxels zijn onbekende ruimte. Als verschillende voxels dezelfde waarde bevatten kunnen ze samengenomen worden om zo een compactere octree te krijgen, dit heet pruning.



Figuur 4: Voorbeeld van de octree structuur. [3]



Figuur 5: Verschillende resoluties van een octomap. [3]

De grootste problemen bij het voorstellen van de omgeving is dat het geheugengebruik te hoog is, er geen verschil kan gemaakt worden tussen vrije en niet vrije omgeving, en er geen goede volumetrische voorstelling bereikt wordt. Octrees stellen het initialiseren van de kaart uit tot er metingen worden toegevoegd. Zo moet de omgevingsgrootte niet op voorhand gekend zijn. De kaart bevat op deze manier ook alleen de gemeten omgeving waardoor het geheugengebruik kleiner is. De nodes van de octree kunnen ook gemakkelijk verkleind worden zodat verschillende resoluties haalbaar zijn.

2.3.4 Compressie van de octomap

Het berekenen van het vrij zijn of niet van een voxel gebeurt door een formule waarmee een waarschijnlijkheid berekend wordt. Indien de waarschijnlijkheid boven een grenswaarde stijgt zal de voxel een toestand toegewezen krijgen. Door verschillende metingen te doen zal een onstabiele voxel een toestand toegewezen krijgen. Als nu alle kinderen van een node dezelfde toestand hebben kunnen ze gepruned worden. Zorgt een nieuwe meting voor een nieuwe toestand dan zullen ze weer gescheiden worden. Een compressie tot 44% is zo mogelijk [3].

2.3.5 Uitbreiding van de octomap

Het is mogelijk om de octree nodes meer informatie mee te geven zoals bijvoorbeeld kleur of temperatuur. Hierbij worden best verschillende metingen samengenomen om zo correcte informatie te krijgen. Dit creëert een visualisatie voor de gebruiker en staat kleurgerelateerde

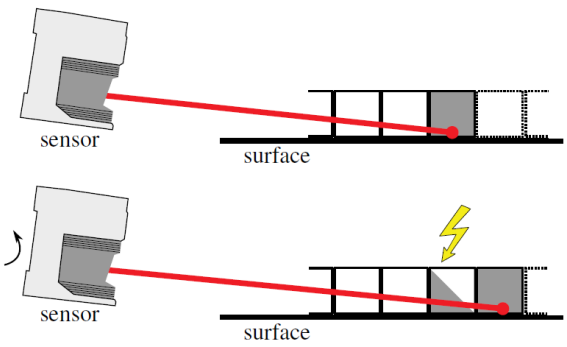
classificatie van de omgeving toe. Een toepassing is het creëren van hoge resolutie oppervlakte meshes voor gebruik in andere software.

Wat een beter beeld van de omgeving kan geven is het gebruik van subkaarten. Via deze kaarten kan er bijvoorbeeld een onderscheid gemaakt worden tussen de muren, de tafel en de objecten op de tafel. Aan deze methode zijn enkele voordelen verbonden. Ten eerste kan er voor iedere submap een aparte resolutie toegepast worden. Ten tweede kan elke submap apart behandeld of verplaatst worden terwijl de rest van de kaart hetzelfde blijft. Ten laatste is het ook mogelijk om afhankelijkheden in te stellen tussen submappen. Als bijvoorbeeld een tafel beweegt zullen de objecten op de tafel mee verschuiven.

2.3.6 Besluit

Een octomap kan gemaakt worden met bijna iedere soort sensor. Het is mogelijk om goede kaarten te bouwen door de waarschijnlijkheidsparameter te veranderen. Binnen Binvox, het programma waarmee in de deze masterproef de puntenwolk wordt omgezet naar een octomap, is het niet mogelijk om de waarschijnlijkheidsparameter aan te passen. Afhankelijk van de sensor kan een andere waarde een beter resultaat geven. Een nadeel is dat als het vertrouwen in de kaart moet toenemen het geheugengebruik vergroot.

Het gebruik van een laserscanner brengt een probleem met zich mee. Bij het afscannen van een vlak oppervlak, bv de grond, onder een kleine hoek zullen de voxels eerst de toestand bezet krijgen en daarna geüpdate worden naar vrij. Dit gebeurt omdat de laser nu door de laatst gescande voxel gaat en deze terug als vrij zal markeren. Deze ongewenste updates zorgen voor gaten in het model. Dit kan opgelost worden door de toestand van iedere voxel slechts één keer per cyclus te laten veranderen. Figuur 6 toont wat met dit probleem bedoeld wordt. Binnen de *bin picking* opstelling op ACRO treedt dit probleem niet op omdat de hoek van de camera t.o.v. de laser op een vaste positie staat. De openingshoek is hier voldoende groot om dit probleem te voorkomen.



Figuur 6: Probleem bij het scannen van een vlakke oppervlakte onder een kleine hoek. [3]

De nauwkeurigheid van de kaart is gecontroleerd door de verschillende gemeten kaarten met elkaar te vergelijken. Een nauwkeurigheid van 97% is haalbaar via octomapping. De resterende foute metingen zijn het gevolg van *sensor noise*, het discretiseringseffect en slechte uitlijning. Octomapping is dus een goed bruikbare methode in verschillende toepassingen.

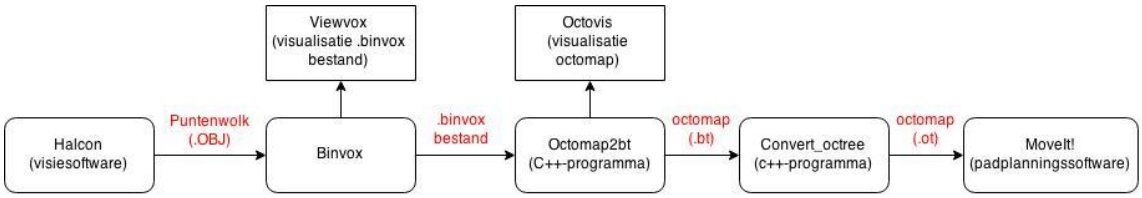
3 Communicatie tussen visie- en padplanningsoftware

3.1 Inleiding

Bin picking toepassingen vertrekken vaak vanuit een visiesysteem dat (1) de te grijpen objecten herkent en localiseert en (2) een 3D puntenwolk genereert die gebruikt kan worden voor botsingsvermijding van de robotarm, zie **Fout! Verwijzingsbron niet gevonden.**. In een volgende stap gebruikt een padplanner deze data, de grijpposities van gedetecteerde objecten en de 3D puntenwolk, om een botsingsvrij traject te berekenen voor de industriële robotarm. Voor de opstelling in ACRO gebeurt dit net zo, en verwacht de padplanner uit MoveIt! een voorstelling van de omgeving in de vorm van een octomap. Hiertoe is dus een omvorming nodig van de 3D puntenwolk naar een octomap. Dit hoofdstuk bepaalt eerst hoe een octomap wordt opgebouwd en vorvolgens hoe deze tot bij de planner geraakt. Deze methode wordt daarna ook geëvalueerd.

In deze masterproef is gekozen om te werken met een octomap volgens de octomapping methode van A. Hornungen et al..[3] Deze octomap voorstellingswijze is gemakkelijker te verwerken in de padplanning dan een puntenwolk. In ROS zijn verschillende bibliotheken aanwezig voor de omzetting van een puntenwolk naar een octomap.In figuur 7 is een overzicht te zien van de omzetting van een puntenwolk tot een octomap.[8][9]

In sectie 3.2 wordt de omvorming van de puntenwolk naar een octomap besproken. Daaropvolgend wordt in sectie 3.3 verklaard hoe de grijpposities vanuit Halcon worden doorgestuurd en gebruikt. Ten slotte wordt een besluit gevormd in sectie 3.4.



Figuur 7: Overzicht van de omzetting van een puntenwolk naar een octomap.

3.2 Omvorming van de puntenwolk naar een octomap

3.2.1 Gebruik van Binvox voor de omvorming

Een octomap is een 3D voorstellingswijze van de omgeving in de vorm van kubussen. Deze voorstellingswijze is ondersteund door MoveIt! en leek een goede mogelijkheid voor de bin picking opstelling. Er zijn verschillende bibliotheken beschikbaar zoals PCL en Octomap in ROS. Voor de omvorming van een puntenwolk naar een octomap is echter gekozen voor de software Binvox.[10] Binvox bevat namelijk zijn eigen visualisatieprogramma “Viewvox”. Dit leek op het eerste zicht veelbelovend aangezien hiervoor nog geen ROS nodig was en snel een resultaat kon bekomen worden.

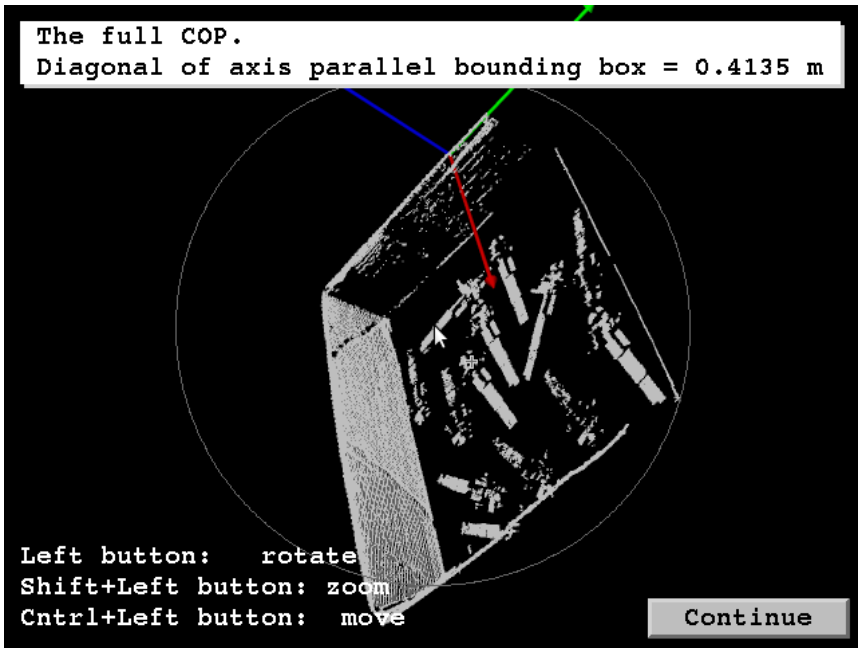
Binvox leest een 3D model bestand in, zoals weergegeven in figuur 8 en converteert deze naar een 3D voxel rooster van het type .binvox, HIPS, MIRA, VTK, minecraft schematic formaat, Gmsh .msh format en nrrd. Figuur 10 toont de visualisatie van een .binvox file. De mogelijke inputformaten zijn Wavefront OBJ, Geomview OFF, Autocad DXF, PLY en STL. Halcon voorziet de puntenwolk die omgevormd wordt. De mogelijke bestandstypes zijn dezelfde als de

inputformaten van Binvox. Er is voor het uitvoeren van de tests voor het .OBJ type gekozen omdat dit het beste ondersteund wordt.

Het .OBJ bestand bevat standaard niet de polygonen die Binvox nodig heeft om de voxels te bepalen. De polygonen worden door Halcon gegenereerd nadat het beeld van de camera is ingeladen. Hiervoor zijn twee methodes mogelijk binnen Halcon:

1. De eerste is de "greedy" methode waar een algoritme het oppervlak construeert dat door de punten van de puntenwolk gaat. Deze methode geeft het beste resultaat en is dus gebruikt voor de omzetting;
2. De tweede methode is de "implicit" methode dat een waterdicht oppervlak creëert over de punten van de puntenwolk. Dit geeft geen bruikbaar resultaat zoals zichtbaar is in figuren 10 en 11.

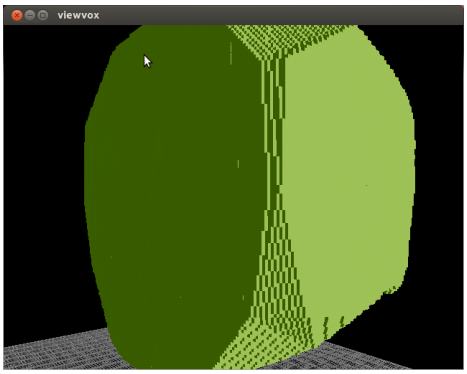
Een vergelijking van de 2 methodes is te zien in figuur 9.



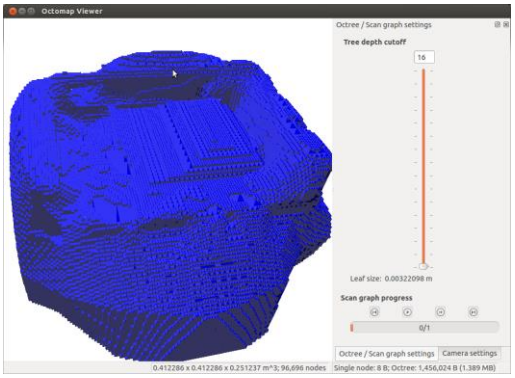
Figuur 8: De .OBJ puntenwolk afkomstig uit Halcon die voor de simulaties is gebruikt.

Property	Greedy triangulation	Implicit triangulation
required data:	3D points with 3D normals	3D points with 3D normals, the normals must point consistently inwards
resulting surface:	open, triangulation of the input points	closed (water-tight), approximation of the input points
resulting point data:	the input point data is preserved	new point data is generated
noise handling:	moderate point and normal noise handled properly	point and normal noise handled implicitly; moderate and high noise levels are accepted
triangulation resolution:	explicit, controlled by surface neighborhood parameters	implicit, controlled by octree depth and minimal number of point samples per node
time complexity:	O(N k log(N))	O(N D^3)
memory complexity:	O(N k), with neigh. prefetching O(N), without neigh. prefetching	O(D^3) *where:* N: *number of points* k: *size of the neighborhood* D: *depth of the octree*

Figuur 9: Vergelijking van triangulatie methodes uit Halcon voor het genereren van polygoonoppervlakken op basis van puntenwolken.[11]



Figuur 10: Visualisatie van het .binvox bestand, gemaakt m.b.v. de implicit methode, in Viewvox.



Figuur 11: Visualisatie van de octomap gemaakt m.b.v. de implicit methode. De *Leaf size is 3,2 mm.*

Nadat de puntenwolk uit Halcon is omgezet naar een .binvox bestand moet deze nog omgezet worden naar een octomap. Uit de tests, die in sectie 3.2.2 worden besproken, blijkt echter dat de .binvox file helemaal niet op de originele puntenwolk lijkt. Er is toen besloten om toch te controleren of het omvormen van het .binvox bestand naar een octomap een goed resultaat geeft.

Het C++ programma “octomap2bt”, dat standaard in de Octomap bibliotheek voor ROS zit, zet het .binvox bestand om naar een .bt bestand, een extensie voor een octomap. Er zijn verschillende tests uitgevoerd om de beste parameterinstellingen voor de conversie te vinden.

De Octomap bibliotheek in ROS bevat een C++-programma “octomap2bt” dat een .binvox bestand omzet naar een octomap van het type .bt. De nieuwere versies van MoveIt! maken gebruik van het .ot type om een octomap voor te stellen. Een .ot type staat voor “octree” en bevat een volledige bezettingsgraadkaart in octomap formaat. De knopen in de octree slaan de volledige bezettingsgraadwaarschijnlijkheid op. Het .bt type staat voor binary tree. Dit type slaat enkel de maximum waarschijnlijkheid op. Iedere node is ofwel vrij ofwel bezet. Meer informatie rond de octomap structuur is terug te vinden in sectie 2.3. De Octomap bibliotheek bevat ook een visualisatieprogramma voor octomappen, Octovis genaamd. M.b.v. dit programma is het resultaat van de omzetting naar een octomap gecontroleerd. De tests met hun resultaten staan beschreven in sectie 3.2.2.

3.2.2 Simulaties m.b.v. Viewvox en Octoviz

De omzetting van de puntenwolk wordt visueel gecontroleerd met behulp van twee visualisatieprogramma's. Enerzijds viewvox om het .binvox bestand te bekijken en anderzijds octovis om de octomap te bekijken.

Binvox biedt parameters aan om de omzetting aan te passen. Deze worden mee ingegeven in de terminal bij het uitvoeren van Binvox. De gebruikte parameters voor deze tests zijn:

Voxelroostergrootte (-d): standaard 256, maximum 1024 kubussen. Er is geen maximum als de parameter –e wordt gebruikt. De langste zijde van het ingeladen bestand zal uit het gekozen aantal kubussen bestaan. De andere zijdes bestaan uit even grote kubussen. De grootte van een kubus hangt dus af van de grootte van het ingeladen bestand en de gekozen resolutie.

Externe buffer (-pb): de omzetting wordt niet op het scherm getoond, dit zorgt ervoor dat de omzetting sneller kan gebeuren.

Z-buffering gebaseerde carving (-c): z-buffer of dieptebuffering wordt gebruikt in grafische toepassingen om een onderscheid te maken tussen de objecten die gerenderd moeten worden. Deze buffer is meestal gerangschikt volgens een x-y veld met een element voor iedere pixel. Als een ander element in dezelfde pixel moet worden gerenderd worden de diepteafstanden vergeleken en de kortste diepte wordt gebruikt als pixel.

Z-buffering pariteitsstemming (-v): enkel *z-buffer based parity voting* methode.

Begrenzing instellen (-fit): ken enkel een waarde bezet of vrij toe aan de voxels in de voxel "bounding box". De doos kan ingesteld worden met de parameter –bb. Is er geen "bounding box" gespecifieerd dan zal er zelf een gemaakt worden die de hele puntenwolk omsluit. De omzetting kan hierdoor sneller gebeuren.

Exacte voxelizatie (-e): elke voxel die voor een deel bezet is wordt als bezet gemarkeerd.

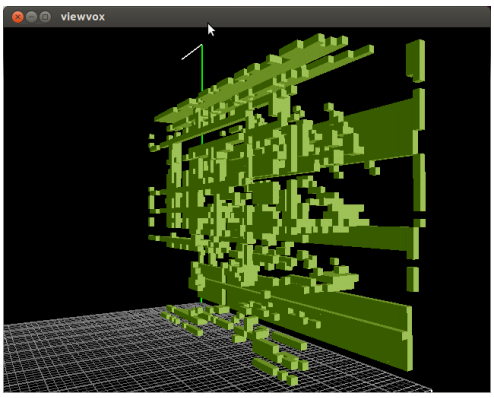
Met onderstaande simulaties is gecontroleerd of Binvox een bruikbare manier is om een puntenwolk om te zetten naar een octomap. Voor de verschillende simulaties zijn het .binvox bestand en de octomap gevisualiseerd. Dit is gedaan om de parameterinvloed te controleren en een visueel beeld van de bestanden te krijgen. De bruikbaarheid van het geproduceerde octomapbestand wordt op die manier ook voor een deel geverifieerd. De input voor alle simulaties is de puntenwolk uit figuur 8.

Simulatie 1: Figuren 12 en 13 tonen respectievelijk de visualisatie van het .binvox bestand in Viewvox en de octomap in octovis. De volgende instellingen werden gebruikt:

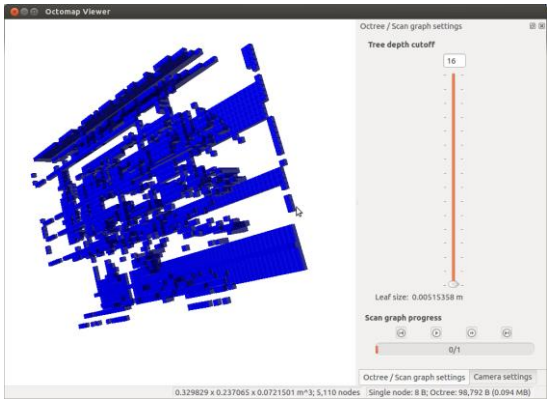
- resolutie: 64,

- parameters: -pb,

- omzettingstijd van .OBJ naar octomap: 14.07 sec.



Figuur 12: Gevisualiseerde .binvox file in Viewvox van simulatie 1



Figuur 13: Gevisualiseerde octomap in Octovis van simulatie 1.

Conclusie:

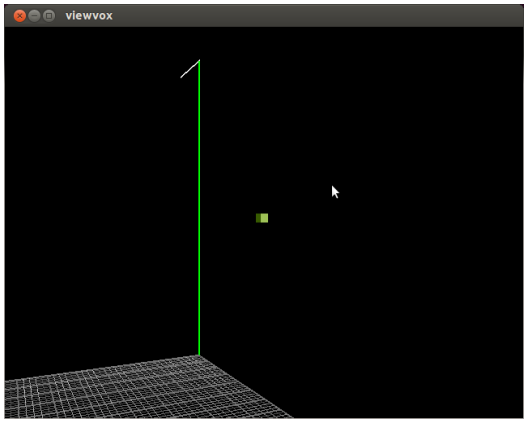
De omzettingstijd voor deze simulatie is veel te lang om in de praktijk bruikbaar te zijn. Er is een ruwe vorm van de puntenwolk aanwezig, zowel in het .binvox bestand als in de octomap. Dit is echter helemaal niet correct genoeg om mee verder te werken. Ook zitten er volzette *voxels* in de octomap die in de puntenwolk vrij zijn. Een bruikbare octomap maken met deze parameters is niet mogelijk.

Simulatie 2: Figuren 14 en 15 tonen respectievelijk de visualisatie van het .binvox bestand in Viewvox en de octomap in octovis. De volgende instellingen werden gebruikt:

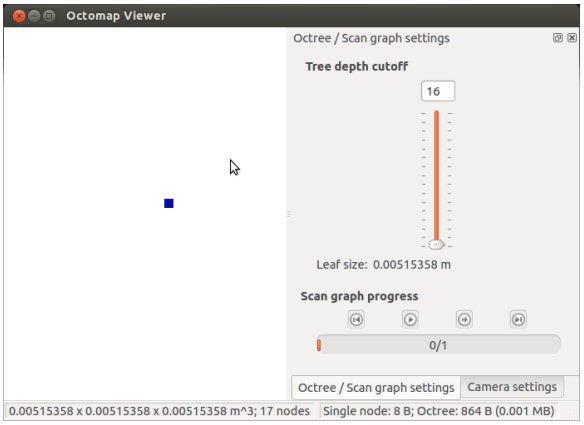
- Resolutie: 64,

- parameters: -pb, -c,

- omzettingstijd van .OBJ naar octomap: 3 sec.



Figuur 14: Gevisualiseerde .binvox file in Viewvox van simulatie 2.



Figuur 15: Gevisualiseerde octomap in Octovis van simulatie 2.

Conclusie:

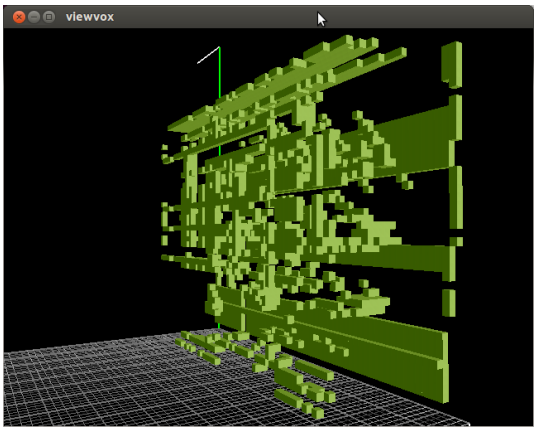
De omzettingstijd voor deze simulatie is al beter dan in simulatie 1. Ook hier is duidelijk zichtbaar dat de octomap totaal niet overeenkomt met de puntenwolk. Een bruikbare octomap maken met deze parameters is niet mogelijk.

Simulatie 3: Figuren 16 en 17 tonen respectievelijk de visualisatie van het .binvox bestand in Viewvox en de octomap in octovis. De volgende instellingen werden gebruikt:

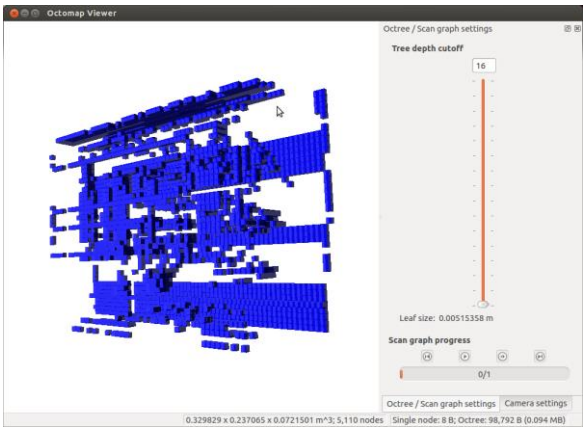
- resolutie: 64,

- parameters: -pb, -v,

- omzettingstijd van .OBJ naar octomap: 13.83 sec.



Figuur 16: Gevisualiseerde .binvox file in Viewvox van simulatie 3.



Figuur 17: Gevisualiseerde octomap in Octovis van simulatie 3.

Conclusie:

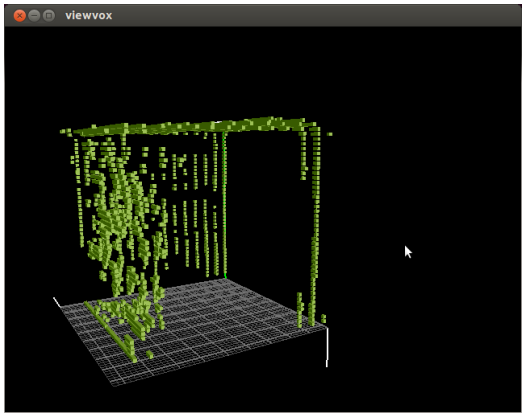
De omzettingstijd voor deze simulatie is even slecht als in simulatie 1. Het .binvox bestand en de octomap komen niet overeen met de puntenwolk en kunnen dus niet gebruikt worden. Een bruikbare octomap maken met deze parameters is niet mogelijk.

Simulatie 4: Figuren 18 en 19 tonen respectievelijk de visualisatie van het .binvox bestand in Viewvox en de octomap in octovis. De volgende instellingen werden gebruikt:

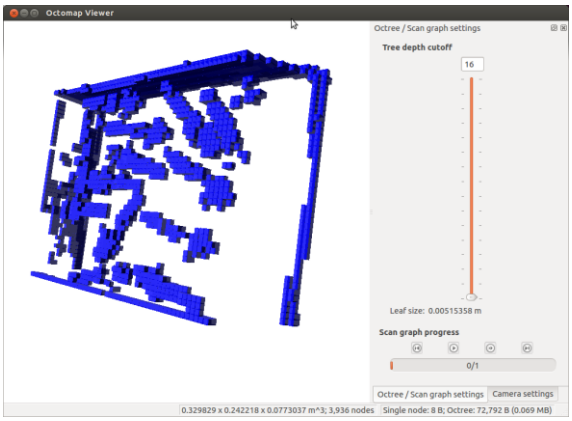
- resolutie: 64,

- parameters: -fit, -e,

- omzettingstijd van .OBJ naar octomap: 2 sec.



Figuur 18: Gevisualiseerde .binvox file in Viewvox van simulatie 4.



Figuur 19: Gevisualiseerde octomap in Octovis van simulatie 4.

Conclusie:

De omzettingstijd voor deze simulatie is nog steeds redelijk hoog. Het .binvox bestand komt totaal niet overeen met de puntenwolk en kan dus niet als referentie gebruikt worden. De octomap toont een zeer goede gelijkenis met de puntwolk. De *leaf size* van de octomap is 5,15mm.

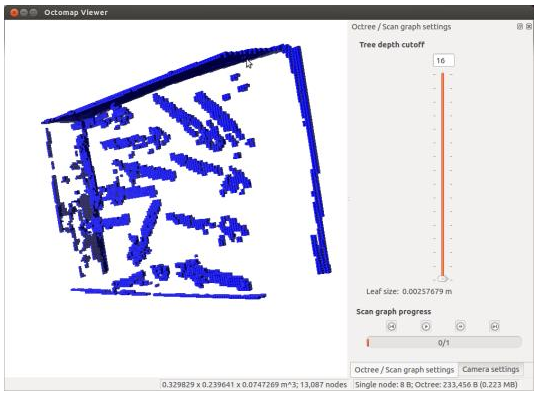
Uit de eerste 4 simulaties blijkt dat door gebruik te maken van de parameters –fit en –e een goed resultaat kan behaald worden. In de volgende simulaties is gekeken wat de invloed is van de resolutie op de verwerkingstijd.

Simulatie 5: Figuur 20 toont de visualisatie van de octomap in octovis. De volgende instellingen werden gebruikt:

- resolutie: 128,

- parameters: -fit, -e,

- omzettingstijd van .OBJ naar octomap: 2 sec.



Figuur 20: Gevisualiseerde octomap in Octovis van simulatie 5.

Conclusie:

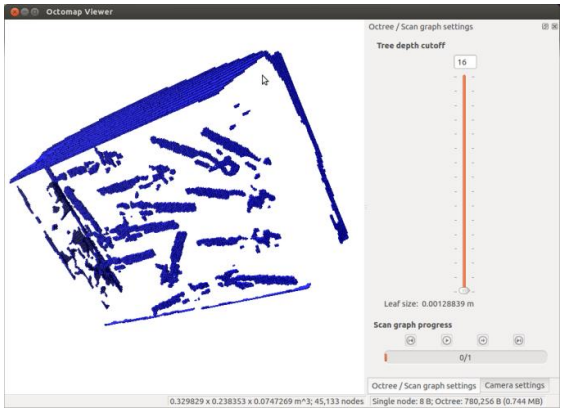
De omzettingstijd neemt niet toe met een stijgende resolutie. De *leaf size* van de octomap is 2,58mm.

Simulatie 6: Figuur 21 toont de visualisatie van de octomap in octovis. De volgende instellingen werden gebruikt:

- resolutie: 256,

- parameters: -fit, -e,

- omzettingstijd van .OBJ naar octomap: 2 sec.



Figuur 21: Gevisualiseerde octomap in Octovis van simulatie 6

Conclusie:

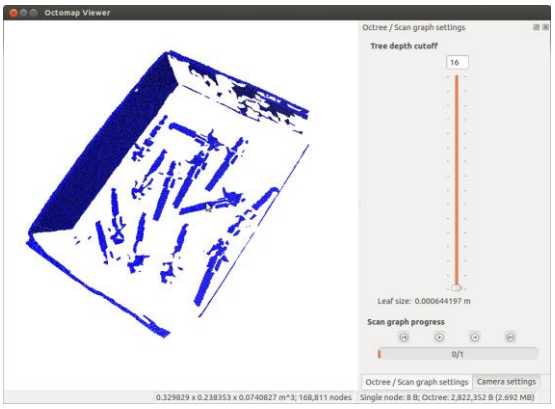
De omzettingstijd neemt niet toe met een stijgende resolutie. De *leaf size* van de octomap is 1,29mm.

Simulatie 7: Figuur 22 toont de visualisatie van de octomap in octovis. De volgende instellingen werden gebruikt:

- resolutie: 512,

- parameters: -fit, -e,

- omzettingstijd van .OBJ naar octomap: 2 sec.



Figuur 22: Gevisualiseerde octomap in Octovis van simulatie 7.

Conclusie:

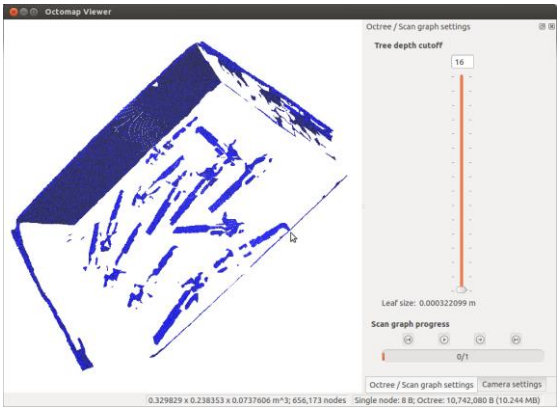
De omzettingstijd neemt niet toe met een stijgende resolutie. De *leaf size* van de octomap is 0,64mm.

Simulatie 8: Figuur 23 toont de visualisatie van de octomap in octovis. De volgende instellingen werden gebruikt:

- resolutie: 1024,

- parameters: -fit, -e,

- omzettingstijd van .OBJ naar octomap: 2 sec.



Figuur 23: Gevisualiseerde octomap in Octovis van simulatie 8.

Conclusie:

De omzettingstijd neemt niet toe met een stijgende resolutie. De *leaf size* van de octomap is 0,32mm.

3.2.3 Besluit

Binvox is in staat, door gebruik te maken van een .OBJ file, een octomap met een "leaf size" van <1,2 mm te maken, zoals te zien is in simulaties 6,7 en 8. Voor de *bin picking* toepassing die uitgevoerd wordt is dit voldoende nauwkeurig.

Het verhogen van de resolutie heeft geen invloed op de omzettingstijd maar het laden van de octomap in MoveIt! zal langer duren met een hogere resolutie

Voor een industriële binpicking installatie is het gebruik van Binvox geen goede methode. Hoewel de resultaten aanvaardbaar zijn, is de omzetting te traag. Voor het gebruik van octomappen in ROS moet er nog verder onderzoek gebeuren. De PCL(Point Cloud Library) en octomap bibliotheken voor ROS zijn mogelijkheden.

Voor de keuze van het bestandstype van de puntenwolk dat aan Binvox wordt aangeboden is geen vergelijkende studie gemaakt. Hier kan in toekomstig werk nog naar gekeken worden.

Aangezien deze masterproef zich meer toespitst op de padplanning is besloten om de octomap met een resolutie van 256 voxels te gebruiken in MoveIt! voor botsingsdetectie.

3.3 Doorsturen van de grijpposities

In deze masterproef is er weinig tijd gespendeerd aan het visiegedeelte aangezien dit al klaar was. De grijpposities voor de verschillende voorwerpen worden berekend door een 3D CAD model te passen op de puntenwolk. Deze posities worden dan omgevormd naar een combinatie van een punt (x,y,z) voor translatie en een quaternion (w,x,y,z) voor rotatie.

De posities worden opgeslagen in een tekstbestand in volgorde van grootste grijpkans. Daarna kan er op basis van deze punten een botsingsvrij pad gepland worden. Het wegschrijven van waardes naar een tekstbestand en ze dan weer opvragen neemt meer tijd in beslag dan ze meteen door te sturen. De communicatie hiervoor kan op dezelfde manier gebeuren als de communicatie tussen padplanningsoftware en de robot, zoals beschreven in sectie 5.2. De implementatie hiervan kan snel gebeuren aangezien de code hiervoor grotendeels aanwezig is.

3.4 Besluit

Dit hoofdstuk behandelde de omzetting van het robotomgevingsmodel in de vorm van een 3D puntenwolk naar een octomapformaat dat bruikbaar is voor de MoveIt! padplanningssoftware. Hiertoe werd de software Binvox gebruikt, waarbij geëvalueerd werd welke Binvox parameters optimaal zijn en hoeveel tijd nodig is voor de omzetting naar een octomap.

Dit leidde tot volgende besluiten:

- Binvox is in staat, door gebruik te maken van een .OBJ file, een octomap met een "leaf size" van <1,2 mm te maken, zoals te zien is in simulaties 6,7 en 8. Voor de *bin picking* toepassing die uitgevoerd wordt is dit voldoende nauwkeurig;
- De omzettingstijd is ongeveer even lang voor andere resoluties, en bedraagt 2 s, wat te traag is voor industriële bin picking toepassingen.
- Voor de keuze van het bestandstype van de puntenwolk dat aan Binvox wordt aangeboden is geen vergelijkende studie gemaakt.

Voor een industriële binpicking installatie is het gebruik van Binvox geen goede methode. Hoewel de resultaten aanvaardbaar zijn, is de omzetting te traag en werkt de huidige software met bestanden (tussen de visie- en padplanningssoftware worden bestanden uitgewisseld i.p.v. data rechtstreeks door te sturen bijv. via sockets). Naar toekomstig onderzoek toe, vormen de PCL (Point Cloud Library) en octomapbibliotheken voor ROS alternatieven die mogelijk een snellere conversie toelaten.

Aangezien deze masterproef zich vooral toespitst op de padplanning is besloten om de octomap met een resolutie van 256 voxels te gebruiken in MoveIt! voor botsingsdetectie.

4 Padplanning in simulatie

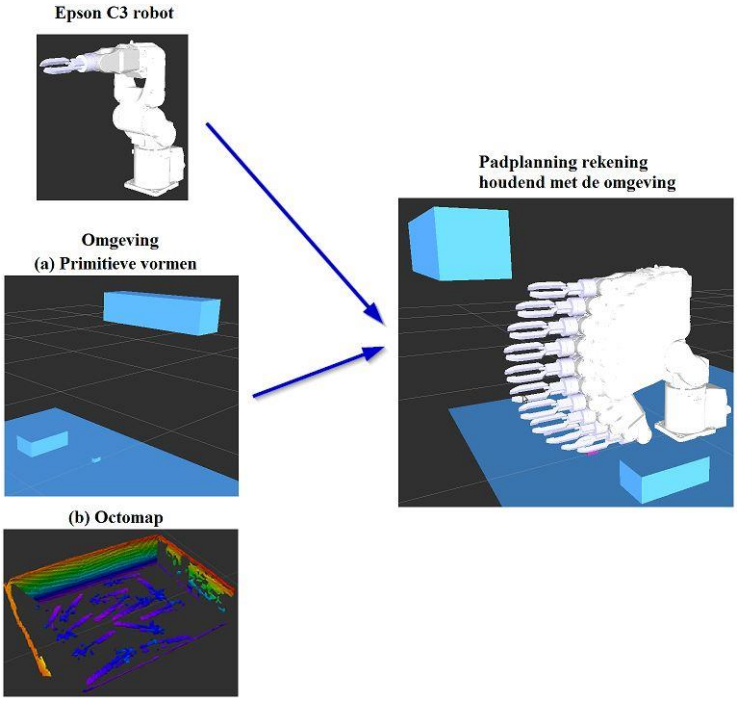
4.1 Inleiding

Een belangrijk deel van een *rondom bin picking* toepassing is de berekening van een botsingsvrij pad. De robot dient dit pad daarna ook zo getrouw mogelijk te volgen. In deze masterproef is gekozen voor de MoveIt! bibliotheek voor de berekening van de botsingsvrije paden.

De berekening van zo een pad kan op verschillende manieren gebeuren, afhankelijk van het gekozen padplanningsalgoritme. De OMPL (Open Motion Planning Library) bibliotheek bevat een grote keuze aan padplanningsalgoritmes. De volledige lijst is terug te vinden in [11]. Er is in een vorig eindwerk [4] reeds een studie gedaan rond de keuze van een padplanningsalgoritme. In dat onderzoek kwam het RRT-connect algoritme als het meest betrouwbare en het snelste naar voren. Het is dit algoritme dat voor de simulaties in deze masterproef wordt gebruikt.

ROS bevat de visualisatiesoftware rviz. In deze software kan de robot, de omgeving en een berekend pad voorgesteld worden door gebruik te maken van de “motion planner” plugin. Dit staat weergegeven in figuur 24. Deze plugin maakt gebruik van MoveIt!. In deze masterproef is gekozen om eerst de botsingsvrije paden in rviz te simuleren, alvorens dit uit te voeren op de Epson C3 robot. Het gebruik van rviz is dus geen vereiste om aan *bin picking* te kunnen doen m.b.v. ROS en MoveIt!, maar dient enkel als visuele controle.

In sectie 4.2 wordt eerst de visualisatie van de robot besproken. Vervolgens wordt in sectie 4.3 de visualisatie van de omgeving getoond. De visualisatie en uitvoering van een berekend botsingsvrij pad wordt in sectie 4.4 besproken. De doelen van de simulaties zijn het bepalen van de haalbaarheid van *random bin picking*, controle van de padplannings- en cyclustijd en een controle voorzien op botsingen. Sectie 4.5 vormt een besluit over de padplanning in simulatie.



Figuur 24: Overzicht van de elementen die gevisualiseerd moeten worden (links) om rekening met elkaar te houden en padplanning (rechts) mogelijk te maken. De omgeving kan door primitieve figuren (a) of een octomap (b) worden voorgesteld.

4.2 Visualisatie van de robot

Het eerste wat moet gebeuren is de robot visualiseren in rviz. Een robot kan worden gevisualiseerd door gebruik te maken van een srdf (Semantic Robot Description File*)* bestand. Dit bestand bevat vooreerst de informatie uit een urdf bestand. Een urdf bestand bevat de informatie van de links, joints en de meshes voor visualisatie alsook voor padplanning zelf. In figuur 25 is de inhoud van een urdf bestand zichtbaar. Het srdf bestand bevat ook nog informatie over de robot zoals groepen van links en joints, de eindeffector en een lijst van links die nooit met elkaar in botsing treden.

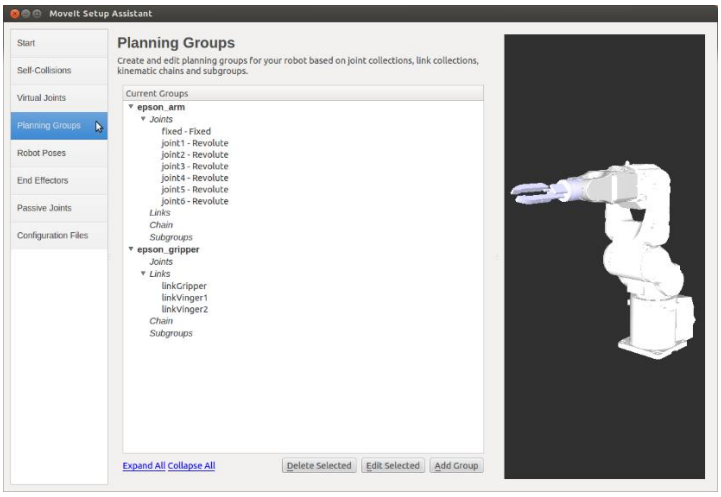
```
<link
  name="link1">
  <visual>
    <origin
      xyz="0 0 -0.1965"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://epsonURDF_gripper/meshes_new/link1.stl" scale=".001 .001 .001"/>
    </geometry>
    <material
      name="">
      <color
        rgba="1 1 1 1" />
    </material>
  </visual>
  <collision>
    <origin
      xyz="0 0 -0.1965"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://epsonURDF_gripper/meshes_new/link1.stl" scale=".001 .001 .001"/>
    </geometry>
  </collision>
</link>

<joint
  name="joint1"
  type="revolute">
  <origin
    xyz="0 0 0.1965"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="link1" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-2.9671"
    upper="2.9671"
    effort="0"
    velocity="0" />
</joint>
```

Figuur 25: Een stuk informatie uit een urdf bestand. Hierin wordt een link en een joint gespecifieerd.

De informatie rond groepen is vooral belangrijk voor het padplannen. In de code voor het berekenen van een pad staat gespecifieerd met welke groep MoveIt! een pad zoekt. Het is dus belangrijk dit srdf bestand correct te maken. Indien dit niet gebeurt, zullen er problemen optreden bij het plannen.

4.2.1 Maken van de SRDF-file

De *MoveIt! Setup Assistant* [12] is een grafische user interface voor het configureren van een robot voor het gebruik met MoveIt!. Deze software zit standaard in de MoveIt! bibliotheek en wordt gebruikt om het srdf bestand aan te maken. In figuur 26 is de *MoveIT! Setup Assistant* te zien. Daarin wordt weergegeven welke planning groepen er zijn. Bij het berekenen van een pad worden op deze de berekeningen uitgevoerd. De eerste groep bestaat uit de 6 joints van de robotarm en de tweede groep uit de links van de eindeffector. De procedure om het SRDF bestand aan te maken staat beschreven op de MoveIt! website.



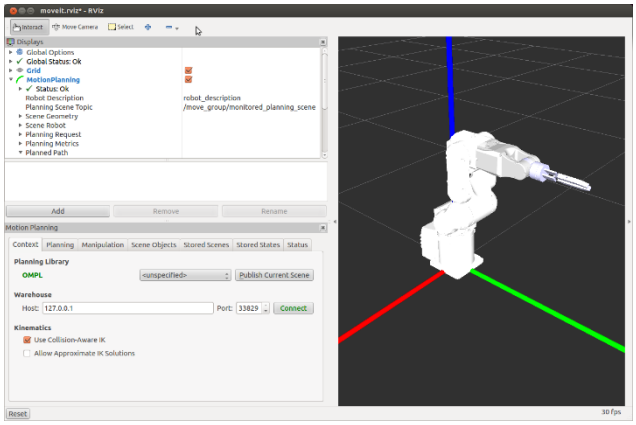
Figuur 26: De MoveIt! setup assistant pagina waar de planninggroepen worden weergegeven.

4.2.2 De robot in rviz

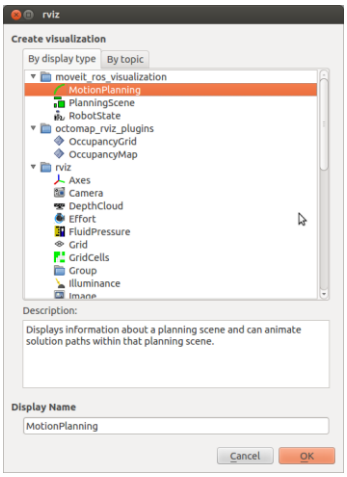
Zodra het srdf bestand is aangemaakt kan de robot gevisualiseerd worden in rviz. Dit is mogelijk door de "demo.launch" file te starten m.b.v het roslaunch commando in ROS. Rviz haalt dan alle informatie op uit de configuratiefiles van het srdf bestand. Indien er een fout in het srdf bestand zit zal rviz dit melden. In dit geval moet de setup opnieuw doorlopen worden. Een controle van het urdf bestand is ook wenselijk.

Om de robot zichtbaar te maken moet de "motion planning" plugin worden toegevoegd. Dit kan gebeuren door op "add" te klikken in de GUI van rviz, zoals zichtbaar is in figuur 27. Figuur 28 toont het scherm dat na het klikken tevoorschijn komt. Dit scherm bevat alle beschikbare plugins.

Vervolgens kan de robot gevisualiseerd worden door in de "motion planning" plugin de parameter "Robot Description" te laten verwijzen naar de srdf file. Om te controleren of de robot goed beweegt kunnen er onder "Planning Request" 2 *interactive markers* gekozen worden. Met deze *markers* kan de start- en doelpositie van de robot gekozen worden. Door het bewegen van de robot is het mogelijk om te zien of alle links juist t.o.v. elkaar geplaatst zijn en bewegen. Wanneer de robot correct wordt gevisualiseerd kan het plannen beginnen.



Figuur 28: De robot gevisualiseerd in rviz.



Figuur 27: De verschillende plugins voor rviz. Dit scherm wordt geopend na het klikken op "add" in rviz.

4.2.3 Besluit

Door het bewegen van de robot m.b.v. *interactive markers* in rviz is visueel gecontroleerd of alle links juist t.o.v. elkaar bewegen. Door het veranderen van de start- en doelpositie van de robot is het mogelijk om snel een padplanningssimulatie in een lege ruimte te doen. Daaruit kan ook het gedrag van de robot gecontroleerd worden. Dit blijkt allemaal in orde te zijn.

4.3 Visualisatie van de omgeving

De volgende stap is het toevoegen van obstakels in de ruimte. Bij het padplannen dient de robot hier rekening mee te houden. In de volgende secties wordt de visualisatie van octomappen en primitieve figuren besproken.

4.3.1 Visualisatie van octomappen

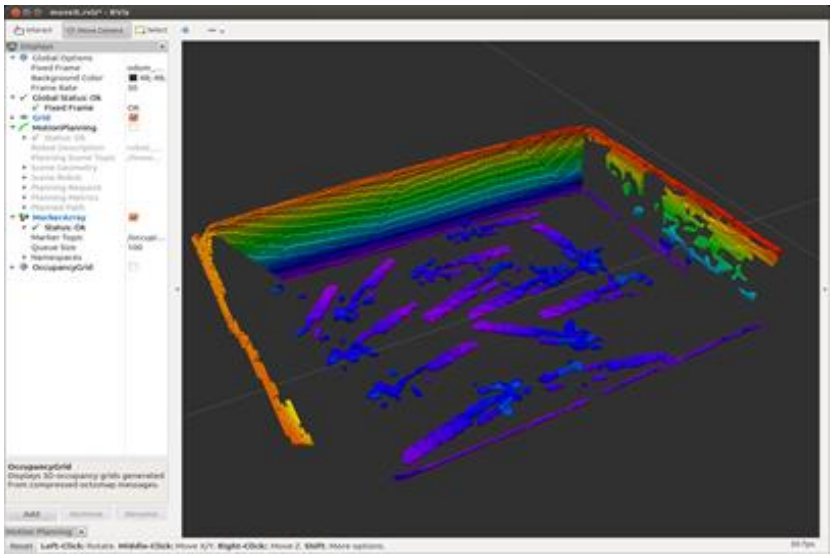
4.3.1.1 Versturing en verwerking van de octomap

De octomap die in hoofdstuk 3 is gemaakt wordt hier gevisualiseerd in rviz. Het is de bedoeling dat de robot deze octomap ziet als een obstakel. De robot moet namelijk de randen van de doos kunnen ontwijken en een object grijpen.

Om het inladen van een octomap mogelijk te maken wordt de octomap bibliotheek [8] gebruikt. Deze bibliotheek bevat functies die nodig zijn om met octomappen binnen ROS te werken.

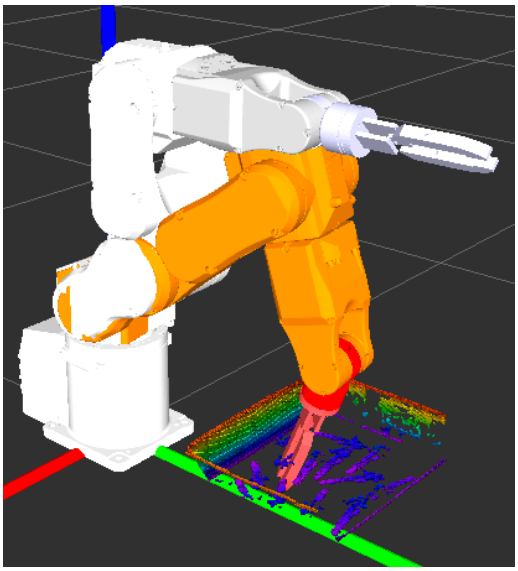
De visualisatie van de octomap gebeurt op de volgende manier. De octomap wordt op het "planning_scene" topic gepubliceerd. Zo een topic is een bus met een naam waarover processen, ook wel nodes genoemd, met elkaar kunnen communiceren binnen ROS. MoveIt! luistert naar dit topic om zo de gepubliceerde data op te vragen.

De data van de octomap zoals bekomen in hoofdstuk 3 is niet direct bruikbaar voor MoveIt!. MoveIt! verwacht een bericht van het type "moveit_msgs::PlanningScene" maar de octomap is van het type "octomap::octree". De data dient dus eerst te worden omgezet naar het juiste type. Zodra een nieuwe octomap wordt aangeboden zorgt een zelf geschreven C++-programma voor de omzetting van de data naar het juiste berichttype en publiceert het bericht op het "planning_scene" topic. Vervolgens zal rviz de octomap visualiseren. Figuur 29 geeft de visualisatie van de octomap in rviz weer.



Figuur 29: Visualisatie van de octomap in rviz.

Nu de octomap gevisualiseerd wordt kan er gekeken worden of de robot dit ook effectief als een obstakel ziet. Hiervoor is de robot bewogen m.b.v. de *interactive markers* tot deze in botsing treedt met de octomap. In figuur 30 is zichtbaar dat de robot met zijn eindeffector botst met de octomap. De kleur van de robot is veranderd van oranje naar rood, wat een botsing illustreert.



Figuur 30: De robot treedt met zijn eindeffector in botsing met de octomap. Dit is zichtbaar omdat de eindeffector rood gekleurd is.

4.3.1.2 Besluit

Er is gebleken dat de robot de octomap als een obstakel ziet. Een probleem dat zich voordeed na het visualiseren van de octomap is dat de robot met zichzelf in botsing treedt in bepaalde willekeurige configuraties. De links die botsen zijn link 5 en link 6 en de grippers van de eindeffector. Door deze botsingen is padplanning niet mogelijk. Voor dit probleem is tijdens de masterproef nog geen oplossing gevonden. Er is dan besloten om verder te werken met primitieve figuren.

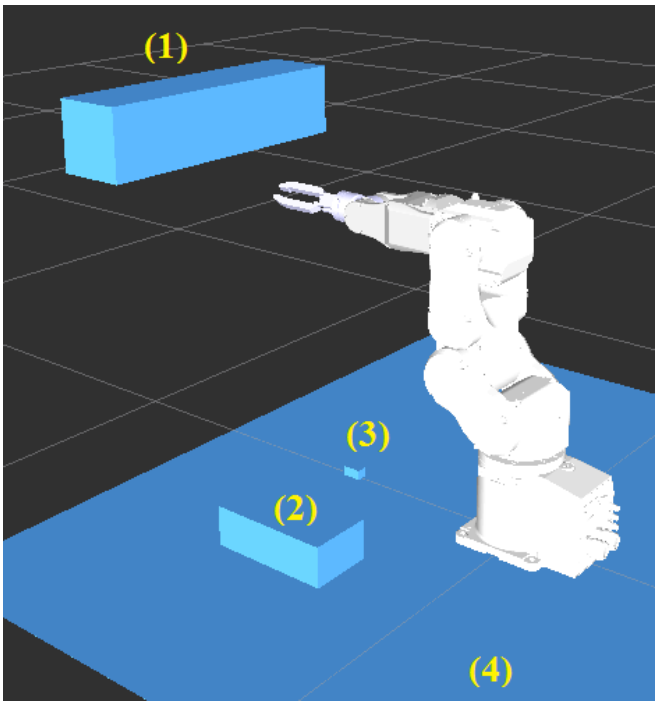
4.3.2 Visualisatie van primitieve vormen

Omdat het gebruik van een octomap vreemde botsingen geeft en padplanning onmogelijk maakt is de masterproef voortgezet met primitieve vormen. Er is gebruik gemaakt van balken om de omgeving voor te stellen. Bollen, cilinders en kegels zijn ook nog mogelijk binnen de primitieve vormen. Er zijn verschillende obstakels nodig om botsingen met de fysieke wereld te vermijden. De ingevoegde obstakels zijn:

- de tafel waar de robot op staat;
- het frame waar de camera en laser op gemonteerd zijn;
- een te grijpen object;
- een obstakel waar een baan rond gepland wordt.

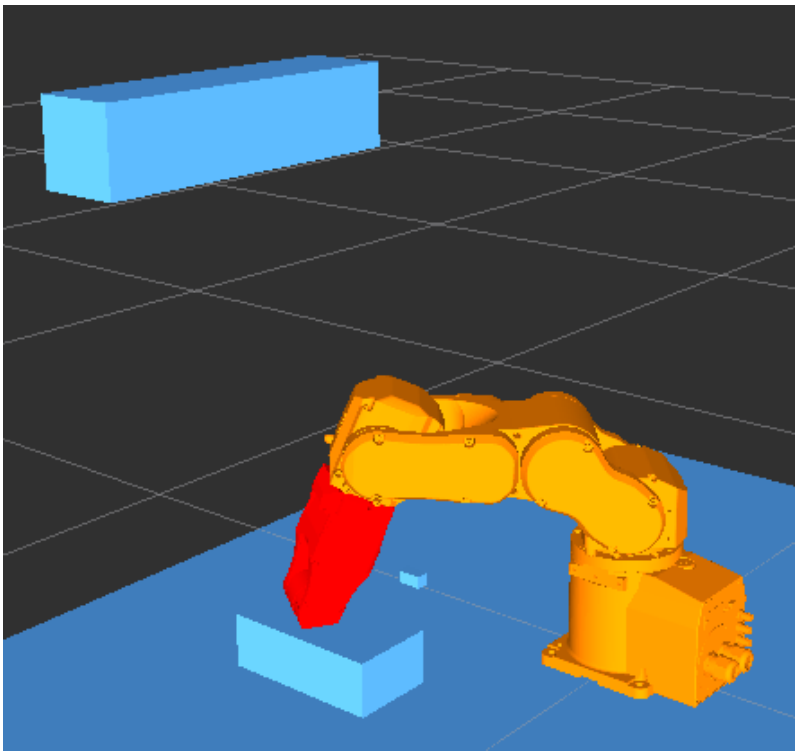
4.3.2.1 Versturing en verwerking van de primitieve vormen

Net zoals bij het inladen van een octomap moet het juiste berichttype aan MoveIt! worden aangeboden. De primitieve vormen zijn van het type “shape_msgs::SolidPrimitive” en worden omgevormd naar “moveit_msgs::CollisionObject”. Ook deze berichten worden op het “planning_scene” topic gepubliceerd. Dit gebeurt allemaal in een C++-programma dat in deze masterproef is geschreven. Het resultaat in rviz is zichtbaar in figuur 31.



Figuur 31: Visualisatie van de primitieve vormen in rviz. (1) het cameraframe, (2) het te ontwijken obstakel, (3) het te grijpen object, (4) de tafel.

Door ook hier weer met de *interactive markers* de robot naar een botsingslocatie te bewegen is duidelijk dat MoveIt! de objecten als een obstakel registreert, zoals figuur 32 weergeeft. In dit geval treden geen onverwachte botsingen op van de robot met zichzelf. Aan de hand van dit resultaat is besloten om de padplanning met primitieve vormen uit te voeren. Voor het te grijpen object is gekozen voor een klein blokje hout van 10 x 20 x 10 mm. Het obstakel is een rechthoekige doos van 125 x 240 x 80 mm.



Figuur 32: De robot treedt in botsing met een obstakel. Dit is zichtbaar omdat de links rood gekleurd zijn.

4.3.2.2 Besluit

In tegenstelling tot de octomap treedt de robot niet met zichzelf in botsing bij het gebruik van primitieven (balken, cirkels, e.d.) als voorstelling van de omgeving. De simulatie van de padplanning is daarom uitgevoerd, gebruikmakend van primitieve vormen als obstakels. Door de primitieve vormen te gebruiken is het ook mogelijk om andere omgevingsvoorwerpen bij de padplanning te betrekken. Zo zijn de tafel en het cameraframe toegevoegd aan de visualisatieomgeving. Dit zorgt voor een extra veiligheid voor het bepalen van een botsingsvrij pad.

4.3.3 Besluit

In de eerste stap naar simulaties is een configuratiefile voor de robot gemaakt. Aan de hand daarvan is het mogelijk om de robot voor te stellen in de visualisatieomgeving rviz.

De tweede stap is het kenbaar maken van obstakels aan MoveIt!. De eerste mogelijkheid hiervoor was het gebruik van een octomap. Het visualiseren van een octomap is gelukt en deze werd ook als een obstakel herkend. Er treedt echter een vreemde fout op waardoor de robot in bepaalde configuraties denkt met zichzelf in botsing te treden terwijl dit niet zo is. Padplanning uitvoeren was daardoor onmogelijk. Er is dan besloten om primitieve vormen te gebruiken om de omgeving voor te stellen.

Het gebruik van primitieve vormen maakt padplanning wel mogelijk aangezien hier de robot niet meer het vreemde gedrag van botsingen vertoont. De objecten worden ook hier weer als een obstakel herkend. Het visualiseren van de tafel en het cameraframe zorgt ervoor dat de robot op de hoogte is van deze obstakels, ook al worden deze niet expliciet ingescand.

4.4 Berekening en uitvoering van de padplanning

Voordat er testen op de robot worden uitgevoerd wordt de padplanning gesimuleerd in rviz. Hierin kunnen de werking, gevonden paden en botsingen gemakkelijk visueel gecontroleerd worden. MoveIt! geeft weer of er binnen de maximum planningstijd een pad gevonden is. Rviz is dus maar een hulpmiddel in de ontwikkelingsfase van het bin picken. Zodra er communicatie met de robot is gemaakt kan de padplanning ook in de realiteit getest worden.

4.4.1 Planning op basis van de grijpposities

Bij het ontvangen van een doelpositie, zoals beschreven staat in sectie 3.3, probeert MoveIt! een pad te plannen. Lukt dit binnen een ingestelde maximumtijd van 10s dan wordt het pad uitgevoerd. Lukt dit niet dan laat MoveIt! dit weten zodat er voor een andere grijppositie een pad kan gezocht worden.

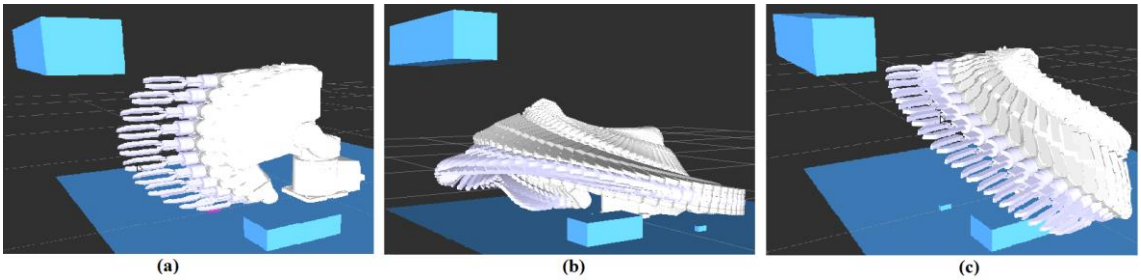
Zodra de robot met zijn eindeffector op de doelpositie is aangekomen wordt het te grijpen object, in de simulatie, aan zijn eindeffector toegevoegd. MoveIt! kan dan bij het berekenen van een volgend pad ook rekening houden met het gegrepen object. In de praktijk moet er ook nog een signaal naar de robot worden gestuurd om zijn grijper te sluiten.

Vervolgens wordt een tweede pad bepaald naar de positie waar het object geplaatst moet worden. Indien zo een botsingsvrij pad bestaat wordt dit uitgevoerd en wordt het object terug in de visualisatieomgeving geplaatst.

In de simulaties komt de grijppositie niet mooi overeen met het te grijpen object omdat er voor de primitieve vormen geen grijpposities zijn. Het is echter mogelijk om met een *interactive marker* de doelpostie in te stellen. Rviz is dan in staat de huidige joint posities te tonen.

In het C++-programma wordt echter de doelpositie bepaald door een punt en een quaternion. In rviz kan de robotpositie enkel in jointverdraaiingen opgevraagd worden. In het gemaakte C++-programma worden echter punten en quaternions gebruikt. Binnen ROS is geen functie aanwezig om de omvorming tussen de 2 formaten te doen. Manueel een nauwkeurige doelpositie instellen is in deze masterproef dus niet mogelijk. Daarom is er gekozen om een positie in te stellen dat boven het te grijpen object ligt. De afstand tussen het object en de eindeffector is klein genoeg zodat er geen obstakels tussenpassen. Dit heeft tot gevolg dat bij het berekenen van een pad het lijkt alsof de grijper het object vastheeft.

Het berekenen van een totale cyclus (bestaande uit de 3 doelposities in figuur 33) duurt gemiddeld 19,7 seconden. In deze tijd wordt een initieel botsingsvrij pad bepaald en daarna vereenvoudigd en geoptimaliseerd. Hierbij is het belangrijk te weten dat de vereenvoudiging en optimalisatie van een pad het meeste van de tijd inneemt. Zonder de vereenvoudiging is de gemiddelde padplanningstijd 7,19s De resultaten van 30 padplanningsimulaties zijn te zien in bijlage 2. Een geldig pad wordt typisch binnen de 3s gevonden.



Figuur 33: De padplanningsimulatie, (a) beweging naar het te grijpen object (kleine balk), (b) beweging naar de eindpositie, over het obstakel (grote balk) heen,(c) beweging naar de home positie.

4.4.2 Besluit

Uit de simulatie blijkt dat het principe van *random bin picking* mogelijk is. Er zijn 30 simulaties uitgevoerd in een identieke omgeving en met dezelfde doelposities om een gemiddelde padplanningstijd te bepalen. De specificaties van de laptop waarop de simulaties zijn uitgevoerd zijn terug te vinden in bijlage 1. In deze simulaties werd ook in 100% van de gevallen een botsingsvrij pad gevonden binnen de maximum planningstijd van 10s. De resultaten zijn terug te vinden in bijlage 2. De gemiddelde padplanningstijd, met vereenvoudiging van het pad, van 19,7s is echter niet snel genoeg om in de praktijk te gebruiken. De industrie verwacht cyclustijden van <4s. De padplanningstijd zonder de vereenvoudiging is gemiddeld 7,19s. Dit is al een stuk korter bij de eisen van de industrie maar nog steeds niet voldoende. In verder onderzoek moet het systeem geoptimaliseerd worden om betere padplanningstijden te bekomen. Hierbij kan best eerst de invloed van padvereenvoudiging en optimalisatie op de padpadplanning bestudeerd worden, aangezien dit niet is gebeurt in deze masterproef.

4.5 Besluit

In dit hoofdstuk is uitgelegd hoe de simulatie van de robot en de omgeving kan gebeuren. Om de robot te visualiseren is eerst en vooral een urdf bestand nodig. Dit bestand bevat de informatie over de links en joints, die weergegeven worden door meshes, en de transformaties

om ze t.o.v. elkaar te visualiseren. Dit urdf bestand wordt dan door de “MoveIt! setup assistant” gehaald om een srdf bestand te verkrijgen dat bruikbaar is in rviz. Dit bestand bevat bovenop de informatie van het urdf bestand ook informatie rond *self collision*, planningsgroepen en de gebruikte eindeffector. Nadat het bestand gemaakt is kan dit geopend worden in rviz.

Vervolgens werd de omgeving van de robot ingeladen in rviz. Dit gebeurde eerst in de vorm van een octomap. Hoewel de robot correct botsingen detecteert met de octomap treedt er een probleem op, zodra de octomap wordt ingeladen treedt de robot met zichzelf in botsing in bepaalde willekeurige configuraties. Dit gebeurt tussen links die nooit met elkaar kunnen botsen. Dit probleem is niet opgelost in deze masterproef zodat er niet te veel tijd verloren zou gaan. Hierdoor is padplanning m.b.v. een octomap niet uitgevoerd.

Ten tweede gebeurde de voorstelling m.b.v. primitieve vormen. In het C++-programma van deze masterproef worden de tafel, het cameraframe, het te grijpen object en het obstakel gevisualiseerd. Vervolgens blijkt ook dat de robot deze primitieve figuren als een obstakel registreert. Omdat hier geen vreemde problemen optreden zijn de padplanningsimulaties in de nieuwe omgeving uitgevoerd.

In totaal zijn er 30 achtereenvolgende simulaties uitgevoerd met hetzelfde startpunt, grijperpositie en uiteindelijke doelpositie. Deze posities, samen met een pad, worden in figuur 33 getoond. De maximum planningstijd in de simulaties is 10s. De simulaties leverden de volgende resultaten op. Indien het pad vereenvoudigd wordt na de berekening is de gemiddelde padplanningstijd 19,7s. Wordt het pad niet vereenvoudigd dan is de gemiddelde planningstijd 7,19s.

De behaalde padplanningstijden zijn niet snel genoeg om aan de eis van <4s cyclustijd vanuit de industrie te voldoen. In toekomstig werk kan de padplanning verder geoptimaliseerd worden. Hierbij kan best eerst de invloed op het pad van de padvereenvoudiging bestudeerd worden, aangezien dit niet is gebeurd in deze masterproef.

In de bijlage op CD-ROM is een handleiding te vinden over ROS die in deze masterproef geschreven is. Hierin wordt uitgelegd hoe de verschillende bibliotheken worden geïnstalleerd. Ook basiskennis over het programmeren met ROS die niet in de oefeningen van de website zitten wordt hier uitgelegd. De voorbeelden die hierin staan komen van de code die gebruikt is in deze masterproef. Deze handleiding is niet af geraakt door tijdsgebrek. Eventuele toekomstige onderzoekers kunnen deze handleiding aanvullen met hun kennis.

5 Padplanning in de realiteit

5.1 Inleiding

Hoewel er al veel informatie uit de simulaties kan gehaald worden is nog steeds het doel om de *bin picking* door een echte robot te laten uitvoeren. In dit hoofdstuk wordt eerst de communicatie tussen de padplanningssoftware en de robot besproken. Tot slot komt de praktische uitvoering aan bod.

5.2 Communicatie tussen padplanningssoftware en de robot

De laatste stap in het *bin picking* proces is de communicatie met de robot maken zodat een gevonden botsingsvrij pad kan worden doorgestuurd. In deze masterproef is ervoor geopteerd om zelf socketcommunicatie te schrijven. Deze communicatie is gemakkelijk op te starten en laat toe berichten in beide richtingen te versturen. Normaal worden ROS drivers gebruikt voor deze communicatie. Voor de Epson robot was deze helaas niet beschikbaar.

De laptop waarop MoveIt! draait wordt rechtstreeks met de robot verbonden via een LAN-kabel. Vervolgens wordt een *client* op de laptop en een server op de robot opgestart. Zodra MoveIt! een pad heeft berekend wordt het padbericht opgeslagen en ontleed in het C++-programma van deze masterproef. Uit het bericht worden, voor ieder punt van het pad, de jointposities voor elk van de assen gehaald. Een functie op de robot wordt daarna opgeroepen die de robot zal aansturen naar deze punten. Als de robot zijn doelpositie heeft bereikt wordt er een bericht teruggezonden naar de *client.* Na deze melding wordt het volgende pad berekend.

5.3 Uitvoering van de padplanning

Het uitvoeren van de padplanning is om veiligheidsredenen eerst manueel uitgevoerd met jointverdraaiingen die uit de padplanner komen. Hiervoor werd nog steeds de C++-code gebruikt die in deze masterproef is geschreven maar werden de waardes manueel via de *terminal* in Linux naar de robot verstuurd. Uit de manuele test blijkt dat de robot correct wordt aangestuurd en dat de posities in de simulatie en de praktijk overeen komen.

In een tweede test is geprobeerd om rechtstreeks vanuit MoveIt! de robot aan te sturen. Per punt op het botsingsvrij pad worden 6 jointverdraaiingen doorgestuurd naar de robot. De robot beweegt doorheen het hele pad zoals zichtbaar is in de visualisatie.

5.4 Besluit

De communicatie tussen de padplanningssoftware en de robot werd volbracht via socketcommunicatie. Het C++-programma van deze masterproef stuurt verschillende punten, die op het berekende pad liggen, naar de robot. Deze punten bestaan uit 6 jointverdraaiingen, 1 voor iedere as. Het is gelukt om de robot volledig automatisch aan te sturen met punten van het botsingsvrij pad en een cyclus te doorlopen. De posities komen ook perfect overeen met simulatie. Het is in deze masterproef niet meer gelukt om de padplanning in de realiteit te evalueren.

6 Besluit

Deze masterproef draagt bij aan de ontwikkeling van automatische, sensorgebaseerde robotprogrammatie, met toepassing op een *bin picking* opstelling. Deze masterproef focust zich voornamelijk op een evaluatie van wat mogelijk is met de huidige open source padplanningssoftware toegepast op een industriële robot.

6.1 Wetenschappelijke en technologische bijdragen

De bijdragen van deze masterproef situeren zich op de volgende gebieden:

1. Ten eerste is er een manier gezocht om de puntenwolk die Halcon genereert om te zetten naar een octomap. Dit is gebeurd met het programma Binvox. De puntenwolk wordt eerst omgezet naar een .binvox bestand om daarna via het C++-programma “octomap2bt” uit de octomap bibliotheek geconverteerd te worden naar een octomap.

   Door verschillende simulaties uit te voeren in Binvox blijkt dat met de parameters –fit (begrenzing van het te converteren gebied) en –e (exacte voxelisatie) een correcte octomap wordt gemaakt binnen de 2 s. Een *leaf size* van < 1,2 mm is haalbaar, wat voldoende nauwkeurig is voor de bin picking opstelling. Het verhogen van de resolutie heeft geen invloed op de omzettingstijd maar het laden van de octomap in MoveIt! zal langer duren met een hogere resolutie.

2. Ten tweede werden de robot en de omgeving gevisualiseerd in rviz. Om de robot te visualiseren was een urdf (Universal Robot Description File) bestand nodig dat door de *MoveIt! setup assistant* naar een srdf (Semantic Robot Description File) bestand wordt omgezet. MoveIt! gebruikt dit bestand om voor de robot een pad te plannen. Rviz gebruikt ook dit bestand om de robot te visualiseren. Door de robot met de *interactive markers* te bewegen is gecontroleerd of alle links juist t.o.v. elkaar bewegen. De robot werd correct gevisualiseerd.

   De omgeving van de robot werd eerst gevisualiseerd door een octomap. De data van de octomap worden door het C++programma van deze masterproef omgevormd naar een berichttype dat MoveIt! begrijpt, namelijk “moveit_msgs::PlanningScene”. Dit bericht wordt op het “planning_scene” topic gepubliceerd, waarop MoveIt! controleert op veranderingen in de omgeving.

   De octomap wordt correct gevisualiseerd en een eventuele botsing met de octomap wordt aangegeven door de robot die rood kleurt. Ook kan er in de motion planning plugin gecontroleerd worden welke links met een voorwerp botsen. Het gebruik van een octomap om de omgeving voor te stellen is dus mogelijk. Zodra de octomap wordt ingeladen is de robot onterecht in botsing met zijn eigen links, in bepaalde botsingsvrije configuraties. Omdat er voor dit probleem geen oplossing is gevonden is er besloten om de omgeving voor te stellen door primitieve vormen.

   De gevisualiseerde obstakels door primitieve vormen zijn:

   - de tafel waar de robot op staat;
   - het frame waar de camera en laser op gemonteerd zijn;
   - een te grijpen object;
   - een obstakel waar een baan rond gepland wordt.

Het aanbieden van deze obstakels gebeurt ook in het C++-programma dat geschreven is voor deze masterproef. Ook hier worden de berichten, deze keer van het type “moveit_msgs::CollisionObject”, op het "planning_scene" topic gepubliceerd. Door de robot te bewegen met de *interactive markers* blijkt dat de robot de objecten als een obstakel herkent. De vreemde botsingen met zichzelf treden hier niet meer op. Daaruit kan besloten worden dat de robot en de omgeving correct worden voorgesteld.

3. Als derde stap werd padplanning in een visualisatieomgeving getest. Alle doorlopen stappen voor de padplanning worden aangestuurd vanuit een zelf geschreven C++-programma. Allereerst worden er doelposities door Halcon berekend op basis van het passen van een 3D CAD tekening van het te grijpen product op de puntenwolk. Deze posities worden vervolgens doorgestuurd naar een tekstbestand om daarna te worden ingelezen in het C++-programma voor gebruik in de padplanning.
   Vervolgens wordt de planner gestart en zoekt MoveIt! een pad naar de geselecteerde doelpositie. Er is een maximale planningstijd ingesteld van 30s. MoveIt! moet binnen deze tijd een geldig botsingsvrij pad vinden. Lukt dit niet, dan laat MoveIt! dit weten zodat er voor een andere grijppositie een pad kan gezocht worden. Het pad wordt berekend volgens het RRT-Connect algoritme. De keuze van het algoritme volgt uit de resultaten van een vorige masterproef.[4]

   De snelheid om een pad te bepalen hangt af van hoeveel obstakels er in de nabije omgeving zijn. Een volledige, gesimuleerde cyclus van object grijpen, verplaatsen en terug naar de home positie bewegen duurt in deze masterproef gemiddeld 7,19s.

4. Ten vierde werd de padplanning op de Epson C3 robot getest. De communicatie tussen de laptop en de robot gebeurt d.m.v. socketcommunicatie. Deze communicatie is zelf geschreven. Normaal zijn hier ROS drivers voor beschikbaar. Voor de Epson robots zijn deze echter nog niet gemaakt.

5. Vanuit het C++-programma van deze masterproef wordt het botsingsvrij pad opgevraagd en ontleed in een 30-tal punten op het pad en de bijhorende jointverdraaiingen. Via de socketcommunicatie worden deze posities doorgestuurd naar de robot. Eerst zijn de berekende posities uit het C++ programma manueel overgetypt uit veiligheidsredenen. Hieruit blijkt dat de robot correct naar een positie kan bewegen, afkomstig uit MoveIt!. De positie in de visualisatie en de realiteit komen overeen.

6. Vervolgens is getracht de volledige cyclus automatisch te doorlopen. Dit is volledig gelukt en daarmee is aangetoond dat *bin picking* mogelijk is, daarmee is de hooddoelstelling van deze masterproef behaald.

6.2 Eigen bijdragen

Deze masterproef vult het onderzoek naar *random bin picking* aan.

1. Allereerst is er een literatuurstudie gemaakt over verschillende opstelling bij andere onderzoeksinstellingen. Hieruit kan de padplanningtijd uit de deze masterproef met hun resultaat vergeleken worden.

2. Vervolgens is er een analyse en evaluatie gemaakt over de omzetting van een puntenwolk naar een octomap. Deze data moet verstuurd worden naar MoveIt! zodat dit geïntegreerd kan worden voor de padplanning. Hiervoor is een C++-programma geschreven dat de

octomap inleest en omvormt naar een berichttype dat MoveIt! begrijpt. Daarna is ook een controle uitgevoerd over de juistheid van de voorstelling in rviz.

3. Ten derde is er op eigen initiatief een handleiding geschreven over het gebruik van ROS en MoveIt!. De handleiding legt alles uit aan de hand van de code die in deze masterproef is geschreven. Deze handleiding is terug te vinden in de bijlage op CD-ROM.

4. Daarna is een visualisatieomgeving in rviz aangemaakt die de omgeving van de robot benaderd. Hierdoor komt de padplanning in simulatie zo getrouw mogelijk overeen met de realiteit.

5. Ten vijfde zijn verschillende padplanningsimulaties uitgevoerd in de visualisatieomgeving om de snelheid en het traject te evalueren.

6. Door het gebrek aan ROS drivers voor de Epson robot, om de robot vanuit ROS aan te sturen, is er voor TCP-communicatie gekozen. Hiervoor is er een C++-programma geschreven dat een verbinding over TCP maakt tussen de laptop met de ROS software en de robot. Hierdoor kan de robot toch aangestuurd worden.

7. Ten slotte is een C++-programma geschreven om een berekend botsingsvrij pad naar de robot door te sturen en deze daarmee aan te sturen. De volledige kring van het *random bin picken* is hiermee gesloten.

6.3 Toekomstig werk

Het doel van deze masterproef was de open broncode ROS en MoveIt! te evalueren voor *random bin picking*. De haalbaarheid van *random bin picking* is succesvol aangetoond. Toch zijn er nog optimalisaties die verder onderzocht moeten worden:

1. In deze masterproef is een puntenwolk omgezet naar een octomap m.b.v. Binvox. Dit duurt echter te lang om in de praktijk bruikbaar te zijn. ROS heeft een aantal bibliotheken beschikbaar die gebruikt kunnen worden voor de omzetting. De octomap- en PCL (Point Cloud Library) bibliotheken bevatten functies om een puntenwolk rechtstreeks om te zetten naar een octomap. Deze bibliotheken worden nog steeds onderhouden en zien er daarom veelbelovend uit. Een tweede uitbreiding rond de octomappen bestaat uit het opsplitsen van de te grijpen objecten in de octomap. Het te grijpen object wordt in de simulatie aan de eindeffector toegevoegd. Daarom moeten de objecten uit de octomap kunnen gehaald worden.

2. De omgeving wordt in deze masterproef voorgesteld door primitieve vormen. Indien de statische omgeving van de robot op voorhand wordt ingescand zou deze omgevormd kunnen worden naar een octomap. De omgeving van de robot in simulatie is dan meer waarheidsgetrouw, wat een invloed kan hebben op de padplanning.

3. In verband met de padplanning moeten de grijpposities vanuit Halcon ook doorgestuurd worden via de TCP-communicatie. In deze masterproef wordt dit m.b.v. een tekstbestand gedaan. Aangezien de TCP-communicatie in orde is, is deze uitbreiding snel te integreren.

4. De totale padplanningstijd en uitvoertijd in simulatie in deze masterproef is gemiddeld 7,19s. Naar de eisen van de industrie is dit niet snel genoeg aangezien een cyclustijd van < 4

s gewenst is. Er kan dus nog verder onderzocht worden hoe de huidige padplanning kan worden geoptimaliseerd.

5. Hoewel de padplanning op de robot is uitgevoerd is er nog geen object gegrepen en verplaatst in deze masterproef. Hiervoor is wel al de code geschreven maar nog niet toegepast omdat de grijppositie niet juist wordt omgezet naar een combinatie van punt en quaternion. De *bin picking* toepassing moet dus verder geëvalueerd worden.

6. Voor de Epson robot zijn geen drivers beschikbaar om de robot vanuit ROS aan te sturen. Een belangrijke bijdrage aan het *bin picking* systeem en aan ROS zou het schrijven van deze drivers zijn.

7. Er is momenteel in de C++-code en in de robotaansturing geen foutafhandeling aanwezig. Beide programma's zouden een foutafhandeling moeten krijgen om geen ongewenste bewegingen toe te staan.

Literatuurlijst

[1] "MVTec HALCON." [Online]. Available: http://www.halcon.com/. [Accessed: 21-Nov-2014].

[2] C. Ioan, A., Sucan; Sachin, "MoveIt!" [Online]. Available: http://moveit.ros.org/. [Accessed: 15-Nov-2014].

[3] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, Feb. 2013.

[4] J. Deneyer and Y. Kenzeler, "Gerandomiseerde padplanningsalgoritmen voor opnemen van willekeurig geplaatste werkstukken met zesassige robot in simulatie," KHLim, 2013.

[5] A. S. C. Hornung, "ROS.org | Powering the world's robots." [Online]. Available: http://www.ros.org/. [Accessed: 15-Nov-2014].

[6] D. Holz, M. Nieuwenhuisen, and D. Droeschel, "Active Recognition and Manipulation for Mobile Robot Bin Picking," *Gearing Up Accel. Cross-fertilization between Acad. Ind. Robot. Res. Eur. Springer Tracts Adv. Robot.*, vol. 94, pp. 133–153, 2014.

[7] R. Bogue Consultant, "Random bin picking: has its time finally come?," *Assem. Autom.*, vol. 34, no. 3, pp. 217–221, 2014.

[8] "octomap - ROS Wiki." [Online]. Available: http://wiki.ros.org/octomap. [Accessed: 30-May-2015].

[9] "PCL - Point Cloud Library (PCL)." [Online]. Available: http://pointclouds.org/. [Accessed: 31-May-2015].

[10] "binvox 3D mesh voxelizer." [Online]. Available: http://www.cs.princeton.edu/~min/binvox/. [Accessed: 23-Nov-2014].

[11] "Available Planners in OMPL." [Online]. Available: http://ompl.kavrakilab.org/planners.html. [Accessed: 29-May-2015].

[12] "PR2/Setup Assistant/Quick Start - MoveIt!" [Online]. Available: http://moveit.ros.org/wiki/PR2/Setup_Assistant/Quick_Start. [Accessed: 29-May-2015].

Bijlagen

Bijlage 1: Specificaties van de laptop

Merk	Fujitsu
Type	Celsius H700
Processor	Intel® Core™ i7 CPU Q 840 @ 1,87GHz x 8
RAM	3,9 GiB
Operating system	Genuine Windows® 7 Professional 64-bit
	Ubuntu release 12.04 (precise) 32-bit
Grafische kaart	NVIDIA® Quadro® FX 880M
Toegewezen videogeheugen	1 GB (GDDR3 VRAM)

Bijlage 2: Resultaten van de padplanningsimulaties

Resultaten van de bin picking simulaties

Nr.	berekening pad 1 (s)	Vereenvoudiging pad 2	berekening pad 2 (s)	Vereenvoudiging pad 2	berekening pad 3 (s)	Vereenvoudiging pad 3	Totale cyclustijd zonder vereenvoudiging	Totale cyclustijd met vereenvoudiging
1	1,94	0,40	1,37	5,81	1,37	0,80	4,68	11,69
2	1,75	0,40	3,04	23,27	1,52	0,91	6,31	30,89
3	1,85	0,41	7,56	5,91	2,28	0,80	11,69	18,81
4	1,81	0,41	2,23	6,24	1,61	1,16	5,65	13,46
5	2,40	0,97	2,43	3,13	2,43	1,47	7,26	12,83
6	2,69	0,98	3,13	8,53	1,77	1,44	7,59	18,54
7	1,84	0,98	2,45	5,70	1,51	0,90	5,80	13,38
8	2,87	0,99	2,53	7,34	1,48	0,90	6,88	16,11
9	1,96	0,40	2,27	9,09	1,41	1,16	5,64	16,29
10	1,85	0,40	1,88	4,47	1,54	0,80	5,27	10,94
11	2,19	0,99	2,04	6,35	1,93	1,30	6,16	14,80
12	1,87	0,40	2,56	20,80	2,93	1,45	7,36	30,01
13	2,21	0,40	2,18	11,19	2,20	1,16	6,59	19,34
14	2,56	0,98	2,01	5,12	1,35	0,98	5,92	13,00
15	1,56	0,40	2,15	12,18	2,15	0,91	5,86	19,35
16	1,87	0,41	3,77	17,60	3,10	1,16	8,74	27,91
17	1,61	0,99	3,28	18,22	1,90	1,26	6,79	27,26
18	2,00	0,40	3,88	13,01	3,52	1,16	9,40	23,97
19	2,57	0,97	3,65	12,17	1,82	1,27	8,04	22,45
20	1,46	0,98	2,24	4,79	2,14	0,91	5,84	12,52
21	1,53	0,40	3,43	16,15	2,63	1,16	7,59	25,30
22	3,38	0,97	3,08	10,41	2,60	1,46	9,06	21,90
23	1,45	0,41	4,66	14,73	1,81	1,46	7,92	24,52
24	2,77	0,40	2,54	5,44	2,45	1,45	7,76	15,05
25	2,33	0,90	3,50	17,68	2,48	1,28	8,31	28,17
26	1,74	0,40	2,73	7,29	1,97	1,16	6,44	15,29
27	1,45	0,40	2,81	15,81	2,56	1,45	6,82	24,48
28	2,23	0,98	5,02	17,45	1,49	0,80	8,74	27,97
29	1,89	0,99	2,62	9,06	2,61	0,81	7,12	17,98
30	1,85	0,98	4,61	6,44	1,90	1,16	8,36	16,94
					Gemiddelde tijd:		7,19	19,705

Pad 1: Van de home positie naar het te grijpen object (grijppositie).
Pad 2: Met het gegrepen object naar de doelpositie bewegen.
Pad 3: Terug naar de startpositie bewegen.

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Experimentele evaluatie van botsingsvrije trajectgeneratie voor 3D random bin picking

Richting: **master in de industriële wetenschappen: energie-automatisering**
Jaar: **2015**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Beuls, Joris

Datum: **1/06/2015**