2015•2016
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: energie

Masterproef
Ontwikkeling van een ROS-driver voor Epson C3 robots

Promotor :
Prof. dr. ir. Eric DEMEESTER

Promotor :
ing. GEERT MOONEN

Luca Castelli , Wim Van der Aelst

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

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

2015•2016
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: energie

Masterproef

Ontwikkeling van een ROS-driver voor Epson C3 robots

Promotor :
Prof. dr. ir. Eric DEMEESTER

Promotor :
ing. GEERT MOONEN

Luca Castelli , Wim Van der Aelst
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

universiteit hasselt | KU LEUVEN

Woord Vooraf

Deze masterproef vormt het sluitstuk van onze opleiding Master in de Industriële Wetenschappen Energie met als focus automatisering.

Langs deze weg zouden wij graag een dankwoord richten tot alle personen die tot het goed einde van deze masterproef hebben bijgedragen.

Allereerst willen wij onze interne promotor Prof. dr. ir. Eric Demeester, en externe promotor ing. Geert Moonen, bedanken voor hun uitstekende begeleiding tijdens onze masterproef.

In het bijzonder willen wij de heer Gijs van der Hoorn, ROS-Industrial Project Manager, TU Delft bedanken. Met behulp van zijn inzicht en deskundige kennis van 'de ROS wereld' hebben wij deze masterproef tot een goed einde kunnen brengen.

Verder willen wij onze Prof communicatie Jeroen Lievens bedanken voor het nalezen en corrigeren van onze thesis.

Ten slotte willen wij ook nog onze ouders bedanken voor ons de opportuniteit te schenken om verder te studeren.

Veel leesplezier,
Luca & Wim

Inhoudsopgave

Woord Vooraf 1

Inhoudsopgave 3

Lijst van figuren 7

Abstract 9

Abstract in English 11

1 Inleiding 13

1.1 Situering 13

1.2 Probleemstelling 15

1.3 Doelstellingen 16

1.4 Materiaal en methode 16

1.5 Bijdrage van onze thesis 17

2 Literatuurstudie 19

2.1 Inleiding 19

2.2 ROS-componenten voor het opzetten van een ethernetcommunicatie met een industriële robot 19

2.2.1 Simple Message 20

2.2.2 Industrial Robot Client 25

2.3 Specificaties van een ROS-driver 27

2.3.1 Initialisering van de socketverbinding 27

2.3.2 Scenario's voor communicatieverlies 27

2.3.3 Correcte link tussen de ROS-omgeving en de ethernetcommunicatie 28

2.3.4 Basisfunctionaliteit 28

2.4 Andere drivers 28

2.5 Besluit 30

3 Ontwikkeling van de driver 33

3.1 Inleiding 33

3.2 Opzetten van een ethernetverbinding 33

3.2.1 Het maken van een server in SPEL+ 33

3.2.2 Simultaan draaien van 2 robotservers 34

3.3 Gebruik van simple messages m.b.v. SPEL+ 35

3.4 Ervaren hindernissen 35

3.4.1 Floating point data wegschrijven of inlezen 35
3.4.2 Meest significante bit (MSB) probleem 36
3.4.3 Writebin/Readbin probleem 37
3.4.4 'Trajectory stop' probleem 39
3.4.5 Trage opstart van MoveIt! 39
3.5 Oplossingen 39
3.5.1 Potentiële oplossingen voor het floating point probleem 39
3.5.2 Potentiële oplossingen voor het MSB-probleem 42
3.5.3 Potentiële oplossingen voor het Writebin/Readbin probleem 42
3.5.4 Oplossing voor het 'Trajectory stop' probleem 43
3.5.5 Oplossing voor trage MoveIt! 43
3.5.6 Finale oplossing 44
3.6 Foutafhandeling 45
3.7 Besluit 47

4 Integratie in MoveIt! 49
4.1 Inleiding 49
4.2 Support package voor de robot 49
4.3 MoveIt! config package voor de robot 53
4.4 Besluit 53

5 Experimentele resultaten 55
5.1 Inleiding 55
5.2 Analyse van het netwerkverkeer 56
5.3 Analyse van de robotbeweging 59
5.4 Analyse van de dode tijd bij een downloadende driver 64
5.5 Besluit 64
5.6 Toekomstig werk 65

Literatuurlijst 67

Bijlagen 69
Bijlage A: trajectory server van de streamende EPSON C3 ROS-driver 69
Bijlage B: state server van de EPSON C3 ROS-driver 73
Bijlage C: Main programma (opstarten van de servers) 77
Bijlage D: simple message protocol include file 77

Bijlage E: robot info include file 78
Bijlage F: geschreven DLL-functies 78
Bijlage G: def file voor de geschreven DLL-functies 80

Lijst van figuren

Figuur 1: random bin picking opstelling te ACRO [1] 14
Figuur 2: Algemeen blokkenschema random bin picking. 15
Figuur 3: De nodige opstelling voor de implementatie van de driver 17
Figuur 4: Voorbeeld JOINT_POSITION simple message 23
Figuur 5: Voorbeeld STATUS simple message 24
Figuur 6: Voorbeeld JOINT_TRAJECTORY_PT simple message 25
Figuur 7: launch file voor het opzetten van een socketverbinding volgens de 'streaming interface' [10] 26
Figuur 8: launch file voor het opzetten van een socketverbinding volgens de'download interface' [10] 26
Figuur 9: Supported hardware by ROS-Industrial [13] 29
Figuur 10: Opstarten van state- en trajectory server bij de EPSON-driver 34
Figuur 11: Zelf gemaakte code ter verduidelijking 36
Figuur 12: De AddToByteArray functie [17] 36
Figuur 13: Binaire code van een byte variabele die bij het gebruiken van Writebin voor een error zorgt 37
Figuur 14: Binaire code van een byte variabele die bij het gebruiken van 'Writebin' niet voor een error zorgt 38
Figuur 15: Binaire code van een integer variabele dat bij het gebruiken van Writebin voor een error zorgt. 38
Figuur 16: Binaire code van een integer variabele dat bij het gebruiken van Writebin voor geen error zorgt. 38
Figuur 17: Binaire code van een long integer variabele die bij het gebruiken van Writebin voor een error zorgt. 38
Figuur 18: Binaire code van een long integer variabele dat bij het gebruiken van ''Writebin' voor geen error zorgt. 39
Figuur 19: Correspondentie tabel [16] 41
Figuur 20: Voorbeeld workaround EPSON-fabrikant 42
Figuur 21: Error code, label en bericht bij een corrupte message 45
Figuur 22: Verschillende sequentienummers [8] 46
Figuur 23:Onderverdeling robot support package [20] 50
Figuur 24:Launch files in een support package van een industriële robot 51
Figuur 25: Coördinatenstelsel EPSON-robot [22] 52
Figuur 26: Commando gestandaardiseerde clients 53
Figuur 27: Wireshark capture van de simple messages die over het netwerk worden geschreven. 56
Figuur 28: Snelheid netwerkverkeer Joint position simple messages 57
Figuur 29: Snelheid netwerkverkeer status simple messages 58
Figuur 30: Snelheid netwerkverkeer trajectory pt simple messages 59
Figuur 31: Beginconfiguratie van de robot bij de robotbewegingsanalyse 60
Figuur 32: Eindconfiguratie van de robot bij de robotbewegingsanalyse 61
Figuur 33: Verloop robotbeweging i.f.v. de tijd 63

Abstract

Dit onderzoek maakt deel uit van de ontwikkeling van *random bin picking* systemen, een van de projecten van de onderzoeksgroep ACRO, KU Leuven. Dergelijke systemen hebben vaak nood aan een padplanningsalgoritme dat botsingsvrije trajecten voor de robot genereert. Het doel van deze masterproef is om een driver te ontwikkelen die de compatibiliteit verwezenlijkt tussen een industriële EPSON C3 robot en het open bron framework 'ROS' dat voorzien is van de padplanningssoftware 'MoveIt!'. Deze driver zet de gegenereerde botsingsvrije paden van MoveIt! om naar werkelijke bewegingen van de robot.

De eisen waaraan de driversoftware moet voldoen zijn de volgende. Ten eerste moet "MoveIt!" ten allen tijde op de hoogte zijn van de werkelijke robotconfiguratie. Ook moet de padplanningssoftware enkele kritische toestanden herkennen zoals o.a. die van de noodstop. Om de driver robuust te maken, moet het programma adequate foutafhandeling voorzien. Zo moet de driver onverwachte communicatiefouten en onrealistische data op een veilige manier opvangen. Ten slotte is het gewenst om een vloeiende robotbeweging te bekomen, wat de levensduur van de robot aanzienlijk vergroot en het energieverbruik verlaagt.

Om de driver tot stand te brengen, werd een socketcommunicatie tussen de ROS-omgeving en de EPSON C3 robot opgezet. ROS stelt hiervoor clients ter beschikking die d.m.v. een specifiek protocol kunnen communiceren met externe machines. Op de robotcontroller werden twee servers geprogrammeerd en getest die data uitwisselen met de clients van ROS.

Abstract in English

This research is part of the development of random bin picking systems at ACRO, a research group of KU Leuven. Such systems often rely on path planning algorithms that generate collision-free paths. The purpose of this master's thesis is to develop a driver that ensures compatibility between an industrial EPSON C3 robot and the open source framework "ROS", which provides the path planning software "MoveIt!". The driver turns the collision-free paths generated by "MoveIt!" into actual movements of the robot.

The movements of the robot must be reliable in all circumstances. Hereby, it is obvious that the driver must adhere to several requirements. First of all, "MoveIt!" must be aware of the actual robot configuration at all times. Furthermore, the path planning software needs to recognize a few critical conditions, e.g. the state of an emergency button. To guarantee the robustness of the driver, the program should handle potential communication errors or unrealistic data. Finally, it is desirable to obtain a smooth movement of the robot, which considerably increases the manipulator's durability.

To establish the driver, a socket communication between the ROS-environment and the robot was arranged. ROS contains an industrial_robot_client package which provides generic clients for connecting to industrial robot controllers to run servers that adhere to a specific message protocol.

1 Inleiding

1.1 Situering

Deze masterproef vindt plaats in ACRO (AutomatiseringsCentrum Research en Opleiding), een onderzoeksgroep van de KULeuven. ACRO verricht onderzoek naar robotica en visietechnologie en daarnaast geeft de onderzoeksgroep opleidingen omtrent industriële automatisering zoals Profibus, Profinet en PLC-programmering.

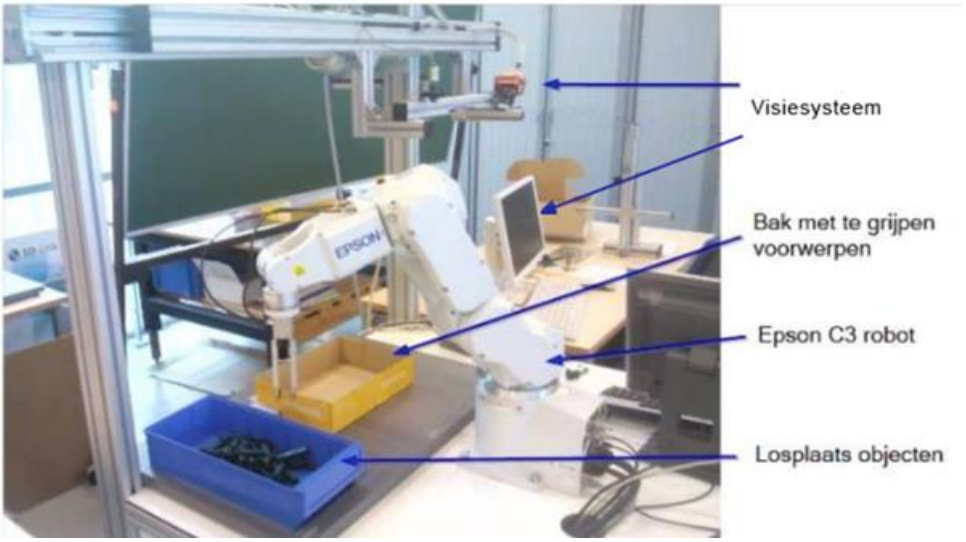
ACRO werkt momenteel aan de innovatie van random bin picking systemen. In de industrie moeten de werkstukken vaak op een heel nauwkeurige locatie aan de robot aangeboden worden zodat deze robot met een voorgeprogrammeerde configuratie het object kan grijpen. Als de werkstukken op een willekeurige manier in een doos liggen, kan een voorgeprogrammeerde robot niet alle objecten uit de doos nemen. Door gebruik te maken van een *random bin picking* systeem kunnen objecten waarvan de positie en oriëntatie op voorhand niet exact gekend zijn (*random*), gelokaliseerd en autonoom gegrepen worden (*picking*) door de robot. Wanneer de robot autonoom objecten uit een doos (*bin*) kan opnemen, spreekt men over random bin picking.

Een random bin picking systeem bestaat uit een visiesysteem, een robot en een padplanningspakket. De werking van zo een systeem verloopt vaak als volgt:

1. De visiehardware levert typisch een 2D beeld of een 3D puntenwolk van de objecten in de doos.
2. De visiesoftware herkent in het beeld objecten door een CAD-model (of primitieve vormen) met het beeld te vergelijken.
3. Door middel van de visiesoftware wordt de 3D positie en oriëntatie van het object uit de doos bepaald.
4. Hieruit kan een botsingsvrije grijpconfiguratie bepaald worden die de robot kan gebruiken om het object uit de doos te grijpen.
5. De bepaalde grijpconfiguratie wordt aan een padplannigssoftware ter beschikking gesteld.
6. Het padplanningspakket berekent met behulp van een gekozen algoritme een botsingsvrij pad tussen de huidige robotconfiguratie en de grijpconfiguratie.
7. Het geplande pad wordt uitgevoerd door de robot.
8. Wanneer de robot het object in de doos heeft gegrepen, berekent het padplanningspakket een botsingsvrij pad tussen de grijpconfiguratie en de ‘place-configuratie’. De ‘place-configuratie’ is de robotconfiguratie die zorgt dat het gegrepen object op de juiste plaats in de werkomgeving terechtkomt zoals bv. de losplaats in figuur 1. Tijdens het plannen van dit traject moet het padplanningspakket het gegrepen object mee in rekening brengen.

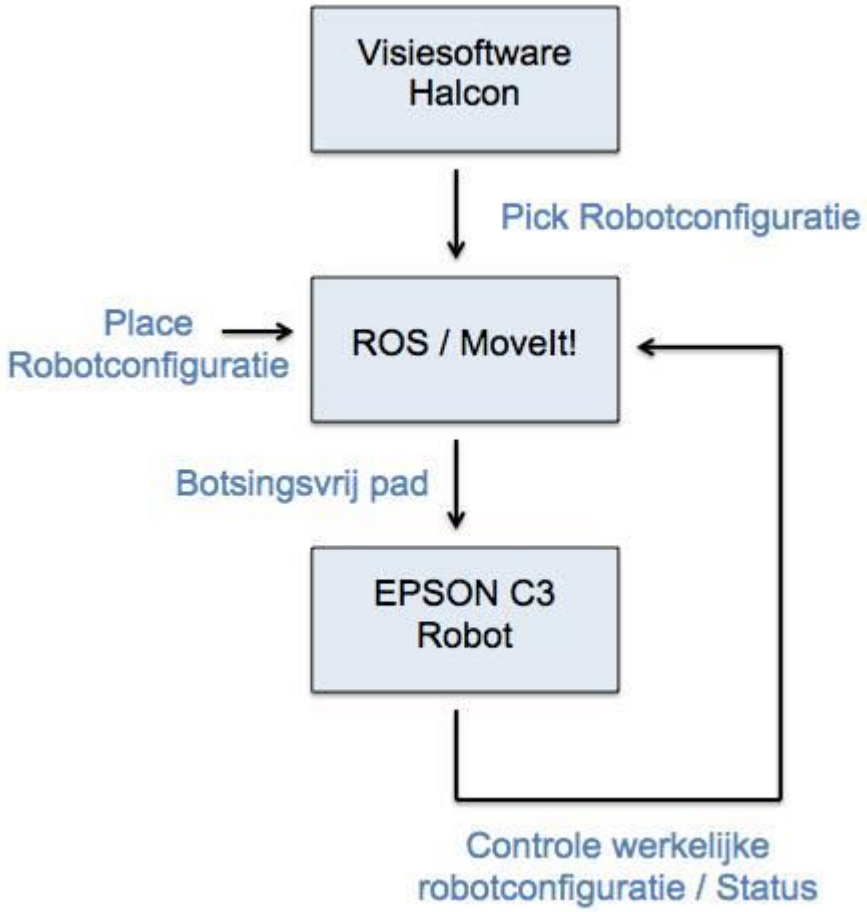
Bepaalde visiesystemen zijn ook in staat om een 3D puntenwolk te produceren van de werkomgeving. Deze werkomgeving bevat vaak obstakels waarmee het padplanningspakket rekening moet houden

Het is dus vanzelfsprekend dat de gedetecteerde obstakels aan het padplanningspakket ter beschikking gesteld moeten worden. Obstakels kunnen ook aan de hand van een CAD-model door een padplanningspakket herkend worden.



Figuur 1: random bin picking opstelling te ACRO [1]

Deze masterproef vertrok van een niet-complete random bin picking opstelling. Figuur 1 toont de opstelling die ACRO ter beschikking stelt. De onderzoeksgroep had reeds een eerste versie van het visiegedeelte van de random bin picking opstelling gerealiseerd. Objecten die in de doos liggen kunnen gelokaliseerd worden met behulp van het softwarepakket Halcon. Het padplanningsgedeelte werd gedeeltelijk verwezenlijkt met de padplanningssoftware 'MoveIt!', dit dankzij de voorgaande masterproeven die in ACRO plaats vonden. 'MoveIt!' kan door middel van het open bron framework 'ROS' eenvoudig gebruikt worden. Sectie 1.2 beschrijft de gebreken van de random bin picking opstelling te ACRO. Wanneer het random bin picking systeem volledig wordt verwezenlijkt, zullen de deelsystemen volgens het blokkenschema uit figuur 2 communiceren. De grijpconfiguratie komende van Halcon wordt door het padplanningspakket gebruikt om een botsingsvrije pad te berekenen. Dit pad wordt naar de robot geschreven zodanig dat het traject uitgevoerd wordt. Door een gewenste robotconfiguratie voor het lossen van het gegrepen object aan de padplanningssoftware ter beschikking te stellen, kan ook een botsingsvrij traject vanuit de grijpconfiguratie (*Pick*) naar de losconfiguratie (*Place*) gepland en uitgevoerd worden. MoveIt! is in staat om het geplande traject met de werkelijke robotbeweging te vergelijken en kan de executie van de beweging stoppen indien de robotbeweging de geplande traject niet blijkt te volgen. Ook werd er een 6-assige EPSON C3 robot te**r** beschikking gesteld.



Figuur 2: Algemeen blokkenschema random bin picking.

1.2 Probleemstelling

Bij de aanvang van de masterproef beschikte ACRO al over een programma dat een botsingsvrij pad (afkomstig van *MoveIt!*) naar de robot kan sturen en op deze manier de robot autonoom kan laten bewegen. Hoewel dit programma functioneert, is de betrouwbaarheid en gebruiksvriendelijkheid maar miniem. Indien bijvoorbeeld de robotcontroller corrupte data inleest, kan de robot een onverwachte beweging maken. Ook wordt de werkelijke robotconfiguratie niet teruggekoppeld naar het padplanningspakket "MoveIt!". Met andere woorden werd een gepland traject door de robot uitgevoerd, maar kent MoveIt! niet ten alle tijde de werkelijke robotconfiguratie. Hierdoor bestaat niet de mogelijkheid om de robotbeweging a.d.h.v. het geplande traject te controleren. Ook spreekt het voor zich dat MoveIt! ten alle tijde de werkelijke robotconfiguratie moet kennen anders kan het padplanningspakket nooit gebruiksvriendelijk werken. Vooraleer een pad gepland zal worden, moet namelijk de werkelijke robotconfiguratie en de robotconfiguratie in MoveIt! overeenkomen, anders kloppen de berekeningen van het padplanningspakket niet. Om bovenstaande problemen op te vangen kwam ACRO met het idee om een zogenaamde ROS-driver (Robot Operating System) voor de EPSON-robot te implementeren in het random bin picking project. ROS is een openbronsoftware die bibliotheken en tools voorziet om softwareontwikkelaars te assisteren bij het opbouwen van een

bepaalde robotapplicatie. Eén van deze tools is de padplanningssoftware *MoveIt!*. Door deze driver te implementeren in een project, wordt de communicatie tussen de robot, padplanningssoftware en andere handige bibliotheken die ROS ter beschikking stelt vereenvoudigd en verbeterd. Zo kan het padplanningspakket MoveIt! eenvoudig gebruikt worden (zonder het schrijven van code) om de robot geplande bewegingen te laten uitvoeren. De ROS-omgeving en dus ook MoveIt! zullen door middel van een ROS-driver ten alle tijde op de hoogte zijn van de werkelijk robotconfiguratie en kritische toestanden zoals bv. die van de noodstop. De implementatie van de driver zorgt voor een comptabiliteit met de gehele ROS-omgeving. Dit wil zeggen dat de data afkomstig van de robotcontroller omgezet wordt naar ROS-berichten binnen het ROS-framework. Hierdoor kunnen de robotdata ook gebruikt worden in de ROS-omgeving om bv. een analyse van de robotbeweging uit te voeren. Voor de EPSON-robot bestaat er nog geen dergelijke ROS-driver. Die moet dus nog geschreven worden om toekomstig onderzoek op random bin picking te vergemakkelijken. [2]

1.3 Doelstellingen

De doelstelling van deze masterproef is om een ROS-driver te ontwikkelen voor de EPSON-robot. De meest voor de hand liggende functionaliteit van een ROS-driver voor industriële robots is het omzetten van interne padplanningsberichten van de ROS-omgeving naar werkelijke robotbewegingen. Er bestaan reeds verschillende ROS-drivers voor sensoren zoals camera's en laserscanners. Ook voor enkele mobiele en industriële robots werden al ROS-drivers ontwikkeld, maar deze drivers zijn meestal nog in een experimentele fase. De EPSON ROS-driver verzorgt de communicatie tussen ROS en de EPSON-robot in kwestie. Wanneer nu de EPSON-robot compatibel wordt gemaakt met de ROS-software, kan de ingenieur hardware (bv. camera's) en software (bv. MoveIt!) die door ROS ondersteund wordt, eenvoudig integreren in zijn project.

Buiten het omzetten van padplanningsberichten naar robotbewegingen moet de driver ook aan de volgende eisen voldoen:

1. Ten eerste moet de ROS-omgeving (en dus ook MoveIt!) met een frequentie van minimaal 1Hz op de hoogte worden gesteld van de werkelijke robotconfiguratie.
2. Ook moet ROS enkele kritische toestanden kennen zoals bijvoorbeeld die van de noodstop (aan of uit). Het is gebruikelijk om deze toestanden, met een frequentie die ongeveer 10 maal lager ligt dan de frequentie uit eis 1, aan de ROS-omgeving ter beschikking te stellen. Voor deze frequentie wordt er echter geen eis opgelegd door de ROS-community.
3. Verder is het ook belangrijk dat ROS weet wanneer er bv. een fout in het robotprogramma is opgetreden.

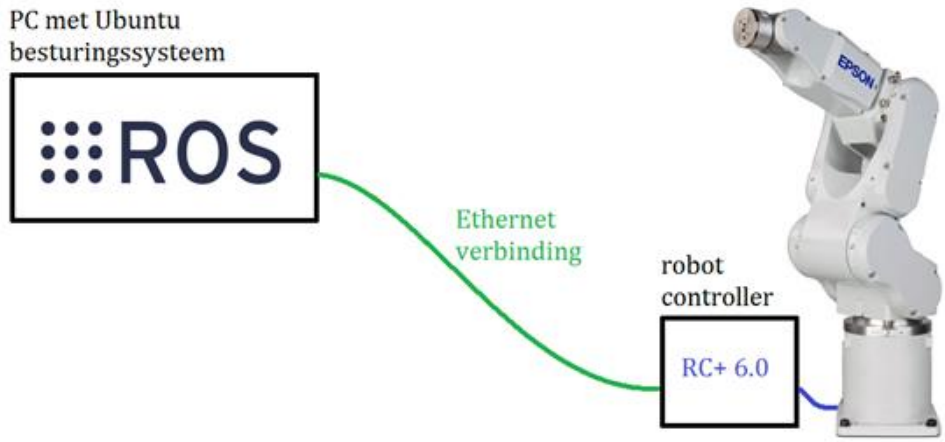
1.4 Materiaal en methode

Zoals in bovenstaande sectie werd vermeld is de doelstelling van deze masterproef om een ROS-driver voor de EPSON C3-robot te schrijven. Om deze opdracht tot een goed einde te brengen moeten er enkele stappen doorlopen worden. Eerst moet men uitzoeken hoe een ethernetverbinding tussen

de ROS-omgeving en de EPSON-robot tot stand kan komen. De ROS-omgeving stelt hiervoor gestandaardiseerde clients ter beschikking die kunnen communiceren met servers. Zo een server kan opgebouwd worden vanuit een robotprogramma van de EPSON-robot. Eenmaal de ethernetverbinding tot stand is gekomen, moet men uitzoeken hoe de communicatie tussen de servers en de ROS-clients moet verlopen. Hierbij moet er rekening gehouden worden met o.a. de berichtstructuur, eisen waaraan de servers moeten voldoen (bv. welke ethernetpoort gebruikt moet worden) en beperkingen i.v.m. de programmeertaal van de robot.

Daarna kan men de serverprogramma's zodanig aanpassen dat de EPSON-controller de actuele stand van de robot naar de ROS-omgeving kan schrijven. Ook kritische robotparameters worden op een gelijkaardige wijze naar de ROS-omgeving geschreven. Op deze manier kan visueel en door toezicht van enkele ROS-berichten gecontroleerd worden of de robotconfiguratie volgens ROS en de actuele configuratie overeenkomen. Indien dit het geval is, kan het robotprogramma verder uitgebreid worden zodat de botsingsvrije trajecten van MoveIt! door de robot kunnen worden uitgevoerd.

Figuur 3 toont de nodige opstelling voor de ontwikkeling en implementatie van de EPSON-driver. Omdat het ROS-framework enkel op een machine kan draaien die gebruik maakt van een Linux besturingssysteem (Ubuntu 14.04 in deze masterproef), werd ROS op een externe computer geïnstalleerd. De robotcontroller maakt namelijk gebruik van een Windows besturingssysteem. Om deze reden moet er een ethernetverbinding gelegd worden tussen de Ubuntu PC en de robotcontroller om zo een communicatie te kunnen opzetten tussen de ROS-omgeving en de robot in kwestie.



Figuur 3: De nodige opstelling voor de implementatie van de driver

1.5 Bijdrage van onze thesis

Tijdens deze masterproef werd een ROS-driver voor de EPSON C3 A601S robot ontwikkeld die de comptabiliteit tussen de ROS-omgeving en de robotcontroller bezorgt door middel van een ethernetcommunicatie. Deze comptabiliteit houdt in dat de driver aan de volgende basisfunctionaliteiten voldoet:

- Botsingsvrije trajecten die door het padplanningspakket MoveIt! worden gegenereerd worden vertaald naar werkelijke robotbewegingen.
- De ROS-omgeving (en MoveIt!) is ten alle tijde op de hoogte van de werkelijke robotconfiguratie.
- De ROS-omgeving (en MoveIt!) is ten alle tijde op de hoogte van enkele kritische toestanden zoals die van de noodstop (aan/uit), de bedrijfsmode van de robot en errors gegenereert door de robotsturing.
- Een vloeiende robotbeweging werd gerealiseerd door het 'downloaden' van het traject.

Verder werd er foutafhandeling aan de driver toegevoegd, die storingen opvangt die kunnen optreden tijdens de werking van de driver. Volgende storingen worden gedetecteerd en opgevangen:

- Corrupte data wordt opgevangen. Met andere woorden, indien de robot een padplanningsbericht inleest dat niet voldoet aan het verwachte protocol,wordt dit bericht genegeerd en geen robotbeweging uitgevoerd.
- Motors off/on foutafhandeling: indien de robot een traject komende van MoveIt! wil uitvoeren terwijl de motoren af staan, kunnen deze tijdens het 'runnen' van de driver nog aangezet worden om errors in het robotprogramma te vermijden.

De driver werd volledig geschreven in de programmeertaal van de robot (SPEL+) en kan hergebruikt worden voor alle EPSON-robots die de RC+6.0 robotsturing aanwenden. Let op: de 'support' en 'moveit config' packages zijn enkel bruikbaar voor de EPSON C3 A601S robot. Indien men de EPSON-driver voor een andere EPSON-robot wil gebruiken, moeten hiervoor nieuwe packages aangemaakt worden (zie hoofdstuk 4). Het is de bedoeling dat de ontwikkelde driver na enkele testen en eventuele uitbreidingen/verbeteringen (door andere ontwikkelaars) voor alle ROS-gebruikers in de wereld online wordt gezet. Op deze manier kunnen alle ROS-gebruikers in de toekomst eenvoudig trajecten plannen en uitvoeren met een EPSON-robot. Er werd namelijk rekening gehouden dat de driver universeel bruikbaar is. Er werden dus geen specifieke robottools gedefinieerd. Hier kan de ROS-gebruiker eenvoudig zelf voor zorgen.

2 Literatuurstudie

2.1 Inleiding

Eerst en vooral is het belangrijk dat de ontwikkelaar leert werken met de ROS-omgeving. Ook moet de ontwikkelaar begrijpen hoe een ethernetcommunicatie tussen de ROS-omgeving en de industriële hardware tot stand kan komen. Daarom gaat sectie 2.2 dieper in op de relevante ROS componenten voor deze masterproef. Ook hoe deze componenten eenvoudig gebruikt kunnen worden, komt in sectie 2.2 aan bod . Daarna komen de specificaties waar een ROS-driver voor een industriële robot aan moet voldoen aan bod. Sectie 2.4 bespreekt de bestaande ROS-drivers voor andere industriële robots en vergelijkt deze met de EPSON-driver. Sectie 2.5 vormt het besluit, en overloopt in het kort de belangrijkste aandachtspunten uit hoofdstuk 2.

2.2 ROS-componenten voor het opzetten van een ethernetcommunicatie met een industriële robot

Het ROS-framework bestaat uit een groot aantal bibliotheken en tools waarmee robotapplicaties opgebouwd kunnen worden [3]. De meerderheid van deze bibliotheken en tools zijn voornamelijk geschikt voor implementatie bij mobiele/service robots. Er is echter een openbronproject binnen de ROS-omgeving dat zich richt op de industriële hardware, namelijk 'ROS-Industrial'. ROS-Industrial bevat allerlei bibliotheken, tools en drivers voor industriële robotapplicaties op te bouwen a.d.h.v. de ROS-architectuur. [4] [5]

ROS-industrial stelt een aantal 'packages' ter beschikking. Een ROS-package is in feite een folder binnen de ROS-omgeving die verschillende zaken zoals bibliotheken, configuratiebestanden, stukjes software of andere handigheden kan bevatten. Het doel van deze packages is om deze nuttige functionaliteiten (bv. software) op een gemakkelijke manier te kunnen hergebruiken binnen de ROS-architectuur. De ROS-architectuur zorgt voor een eenvoudige communicatie tussen verschillende stukjes software van verschillende packages. Sommige van deze packages (zoals bv. de 'Simple Message' en 'Industrial Robot Client' package) bevatten files die gebruikt kunnen worden om de ethernetcommunicatie tussen de ROS-omgeving en de robot tot stand te brengen [6]. Bepaalde industriële controllers van bv. Fanuc en ABB worden door ROS-industrial ondersteund d.m.v. fabrikant specifieke packages [4]. De EPSON-robot hoort hier echter nog niet bij. Het ontwikkelen van de EPSON C3 A601S packages is dan ook het hoofddoel van deze masterproef. Er moeten namelijk 3 ROS-packages gemaakt worden om de compatibiliteit tussen de ROS-omgeving (en MoveIt!) en de EPSON C3 A601S robot te verzorgen. De eerste package moet software bevatten dat de interface tussen ROS-Industrial en de RC+6.0 sturing van de EPSON-robot bezorgt. Deze package is dus herbruikbaar voor andere EPSON-robotvarianten die gebruik maken van dezelfde robotsturing. Dit wordt binnen de ROS-omgeving de driver van de industriële robot genoemd. Ook moet een 'robot support' package (zie hoofdstuk 4) gemaakt worden voor de EPSON C3 A601S robot. In deze package moeten alle bestanden geplaatst worden, die informatie bevatten over de robot. Zo zal de package o.a. CAD-modellen van de robotlinks bevatten. Ook zit vaak in deze package een file dat het robotmodel

beschrijft (jointlimits, draairichting joints, opbouw robot, ...). De laatste package dat voor de EPSON C3 A601S robot gemaakt moet worden is de zogenaamde 'moveit config' package. Deze package bevat alle files die nodig zijn om aan de hand van MoveIt! te communiceren met de robotcontroller. Ook bevat deze package files die zorgen voor de collisiedetectie van de robot (zie hoofdstuk 4). Nadat deze packages voldoende zijn getest, en de bedrijfszekerheid gegarandeerd kan worden, kan de ontwikkelaar deze packages ter beschikking stellen aan alle ROS-gebruikers.

Om een ethernetcommunicatie te realiseren tussen de ROS-omgeving en een industriële robot, stelt ROS-Industrial de 'Simple Message' en 'Industrial Robot Client' package ter beschikking. Deze 2 packages moeten dus zeker geïnstalleerd worden op de Ubuntu machine. Het oproepen van de nodige ROS-files kan dan d.m.v. gestandaardiseerde commando's gebeuren. Voor de ontwikkeling van de EPSON-driver is het voornamelijk belangrijk om te weten wat precies een 'simple message' is. De 'simple message' package definieert een protocol voor het communiceren met een industriële robot over het ethernet. Dit protocol wordt in sectie 2.2.1 meer in detail besproken. Sectie 2.2.2 bespreekt de gestandaardiseerde ROS-clients die de 'Industrial Robot Client' package ter beschikking stelt, deze clients maken gebruik van het simple message protocol. Ook hoe deze clients eenvoudig gebruikt kunnen worden met een standaardcommando wordt in sectie 2.2.2 besproken. [4]

2.2.1 Simple Message

Indien een ethernetverbinding tussen de robotcontroller en de ROS-omgeving tot stand werd gebracht (ip-adressen juist ingesteld, client en server), kan een makkelijke communicatie tussen de twee gerealiseerd worden door het schrijven en inlezen van zogenaamde 'simple messages'. Een simple message is in feite een bericht dat over het ethernet verzonden wordt en dat het zogenaamde simple message protocol volgt. Het simple message protocol definieert de berichtstructuur die nodig is om een ethernetcommunicatie te realiseren tussen industriële robots en gestandaardiseerde ROS-clients [7]. Deze clients kunnen door alle mogelijke industriële robots gebruikt worden indien deze voldoen aan het simple message protocol. Indien men over ethernet wil communiceren met ROS, verwachten de gestandaardiseerde ROS-clients dus berichten die voldoen aan het simple message protocol.

Bij het ontwikkelen van een nieuwe ROS-driver voor industriële robots is het daarom essentieel om het simple message protocol te begrijpen. De algemene berichtstructuur van een simple message bestaat uit [7]. :

- **Prefix**: de prefix van het bericht zijn de eerste 4 bytes die samen een 'Long Integer' ('Long Integer' in robotprogrammeertaal is 4 bytes groot) waarde vormen en op deze manier de lengte van de simple message (Header + Body) aangeven. Dit is voornamelijk handig bij het inlezen van simple messages. Men kan de prefix namelijk gebruiken om in het robotprogramma op een dynamische manier per bericht te werken.

- **Header:** na de prefix volgt de header van het bericht, dit deel van het bericht bestaat uit drie delen:
    - **MSG_type**: Er bestaan verschillende soorten simple messages. Tijdens het communiceren moeten de robotcontroller en de ROS-omgeving in staat zijn om

onderscheid te maken tussen de verschillende berichten. Dit deel van het bericht bestaat uit 4 bytes die een 'Long Integer' waarde vormen en het geeft op deze manier aan over welk berichttype het gaat.

- **COMM_type**: Dit deel van de header bestaat ook uit 4 bytes en geeft het communicatietype aan. Het communicatietype kan bv. als Topic, service request of service reply gedefinieerd worden. Het verschil luidt als volgt [8]:
  - **Topic**: communicatietype dat men enkel moet gebruiken wanneer de zender geen 'acknowledge' van de ontvanger nodig heeft.
  - **Service request**: In tegenstelling tot de 'topic' verwacht de 'Service request' wel een 'acknowledge' van de ontvanger.
  - **Service reply**: Wanneer de ontvanger een 'acknowledge' simple message schrijft naar de verzender moet het communicatietype van het type 'service reply' zijn.
- **REPLY_type**: Het reply deel van de header wordt enkel ingevuld wanneer de ontvanger een 'service reply' naar de zender moet sturen en geeft aan of het bericht goed werd ontvangen of niet. Deze code kan drie waarden aannemen:
  - **Invalid**: simple message is geen 'service reply'
  - **Succes**: het bericht (service request) werd goed ontvangen
  - **Failure**: het bericht (service request) werd niet goed ontvangen

- **Body:** dit gedeelte van het bericht bevat bijvoorbeeld bij een Joint Position simple message de verworven data van de robot zoals joint posities, snelheden, etc. zodat deze data naar ROS geschreven kan worden. Ook omgekeerd kunnen de gewenste hoekposities van de joints (bv. berekend met MoveIt!) zo naar de robotcontroller geschreven worden. De opbouw en grootte van de Body is afhankelijk van het type simple message dat gebruikt werd. De meeste data in de body zijn vaak van het floating point type.

Voor de ontwikkeling van een driver zijn er 3 soorten simple messages van belang. Deze worden meer in detail besproken in de onderstaande secties.

JOINT POSITION Simple Message

Eerst en vooral is het belangrijk dat de ROS-omgeving met een minimale frequentie van 1Hz op de hoogte wordt gesteld van de huidige configuratie van de robot. Indien ROS de huidige configuratie van de robot niet kent, kan er later geen correct traject gepland worden door de padplanningstool MoveIt! omdat MoveIt! de initiële robotconfiguratie moet kennen en ook bij uitvoering van het traject voortdurend verifieert of de robot het traject correct uitvoert. ROS-Industrial stelt hiervoor de 'JOINT_POSITION' simple message ter beschikking. De ontwikkelde driver moet dit bericht zelf opbouwen en met een frequentie groter dan 1Hz naar de ROS-omgeving schrijven. Het is gewenst om de frequentie groot genoeg te maken, hierdoor kan de ROS-omgeving later dan gemakkelijk de beweging van de robot evalueren.

Figuur 4 toont een voorbeeld van een JOINT_POSITION simple message [8]. Zoals deze figuur aangeeft zijn de 4 bytes die de msg_type aangeven ingesteld op de hexadecimale waarde '0000000A', wat overeenkomt met een decimale waarde van 10. Indien het MSG_TYPE gedeelte van het bericht het getal 10 bevat, spreken we dus van een JOINT_POSITION simple message. Zoals hierboven uitgelegd,

betekenen het comm_type (topic) en reply_code (invalid) gedeelte van de header van de JOINT_POSITION simple message dat de robotcontroller een bericht naar ROS schrijft, en geen reply van de ROS-omgeving verwacht. In de body van het JOINT_POSITION bericht zitten 4 bytes die het sequentienummer van het bericht aangeven. Dit gedeelte van het bericht word niet gebruikt bij de terugkoppeling van de werkelijke robotconfiguratie, en wordt dus als een leeg veld naar de ROS-omgeving geschreven. De resterende 40 bytes dienen om de posities van de robotassen naar de ROS-omgeving te verzenden. Onderstaande figuur toont aan dat de simple message 10 velden bevat om de posities van de robotassen (in radialen) mee naar ROS te schrijven. Dat wil dus zeggen dat er 4 bytes worden gereserveerd per jointpositie. Indien de robot maar 6 assen heeft, moeten de laatste 4 velden leeg naar ROS geschreven worden. Met andere woorden moeten in de driver de velden van de JOINT_POSITION simple message ingevuld worden a.d.h.v. de actuele standen van de robotassen. De resolutie van de joint posities (floating point getallen) is afhankelijk van het exponentgedeelte van het datatype en kan berekend worden door de binaire code van het exponentgedeelte en de minimale mantisse waarde (1/(2^23)) in rekening te nemen. De binaire code van het exponent gedeelte is afhankelijk van het aantal vermenigvuldigen of delingen die nodig zijn om een getal tussen 1 en 2 te bekomen.
Voor jointposities tussen de -6.28 en 6.28 radialen zal de laagst mogelijke resolutie (minst nauwkeurig) gelijk zijn aan:
2^(binaire code exponent bij het getal 6.28 - 127) * 1/(2^23)
= 2^2 * 1/(2^23) = 4.76837 * 10 ^ -7 radialen
Dit komt overeen met 0,000027 graden en wordt voor industriële toepassingen als nauwkeurig genoeg beschouwd.
Wanneer één van de robotassen bijvoorbeeld 720° (12,56 radialen) kan verdraaien, bestaat de mogelijkheid dat het exponent gedeelte van het datatype verandert om een groot genoeg getal te bekomen. Hierdoor verandert de laagst mogelijke resolutie (minst nauwkeurig) naar :
2^(binaire code exponent - 127) * 1/(2^23) = 2^3 * 1/(2^23) = 9.53674 * 10 ^ -7 radialen , wat nog steeds als nauwkeurig genoeg beschouwd kan worden.

Hex	Field	Description
	Prefix	
00000038	length	56 bytes
	Header	
0000000A	msg_type	Joint Position
00000001	comm_type	Topic
00000000	reply_code	Unused / Invalid
	Body	
00000000	sequence	0 (unused)
B81AD9FA	joint_data[0]	-0.000036919
B6836312	joint_data[1]	-0.000003916
B7C043F5	joint_data[2]	-0.000022920
B8B81516	joint_data[3]	-0.000087777
B865D055	joint_data[4]	-0.000054792
B8B6365E	joint_data[5]	-0.000086886
00000000	joint_data[6]	0.000000000
00000000	joint_data[7]	0.000000000
00000000	joint_data[8]	0.000000000
00000000	joint_data[9]	0.000000000

Figuur 4: Voorbeeld JOINT_POSITION simple message

STATUS Simple Message

Een ander simple message dat de ROS-omgeving moet ontvangen is het 'STATUS' bericht. Zoals figuur 5 aangeeft is het bericht 40 bytes groot (exclusief de 4 bytes die de lengte van het bericht aangeven). Het MSG_TYPE van het bericht bevat het getal 13. Dit bericht wordt vaak met een frequentie die 10 keer lager ligt dan de frequentie van het JOINT_POSITION bericht, naar de ROS-omgeving geschreven. De Body van het 'STATUS' bericht bevat voornamelijk de kritische toestanden zoals bijvoorbeeld de toestand van de noodstop en de motoren. Ook de bedrijfsmode van de controller (manueel of automatisch bedrijf) wordt aan de ROS-omgeving doorgegeven. Indien er een fout in het programma van de driver opduikt, geeft het STATUS bericht dit aan door de 'error code' naar de ROS-omgeving te verzenden. Ook het veld 'in_error' geeft aan of er een storing in het robotprogramma is opgetreden of niet. Het 'in_motion' veld van het bericht geeft aan of de robot aan het bewegen is of niet. Het 'motion_possible' veld dient om de ROS-omgeving te laten weten of de robot kan bewegen. Indien de ontwikkelde driver niet in staat is om enkele van deze velden correct in te vullen, moet het desbetreffende veld als 'unknown' aangeduid worden in het robotserverprogramma. [8]

Hex	Field	Description
	Prefix	
00000028	length	40 bytes
	Header	
0000000D	msg_type	Status
00000001	comm_type	Topic
00000000	reply_code	Unused / Invalid
	Body	
00000001	drives_powered	True
FFFFFFFF	e_stopped	Unknown
00000000	error_code	0
00000000	in_error	False
00000000	in_motion	False
00000002	mode	Auto
00000001	motion_possible	True

Figuur 5: Voorbeeld STATUS simple message

JOINT TRAJECTORY POINT Simple Message

De vorige twee soorten simple messages worden beide door de driver opgebouwd en weggeschreven naar ROS. De ROS-omgeving moet echter ook berichten naar de robotcontroller sturen, anders is het niet mogelijk om een geplande pad van *MoveIt!* om te zetten naar een actuele robotbeweging.
Wanneer een correcte socketverbinding met de robotcontroller tot stand werd gebracht is de logische volgende stap om een traject te plannen of uit te voeren met *MoveIt!*. Wanneer dit gebeurt, worden door de ROS-omgeving zogenaamde 'JOINT_TRAJECTORY_PT' simple messages gegeneerd die naar de robotcontroller geschreven kunnen worden.
FIguur 6 toont een voorbeeld van zo een bericht. Het bericht (exclusief de 4 bytes die de lengte aangeven) bestaat uit 64 bytes. De msg_type voor deze simple message komt overeen met het getal 11. Het comm_type van dit bericht verschilt echter van de vorige messages. De client van ROS stuurt namelijk een JOINT_TRAJECTORT_PT bericht naar de robotcontroller die zich als server gedraagt. Het bericht is van het type 'service request', wat betekent dat de robotcontroller na het inlezen van dit bericht een 'service reply' moet schrijven naar de ROS-omgeving. De opbouw van zo een 'service reply' simple message mag een lege JOINT_POSITION simple message zijn waarvan de reply_code en comm_type velden juist zijn ingevuld.

Hex	Field	Description
	Prefix	
00000040	length	64 bytes
	Header	
0000000B	msg_type	Joint Trajectory Point
00000002	comm_type	Service Request
00000000	reply_code	Unused / Invalid
	Body	
00000001	sequence	1 (second TrajectoryPoint)
A7600000	joint_data[0]	-0.000000000
3EA7CDE8	joint_data[1]	0.327742815
BF5D9E57	joint_data[2]	-0.865697324
C0490FDB	joint_data[3]	-3.141592741
3F34815F	joint_data[4]	0.705099046
C0490FDB	joint_data[5]	-3.141592741
00000000	joint_data[6]	0.000000000
00000000	joint_data[7]	0.000000000
00000000	joint_data[8]	0.000000000
00000000	joint_data[9]	0.000000000
3DCCCCCD	velocity	0.1
40A00000	duration	5.0

Figuur 6: Voorbeeld JOINT_TRAJECTORY_PT simple message

2.2.2 Industrial Robot Client

De industrial robot client package bevat gestandaardiseerde clients die gebruikt kunnen worden om een ethernetverbinding op te zetten tussen de ROS-omgeving en een industriële robotcontroller. Een eis is echter dat de communicatie tussen de client en de robotcontroller verloopt volgens het simple message protocol. [9]
Het is dus niet nodig om zelfstandig een clientprogramma te maken, ROS stelt dit al ter beschikking. Hierdoor is er ook meteen een link gelegd tussen de simple messages die over de ethernetverbinding worden geschreven, en de berichten die intern in de ROS-omgeving worden gepubliceerd. Het spreekt voor zich dat de robotcontroller zich dus als server moet gedragen en simple messages moet kunnen ontleden en opbouwen. Dit zou in in ieder geval de gemakkelijkste werkwijze zijn indien de programmeertaal van de EPSON-controller dit toelaat.

Binnen deze package wordt er onderscheid gemaakt tussen de zogenaamde streaming- en downloadinterface. Met behulp van de streaminginterface worden traject commando's in 'real time' naar de robot geschreven. De robot moet deze commando's dan meteen omzetten naar bewegingen. De downloadinterface wordt voornamelijk gebruikt wanneer men wenst om eerst alle nodige data in te lezen, en daarna pas het traject door de robot te laten uitvoeren. [10]

ROS-drivers ondersteunen in het algemeen enkel één van deze twee methoden. De ROS-driver die uiteindelijk online wordt geplaatst is namelijk diegene die voor de beste prestaties zorgt. Momenteel heeft binnen de ROS-Industrial community de streaming interface de voorkeur omdat een robotbeweging meteen uitgevoerd kan worden (zonder dode tijd). Enkel de ABB-driver maakt gebruik van de download interface. Tijdens de ontwikkeling van de initiële EPSON-driver werd er dan ook voor de 'streaming interface' gekozen. In een latere fase werd ook een downloadende versie van de driver gemaakt, om op een eenvoudige manier een vloeiende robotbeweging te realiseren. Het is echter de bedoeling om ook een vloeiende beweging bij de streamende driver te realiseren. Hierdoor zouden de prestaties en de gebruiksvriendelijkheid van de streamende driver die van de downloadende versie overschrijden. Bij een downloadende driver zullen er namelijk problemen optreden wanneer men grote/ingewikkelde trajecten plant. Meer informatie omtrent de downloadende driver kan men terugvinden in sectie 5.4

Implementatie van de gestandaardiseerde clients

De clients die werken volgens het simple message protocol kunnen makkelijk gestart worden door ROS commando's in de terminal van de Ubuntu machine uit te voeren. Afhankelijk van de werking van de driver moet dan het juiste commando gekozen worden. Figuur 7 toont het commando dat nodig is om een socketverbinding op te zetten met een industriële robot, waarbij de robot meteen de ingelezen trajectberichten omzet naar bewegingen.

```
roslaunch industrial_robot_client robot_interface_streaming.launch robot_ip:=<value>
```

Figuur 7: launch file voor het opzetten van een socketverbinding volgens de 'streaming interface' [10]

Indien de driver er voor zorgt dat de robot eerst alle data inleest en dan pas deze data gebruikt om het pad uit te voeren dan kan het download commando (zie figuur 8) gebruikt worden om een socketverbinding op te zetten.

```
roslaunch industrial_robot_client robot_interface_download.launch robot_ip:=<value>
```

Figuur 8: launch file voor het opzetten van een socketverbinding volgens de'download interface' [10]

Er bestaat ook een commando om visueel de werkelijke robotconfiguratie te controleren met de 'rviz' applicatie. Dit is visualisatiesoftware binnen de ROS-omgeving die gebruikt kan worden om bijvoorbeeld een industriële robot te visualiseren . Met dit commando worden enkel de JOINT_POSITION en STATUS simple messages ingelezen door de gestandardiseerde ROS-client. In deze situatie worden er dus geen TRAJECTORY_PT simple messages van ROS naar de robotcontroller geschreven. Buiten het ip-adres van de robot, heeft ROS ook de locatie van het URDF (Unified Robot Description file) bestand nodig om zo de industriële robot te kunnen visualiseren. Dit is een bestand onder de vorm van een XML formaat dat een robotmodel voorstelt. Hierbij horen o.a. de CAD-bestanden van de robotlinks, de joint-limits en de maximale snelheden van de joints.

Het gebruik van deze controle gebeurt d.m.v. het volgende commando dat terug te vinden was in de launchfile robot_state_visualize.launch:
roslaunch robot_state_visualize.launch robot_ip:=<value> urdf_path:=<value>

Om de socketcommunicatie met een simpel commando te laten functioneren moet er aan bepaalde eisen voldaan worden. Wanneer men gebruik maakt van de gestandaardiseerde clients, wordt er van de ROS-gebruiker verwacht dat deze gebruik maakt van specifieke TCP poorten. Er wordt namelijk verwacht dat de robotcontroller JOINT_POSITION en STATUS simple messages stuurt over poort 11002. Deze poort dient dus voornamelijk om de actuele status van de robot d.m.v. een ROS-client in te lezen. Poort 11000 dient om bewegingscommando's naar de robot te schrijven. Het spreekt voor zich dat de JOINT_TRAJECTORY_PT berichten over poort 11000 naar de robot gestuurd moeten worden. IO's kunnen dan weer worden aangestuurd of ingelezen via port 11003. [11]

2.3 Specificaties van een ROS-driver

Een ROS-driver kan niet lukraak geschreven worden maar moet aan bepaalde specificaties tegemoetkomen, zowel aan de ROS-kant als aan de robotkant. Uiteraard voldoen de gestandaardiseerde clients aan de ROS-industrial specificaties. De code binnen de EPSON-programmeeromgeving moet echter ook bepaalde specificaties vervullen. De volgende secties geven een aantal richtlijnen om een correcte functionaliteit van de driver te verzekeren. [12]

2.3.1 Initialisering van de socketverbinding

Het systeem heeft de voorkeur om bepaalde code (die zorgt voor de socketverbinding) langs de robotkant automatisch te 'runnen' bij het opstarten van de controller. Dit kan bijvoorbeeld gebeuren door een programma dat altijd wordt uitgevoerd in de achtergrond. Indien er uitzonderingen zijn of andere opstarteisen, moeten deze expliciet vermeld worden in de documentatie van de driver. Over de volgorde van opstarten wordt niets vermeld. In het geval van de EPSON-driver is het gewenst om eerst de robotservers te starten en daarna pa de ROS-clients op te roepen.

2.3.2 Scenario's voor communicatieverlies

In geval van communicatieverlies tussen de ROS-kant en de robotkant, heeft de robot als taak om zijn beweging te stoppen en zijn motoren af te schakelen. Vervolgens dient de robotcontroller alle nodige socketverbindingen opnieuw te initialiseren en te wachten op een nieuwe verbinding. Wanneer de programmeeromgeving van de robot geen communicatieverlies kan detecteren, kan er een zogenaamde 'heartbeat message' gebruikt worden. Dit is een bericht dat op een constante frequentie tussen een server en client wordt verstuurd. Wanneer de ontvanger dit bericht niet meer kan inlezen, resulteert dit in een geval van communicatieverlies.

2.3.3 Correcte link tussen de ROS-omgeving en de ethernetcommunicatie

Deze sectie beschrijft in het kort de link tussen de simple messages die over de ethernetcommunicatie met de robot worden geschreven en de interne communicatie binnen de ROS-omgeving. Door gebruik te maken van de gestandaardiseerde clients wordt deze link automatisch opgezet en moet men zich in feite geen zorgen maken over de correcte omzetting.

2.3.4 Basisfunctionaliteit

Een initiële ROS-driver voor een industriële robot moet op zijn minst, de actuele standen van de robotassen naar de ROS-omgeving kunnen schrijven. Ook moet de robotcontroller de nodige padplanningsdata kunnen inlezen om een robotbeweging te realiseren. Bij een driver voor een industriële robot bevatten deze data de gewenste joint-posities van de robotconfiguraties in het traject, en een constante snelheid (snelheid hoeft niet gebruikt te worden door de robot) waarmee de robot het traject moet uitvoeren (zie sectie 2.2.1). Buiten de bovenstaande essentiële functies, kan de driver nog uitgebreid worden. Zo is het mogelijk om m.b.v. R OS de gewenste snelheden en versnellingen van de robotassen naar de robotcontroller te schrijven (en omgekeerd). Hiervoor zijn natuurlijk andere simple messages nodig. Deze simple messages worden momenteel echter niet ondersteunt door de gestandaardiseerde clients en vallen dus buiten het bestek van deze thesis. Met andere woorden zijn enkel de simple messages beschreven in sectie 2.2.1 compatibel met de gestandaardiseerde clients van de industrial_robot_client package. ROS is voortdurend in een ontwikkelingsfase, het is dus wel bedoeling dat deze (momenteel niet ondersteunde) simple messages in een latere fase door de gestandaardiseerde clients ondersteund zullen worden. Indien men toch de driver wil uitbreiden, moet o.a. de industrial robot client en simple message package aangevuld worden om dit mogelijk te maken voor de desbetreffende robotcontroller.

2.4 Andere drivers

Deze sectie bespreekt de reeds ontwikkelde ROS-drivers voor industriële robots en hun functionaliteiten. Figuur 9 toont een algemeen schema, ter vergelijking is ook de in deze masterproef ontwikkelde ROS-driver voor de EPSON C3 robot toegevoegd. Aan de hand van dit schema worden de verschillende drivers met elkaar vergeleken en toegelicht. Om het schema echter te begrijpen, moeten er nog enkele begrippen verklaard worden [13]:

- **Position streaming**: Een driver behoort tot deze categorie indien de robot de ROS commando's in real-time omzet in bewegingen. De snelheid waarmee de robot de bewegingen uitvoert, wordt bepaald in het robotprogramma. Bij de ontwikkeling van een driver is deze methode vaak het gemakkelijkst te implementeren.
- **Trajectory downloading**: Wanneer de robotcontroller eerst alle trajectcommando's inleest en daarna pas de beweging uitvoert, spreken we van trajectory downloading. Indien de robotprogrammeeromgeving beperkt is en volgens de 'streaming'

werkwijze geen vloeiende beweging van de robot kan realiseren, wordt deze methode toegepast.

- **Trajectory streaming:** Deze methode is vergelijkbaar met 'Position streaming'. Bij 'Trajectory streaming' wordt er echter nog rekening gehouden met de snelheden en accelaraties van de joints, die door ROS worden opgelegd.
- **IO control**: Sommige drivers zorgen ervoor dat ROS in- en uitgangen kan aansturen. ROS-industrial zelf ondersteunt deze aansturing echter nog niet.
- **MoveIt Pkg:** Om met MoveIt aan padplanning te kunnen te doen, moet er een package worden ontwikkeld voor de specifieke industriële robot. Meer informatie hierover volgt in hoofdstuk 4.

Fabrikant	Controller(s)	Position Streaming	Trajectory Downloading	Trajectory Streaming	IO control	Manipulator	MoveIt Pkg
ABB	IRC5	✗	✓	✗	✗	IRB-2400	✗
		✗	✓	✗	✗	IRB-5400	✗
Adept	CX, CS	✓	✗	✗	✗	Viper 650	✗
Fanuc	R-30iA / R-30iB	✓	✗	✗	✗	LR Mate 200iC (all)	✓
		✓	✗	✗	✗	LR Mate 200iD	✓
		✓	✗	✗	✗	M-10iA	✓
		✓	✗	✗	✗	M-16iB/20	✓
		✓	✗	✗	✗	M-20IA(/10L)	✓
		✓	✗	✗	✗	M-430IA/(2F,2P)	✓
		✓	✗	✗	✗	M-900IA/260L	✗
		✓	✗	✗	✗	Andere	✗
Motoman	DX100	✗	✗	✓	✓	SIA10D/F	✗
	FS100	✗	✗	✓	✓	SIA20D/F	✓
	DX200	✗	✗	✓	✗	MH5F	✓
		✗	✗	✓	✗	SDA10F	✓
		✗	✗	✓	✗	Andere	✗
Universal Robot	CB2/CB3	✓	✗	✗	✓	UR5	✓
		✓	✗	✗	✓	UR10	✗
Epson	RC620	✓	✗	✗	✗	C3-A601S	✓

Figuur 9: Supported hardware by ROS-Industrial [13]

Zoals figuur 9 aangeeft, maken de meeste (bestaande) drivers gebruik van de 'Position Streaming' methode, dit is ook de meest courante methode die wordt toegepast bij de ontwikkeling van een driver. De ABB-driver is een uitzondering en gebruikt de 'Trajectory Downloading' interface. De ABB-controller zal dus eerst alle data van een traject in lezen en pas daarna de robotbeweging uitvoeren. De Motoman-driver is het meest uitgebreid t.o.v. de andere drivers en houdt o.a. rekening met de

opgelegde motorsnelheden en -versnellingen van de ROS-omgeving. ROS-industrial werkt momenteel aan de Motoman-driver en wilt IO controle en besturing door middel van simple messages mogelijk maken voor deze driver. [14]De EPSON-driver die tijdens deze masterproef werd ontwikkeld behoort tot de 'Position Streaming' categorie. Ook werd tijdens de ontwikkeling van de driver, een Moveit-package voor de EPSON C3 A601S robot gemaakt. Daarnaast werd ook een downloadende versie van de driver ontwikkeld. Op deze manier was het eenvoudiger om een vloeiende robotbeweging te realiseren. Het is echter de bedoeling om de robotbeweging voor de streamende driver nog te optimaliseren. Hierdoor zal de prestatie van de streamende driver die van de downloadende driver beduidend overschrijden. Vandaar dat in figuur 9 de EPSON-driver tot de 'Position Streaming' categorie hoort. Andere ontwikkelaars kunnen namelijk verder werken aan de EPSON-driver zodat deze in de toekomst online kan komen te staan.
De ontwikkeling van een driver gebeurt voor elke robotcontroller anders omdat o.a. de programmeeromgeving van de robotcontroller sterk kan verschillen. De ABB- of Fanuc-driver werd in feite volledig geprogrammeerd in enkele robotserverprogramma's, dit gebeurt alleen als de programmeeromgeving van de robot dit toelaat. In deze robotprogramma's worden dan o.a. de simple messages ontleed of opgebouwd, om zo te communiceren met de gestandaardiseerde clients van ROS. De UR(Universal Robot)-driver maakt echter geen gebruik van deze clients en gebruikt dus ook niet het simple message protocol om de compatibiliteit tussen de ROS-omgeving en de robotcontroller te kunnen verwezenlijken. Deze driver beschikt ook al gedeeltelijk over de IO control functionaliteit (echter nog in een experimentele fase) . De Motoman-driver volgt dan weer wel het simple message protocol. Het gaat dan voornamelijk over de zogenaamde JOINT_FEEDBACK en JOINT_TRAJECTORY_PT_FULL simple messages. Deze berichten zijn in feite de uitgebreide versie (inclusief jointsnelheden en -acceleraties) van de JOINT_POSITION en JOINT_TRAJECTORY_PT berichten. Bij de ontwikkeling van de Motoman-driver waren echter aanpassingen in o.a. de industrial_robot_client package nodig. Sommige robotcontrollers maken gebruik van een externe motion interface. Zo een interface maakt vaak gebruik van een specifiek protocol om een externe aansturing van de robot te realiseren, het protocol kan dan direct omgevormd worden naar ROS-berichten om de comptabiliteit te verzorgen. Men is dus niet verplicht om het simple message protocol te volgen bij de ontwikkeling van de driver. Indien men het protocol volgt kan men wel gemakkelijk gebruiken van de gestandaardiseerde clients.

2.5 Besluit

Dit hoofdstuk besprak de literatuurstudie die voorafging bij deze masterproef. Sectie 2.2 besprak de verschillende ROS-componenten die van toepassing zijn tijdens deze masterproef. Ook werden de specificaties van de ROS-driver voor een industriële robot vermeld. Sectie 2.4 beschreef in het kort de werking van andere drivers en in welke categorie de EPSON-driver hoort. Ook gaf sectie 2.4 aan dat de EPSON-driver die tijdens deze masterproef ontwikkeld werd, tot de 'Position streaming' categorie behoort. Er werd voor de 'Position Streaming'voornamelijk voor deze methode gekozen omdat dit de meest gangbare manier van werken is bij het ontwikkelen van een driver. Indien de performantie van de robot hier te veel onder lijdt (bv. geen vloeiende beweging), kan men makkelijk overschakelen naar de 'Trajectory downloading' methode. Binnen de ROS-omgeving moet men dan simpelweg gebruik maken van de trajectory downloading interface i.p.v. de streaming interface. Wel moeten er kleine aanpassingen gemaakt worden in de driver zelf, zodat deze ervoor zorgt dat de robot

eerst alle data inleest en dan pas de robotbeweging wordt uitgevoerd. Tijdens deze masterproef werd zo een downloadende versie van de driver gemaakt. Ook maakt de ontwikkelde EPSON-driver gebruik van de gestandaardiseerde clients die ROS ter beschikking stelt. Op deze manier is er automatisch een link gelegd tussen de interne berichtgeving van ROS en de robotcontroller. Hierdoor zijn zo goed als geen aanpassingen binnen de ROS-omgeving nodig. Het enigste waar de toekomstige ROS-gebruiker op moet letten is het kiezen van het juiste (gestandaardiseerde) commando. Meestal komen de opstarteisen online te staan op een ROS-pagina en moet de gebruiker simpelweg enkele richtlijnen volgen.

Bij de ontwikkeling van de EPSON-driver werd gekozen om de driver volledig in de programmeeromgeving van de robot te maken. Dit omdat de programmeeromgeving van de EPSON-robot dit toelaat. De programmeertaal lijkt namelijk veel op Visual Basic. Ook is dit weer de meest courante manier van werken binnen ROS-industrial. Indien toch blijkt dat de programmeertaal van de EPSON-robot bepaalde beperkingen heeft die de ontwikkeling van de ROS-driver verhinderen, kan nog altijd de industrial_robot_client en/of de simple_message package uitgebreid worden om een correcte werking van de initiële driver te verzekeren.

3 Ontwikkeling van de driver

3.1 Inleiding

Dit hoofdstuk beschrijft de methode die gebruikt is om een ROS-driver voor EPSON C3 robots te ontwikkelen (RC+6.0 robotsturing). Eerst en vooral moet er een correcte ethernetverbinding worden opgezet tussen de robotcontroller en de ROS-omgeving. Dit wordt in sectie 3.2 besproken. In sectie 3.3 wordt uitgelegd hoe de desbetreffende servers zijn opgesteld in de programmeeromgeving van de EPSON-robot. Daarna wordt er dieper ingegaan op het gebruik van de simple messages in SPEL+. In sectie 3.4 komen alle problemen aan bod die ons tijdens deze masterproef parten hebben gespeeld en in sectie 3.5 de oplossingen voor deze problemen. Als laatste worden allerhande fouten, errors en bugs die deze masterproef heeft gekend, besproken samen met hun oplossingen.

3.2 Opzetten van een ethernetverbinding

Sectie 2.5 geeft aan dat de EPSON-driver volledig in een robotprogramma werd ontwikkeld. Om de communicatie tussen de robotcontroller en de ROS-omgeving te verzekeren is natuurlijk de juiste ethernetverbinding nodig. Ook werd in sectie 1.4 vermeld dat bij de ontwikkeling van deze EPSON-driver, gebruik wordt gemaakt van de gestandaardiseerde ROS-clients. Sectie 2.2.2 toont aan dat deze clients enkel kunnen gebruikt worden indien men aan bepaalde eisen voldoet. Een belangrijke eis is dat de communicatie met deze clients moet verlopen volgens de juiste ethernetpoorten. Poort 11000 dient voornamelijk voor het versturen/ontvangen van berichten waaruit robotbewegingen voortkomen. Poort 11002 dient voor berichten die de robotstatus aan de ROS-omgeving ter beschikking stelt. Wanneer de keuze is gemaakt om de driver te ontwikkelen binnen de programmeeromgeving van de EPSON-robot, is het logisch dat het robotprogramma zorgt voor de werking van twee servers. Eén van de servers moet dan voor de ROS-verbinding zorgen over poort 11000 (ook wel de trajectory server genoemd) en de andere over poort 11002 (de state server). De volgende secties geven aan hoe de servers in de programmeertaal van de EPSON-robot (SPEL+) kunnen worden opgebouwd. Omdat deze twee servers simultaan moeten werken, moet in het robotprogramma voor elke server een aparte taak opgestart worden die onafhankelijk van de rest van het robotprogramma kan draaien.. Sectie 3.2.2 bespreekt hoe men ervoor kan zorgen dat zo een taak niet gestopt kan worden door bv. het induwen van een noodstop. Ook wordt in het kort vermeld waarom dit precies in de EPSON-driver geïmplementeerd moest worden.

3.2.1 Het maken van een server in SPEL+

De programmeertaal SPEL+ bevat een aantal commando's waardoor het maken van een server d.m.v. een robotprogramma zeer eenvoudig kan verlopen. Het gaat dan voornamelijk over de volgende commando's [15]:

- OpenNet: Dient om een TCP/IP netwerkpoort te openen, ook kan men hier aangeven of de poort zich als een server of een client moet gedragen.
- CloseNet: Dient om de geopende poort terug te sluiten.
- SetNet: Met dit commando kunnen de parameters (ip-adres, poortnummer, …) voor een TCP poort worden ingesteld.
- ChkNet: Met dit commando kan men controleren hoeveel bytes het netwerk bevat, ook kan dit commando gebruikt worden om verbindingsfouten vast te stellen.
- WaitNet: dit commando zorgt ervoor dat er gewacht wordt totdat er een verbinding tussen de client en de server tot stand is gebracht.

Met de bovenstaande commando's en een oneindige while loop kan nu gemakkelijk een server opgebouwd worden die netwerkfouten kan detecteren en data blijft inlezen of wegschrijven totdat de client of de server wordt gestopt. Ter verduidelijking kan men de serverprogramma's terugvinden in de bijlage.

3.2.2 Simultaan draaien van 2 robotservers

Door het Xqt-commando in het robotprogramma te gebruiken kan een nieuwe taak gestart worden die onafhankelijk van de rest van het robotprogramma zal draaien. Door nu elke server als aparte taak uit te voeren in het robotprogramma, kunnen met de gestandaardiseerde clients die over poort 11000 en 11002 communiceren, verbinding gemaakt worden. Bij de ontwikkelde EPSON-driver zullen dus de state server (communicatie over TCP poort 11002) en de trajectory server (communicatie over TCP poort 11000) tegelijkertijd kunnen draaien omwille van de Xqt-commando's die gebruikt worden in het robotprogramma.
Zoals eerder werd vermeld, moet de state server voornamelijk de actuele status (configuratie van de robot, toestanden van motoren, toestand van noodstop, ...) van de robot naar de ROS-omgeving schrijven. ROS-drivers voor industriële robots moeten zodanig worden geïmplementeerd dat de state server dit **ALTIJD** moet doen, ongeacht of een noodstop werd ingeduwd of een error in een robotprogramma is opgetreden. Met andere woorden, de state server moet nooit stoppen met draaien. Dit is ook helemaal niet gevaarlijk omdat in de state server geen robotbewegingen en dergelijke worden geprogrammeerd. Dit gebeurt namelijk wel in de trajectory server. Om ervoor te zorgen dat de state server niet beïnvloed kan worden door errors en dergelijke, moet het Xqt-commando voor de 'state server taak' als 'NoEmgAbort' gedefinieerd worden. Figuur 10 toont hoe de state en trajectory server worden gestart bij de EPSON-driver.

```
3  Function main
4
5  Xqt robot_state_mainv3, NoEmgAbort
6  Xqt robot_trajectory_main_v2, Normal
7
8
9  Fend
```

Figuur 10: Opstarten van state- en trajectory server bij de EPSON-driver

In bovenstaande figuur is 'robot_state_mainv3' de taak die als state server fungeert. 'robot_trajectory_main_v2' is de trajectory server die als een normale taak werd gedefinieerd. Op deze manier is het mogelijk om in deze taak robotbewegingen uit te voeren, wat bij de 'NoEmgAbort' taken niet toegelaten is door de EPSON-programmeeromgeving. [15] [16]

3.3 Gebruik van simple messages m.b.v. SPEL+

De EPSON-programmeeromgeving stelt enkele commando's ter beschikking die gebruikt kunnen worden om te communiceren over een ethernetverbinding. De meest interessante commando's heten 'Writebin' en 'Readbin'. Deze commando's komen van pas wanneer we binaire data over een ethernetverbinding willen inlezen of wegschrijven [15]. De programmeeromgeving bevat ook andere commando's zoals bv. Write, Read en Print# om te communiceren over ethernet. Deze commando's dienen echter om data van het datatype 'string' (m.a.w. ASCII karakters) weg te schrijven en in te lezen. Deze commando's lijken dus op het eerste zicht niet geschikt om met simple messages te werken, omdat een simple message voornamelijk is opgebouwd uit 'floating points' en 'integers'. De 'Writebin' en 'Readbin' commando's zijn daarentegen wel geschikt om variabelen van het 'integer' datatype weg te schrijven of in te lezen waardoor bv. een 'Integer' naar ASCII omzetting niet nodig zal zijn.
Bij de ontwikkeling van een EPSON-driver komt het er (in grote lijnen) op neer dat de state server eerst de JOINT_POSITION en STATUS simple message zal opbouwen (door bv. een grote array te vullen) en deze dan met de Writebin syntax naar de ROS-omgeving zal schrijven; dit proces wordt herhaald in een oneindige while loop. De trajectory server zal gebruik maken van het Readbin commando om de JOINT_TRAJECTORY_PT simple messages in te lezen. Daarna wordt dit bericht in de trajectory server ontleed en de bekomen data gebruikt om de robotbeweging uit te voeren met behulp van bv. een PTP commando; ook dit proces wordt herhaald in een oneindige while loop.

3.4 Ervaren hindernissen

Allerhande fouten en zelfs bugs hebben de ontwikkeling van deze driver aanzienlijk bemoeilijkt en vertraagd. In deze sectie wordt uitgelegd welke problemen tijdens deze masterproef zijn ondervonden. In sectie 3.5 worden de potentiële oplossingen voor deze problemen overlopen. Ook komt de werkelijke oplossing aan bod.

3.4.1 Floating point data wegschrijven of inlezen

Sectie 3.3 beschreef dat de 'Writebin' en 'Readbin' commando's gebruikt kunnen worden om variabelen van het integer datatype binair te kunnen wegschrijven of inlezen over een ethernetpoort. Zoals sectie 2.2.1 aangeeft bestaat een simple message echter niet alleen uit data van het integer datatype maar ook floating points. De jointposities worden in ROS namelijk in radialen

beschreven. De 'Writebin' en 'Readbin' commando's kunnen spijtig genoeg geen data van het floating point datatype binair wegschrijven of inlezen. Ook is er binnen de EPSON-programmeeromgeving geen commando ter beschikking dat floating point data (de jointposities in radialen) kan omzetten naar bv. integers die dezelfde binaire code bevatten. Indien hiervoor een commando zou bestaan zouden de floating points toch met de 'Writebin' en 'Readbin' syntax behandeld kunnen worden. Sectie 3.5.1 beschrijft de potentiële oplossingen voor dit probleem.

3.4.2 Meest significante bit (MSB) probleem

Tijdens de ontwikkeling van de driver zijn er ook een aantal bugs gevonden binnen de programmeeromgeving van de EPSON-robot. Een van de bugs die tijdens de ontwikkeling voor veel moeilijkheden heeft gezorgd, wordt in deze thesis het MSB (meest significante **bit**) probleem genoemd. Zoals de naam doet vermoeden, treden er binnen de robotprogrammeeromgeving fouten op wanneer men met variabelen wil werken waarvan de toegewezen waarde/binaire code een meest significante bit bevat die als 'hoog' (1 of true) werd gedefinieerd. Deze sectie zal deze bug in detail toelichten.
Voordat het MSB-probleem wordt uitgelegd, moet eerst even vermeld worden dat de meeste datatypen (Byte, Integer, Long Integer, Real, ...) binnen de EPSON-programmeeromgeving altijd als 'Signed' zijn gedefinieerd. Wanneer men dus bv. een variabele wil declareren van het datatype 'Byte', kan deze een getal tussen -128 en +127 bevatten. Dit wordt hier vermeld omdat het goed mogelijk is dat er een verband is met het gevonden MSB-probleem. Wanneer de meest significante bit van de variabele op 'hoog' staat, zal men namelijk met een negatief getal te maken hebben.
Indien men nu op een directe manier een negatief getal toekent aan een variabele (van bv. het datatype Byte), zoals onderstaande code beschrijft, treden er binnen de robotprogrammeeromgeving geen problemen op.

```
Byte var
var = -2
```

Figuur 11: Zelf gemaakte code ter verduidelijking

Het MSB-probleem doet zich echter voor wanneer men op een indirecte manier een negatief getal toekent aan de variabelen. Een voorbeeld van dit indirect toekennen beschrijft de volgende code:

```
Function AddToByteArray(ByRef buf() As Byte, wpos As Integer, ival As UInt32)
        buf(wpos + 0) = (ival And &HFF)
        buf(wpos + 1) = ((ival / &H100) And &HFF)
        buf(wpos + 2) = ((ival / &H10000) And &HFF)
        buf(wpos + 3) = ((ival / &H1000000) And &HFF)
Fend
```

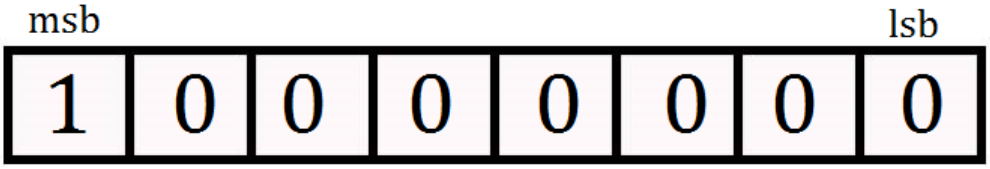
Figuur 12: De AddToByteArray functie [17]

De 'AddToByteArray' functie van figuur 12 werd in een oude (niet werkende) versie van de state server gebruikt en diende voornamelijk om een simple message om te zetten naar een grote byte array die dezelfde binaire code als de desbetreffende message bezat. Dit werd gedaan om zo het

bericht met de Writebin syntax naar de ROS-omgeving te kunnen schrijven. Op deze manier kon er dan gemakkelijk rekening gehouden worden met de 'little endian' of 'big endian' volgorde bij het schrijven van de berichten.
Bij het omzetten van de simple message naar een Byte array waren o.a. variabelen en shift instructies nodig om de juist binaire code aan een 'Byte' element van de array toe te kennen. Dit was dus niet mogelijk door het getal simpelweg hard-gecodeerd toe te wijzen (zoals te zien in figuur 11). In deze situatie kwam het MSB-probleem aan het licht. Indien aan de Byte variabele een binaire code werd toegekend waarvan de meest significante bit als hoog werd gedefinieerd, treedt er een error op in de robotprogrammeeromgeving.
Ook andere datatypen zoals Integers en Long integers leiden onder het MSB-probleem. Wanneer men dus bijvoorbeeld floating points (jointposities van de robot) met een zelfgemaakt robotprogramma wil omzetten naar Integers die dezelfde binaire code bevatten, zal ook het MSB-probleem deze omzetting belemmeren.
Ook werd tijdens de ontwikkeling van de EPSON-driver opgemerkt dat de Writebin en Readbin commando's niet werken indien men een variabele wil schrijven/inlezen waarvan de meest significante bit als hoog werd gedefinieerd. De robotprogrammeeromgeving genereert ook dan een error. Meer informatie omtrent dit probleem komt in de volgende sectie aan bod.
Het MSB-probleem werd tijdens de ontwikkeling van de driver gemeld aan de fabrikant. Op deze manier kan EPSON in de toekomst bij het maken van nieuwe robotsturingen, rekening houden met dit probleem.

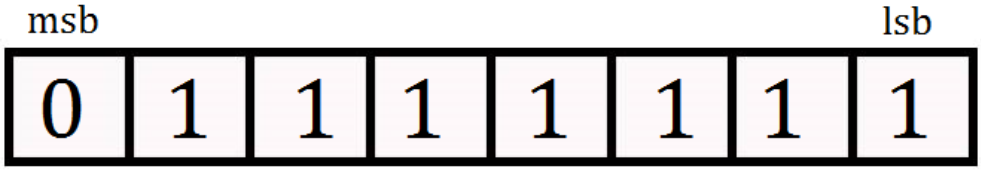
3.4.3 Writebin/Readbin probleem

Tijdens de ontwikkeling van de driver werd ook opgemerkt dat de 'Writebin' en 'Readbin' commando's niet werken zoals het hoort. Om dit probleem beter te kunnen analyseren werden verschillende testen uitgevoerd. Zo werden variabelen van verschillende datatypen (Byte , Integer en Long Integer), uitgebreid getest met behulp van het 'Writebin' en 'Readbin' commando. Een eerste test werd op een variabele van het datatype 'Byte' toegepast. Hieruit bleek dat het 'Writebin' commando de variabele weg kan schrijven indien de meest significante bit **NIET** hoog is, anders treedt er een error op. Voor de verduidelijking toont figuur 13 de binaire code van een 'Byte' variabele die **NIET** door het 'Writebin' commando behandeld kan worden.



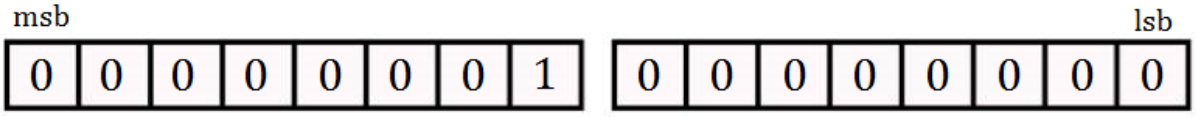
Figuur 13: Binaire code van een byte variabele die bij het gebruiken van Writebin voor een error zorgt

Figuur 14 geeft een voorbeeld van de binaire code van een 'Byte' variabele die wel door middel van het 'Writebin' commando over de ethernetverbinding kan geschreven worden.



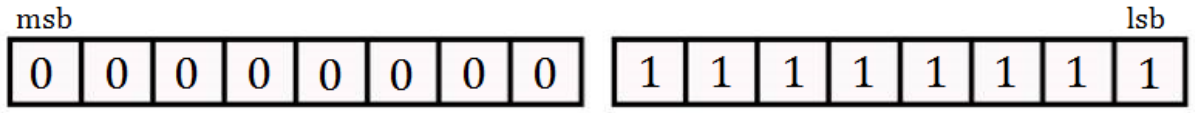
Figuur 14: Binaire code van een byte variabele die bij het gebruiken van 'Writebin' niet voor een error zorgt

De volgende test was het wegschrijven van een variabele van het datatype 'Integer' met het 'Writebin' commando (2 bytes groot). Ook hieruit volgen ongewone resultaten. Het 'Writebin' commando zal in dit geval enkel werken wanneer de hoogst significante **BYTE** van de integer variabele leeg wordt gelaten. Het 'Writebin' commando kan dus enkel een integer waarde van -128 tot 127 wegschrijven of met andere woorden, enkel de minst significante byte van de integer variabele kan het 'Writebin' en 'Readbin' commando verwerken. Wanneer de hoogst significante byte niet leeg is, treedt er namelijk een error op in de EPSON-programmeeromgeving. Ter verduidelijking toont figuur 15 de binaire code van een 'Integer' variabele die niet door het 'Writebin' commando behandeld kan worden.



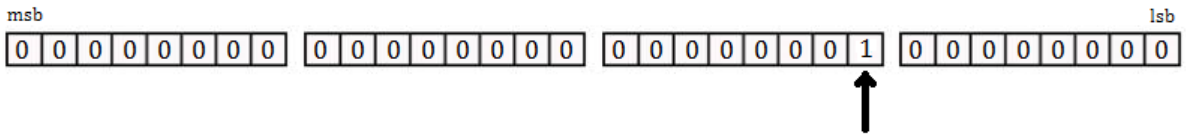
Figuur 15: Binaire code van een integer variabele dat bij het gebruiken van Writebin voor een error zorgt.

Figuur 16geeft een voorbeeld van de binaire code van een 'Integer' variabele die wel door middel van het 'Writebin' commando over de ethernetverbinding geschreven kan worden.



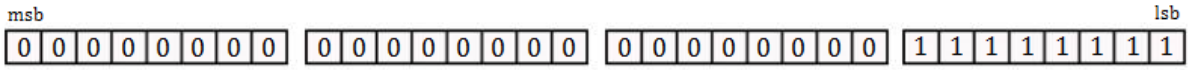
Figuur 16: Binaire code van een integer variabele dat bij het gebruiken van Writebin voor geen error zorgt.

De derde test behandelt de 'Long integer' (4 bytes groot) variabele. Bij deze variabele treedt er een error op wanneer één van de 3 meest significante bytes niet leeg is. Met andere woorden, ook bij deze variabele mag enkel de minst significante byte gevuld worden (waarde van -128 tot 127), anders genereert het 'Writebin' commando ook hier een error. Voor de verduidelijking toont figuur 17 de binaire code van een 'Long integer' variabele die niet door het 'Writebin' commando behandeld kan worden.



Figuur 17: Binaire code van een long integer variabele die bij het gebruiken van Writebin voor een error zorgt.

Figuur 18 geeft een voorbeeld van de binaire code van een 'Long integer' variabele die wel door middel van de 'Writebin' commando over het ethernet geschreven kan worden.



Figuur 18: Binaire code van een long integer variabele dat bij het gebruiken van "Writebin' voor geen error zorgt.

Ook werd aan de hand van de TCP client/server tool 'Hercules' opgemerkt dat bij het schrijven van 'Integer' en 'Long integer' variabelen met 'Writebin', enkel de minst beduidende byte effectief over de ethernetpoort wordt geschreven. Wanneer men dus een waarde aan een integer variabele toewijst waarvan de meest beduidende byte leeg is, zal deze lege byte niet weggeschreven worden door middel van het 'Writebin' commando.

3.4.4 'Trajectory stop' probleem

Tijdens het inlezen van de JOINT_TRAJECTORY_PT berichten genereerde ROS een zogenaamd 'Trajectory Stop' bericht dat ervoor zorgde dat de ROS-omgeving geen JOINT_TRAJECTORY_PT berichten meer naar de robot schreef. Hieruit volgt dat de robot maar een gedeelte van de geplande beweging uitvoerde. Sectie 3.5.4 beschrijft de oplossing voor het 'Trajectory Stop' probleem.

3.4.5 Trage opstart van MoveIt!

Dit probleem wordt in hoofdstuk 4 behandeld.

3.5 Oplossingen

In sectie 3.4 werden allerlei problemen vermeld die de ontwikkeling van de driver belemmerden. Deze sectie beschrijft welke oplossingen hiervoor gevonden werden om een werkende EPSON-driver te bekomen.

3.5.1 Potentiële oplossingen voor het floating point probleem

Het floating point probleem dat sectie 3.4.1 beschrijft kan op verschillende manieren aangepakt worden. Een eerste oplossing kan men bekomen door een robotprogramma te schrijven dat de floating point data omzet naar integers die dezelfde binaire code bevatten. Hierbij moet rekening gehouden worden met de beperkingen van de programmeertaal van de robot. In het geval van de

EPSON-controller moet er dus ook rekening gehouden worden met het MSB en Writebin/Readbin-probleem zoals aangehaald in sectie 3.4.3. De programmeeromgeving van de EPSON-robot stelt shiftinstructies en dergelijke ter beschikking waarmee een omzetting binnen een robotprogramma mogelijk is. Zo kan o.a. de mantisse en het exponentgedeelte van het 'floating point' datatype hier manueel in worden opgebouwd. Een stappenplan dat hiervoor gebruikt kan worden luidt als volgt:

1) Zet a.d.h.v. het teken van de 'floating point' de meest beduidende bit van de 'Long integer' hoog of laag.
2) Vermenigvuldig of deel het 'floating point' getal met 2 totdat men een getal tussen 1 en 2 bekomt.
3) De binaire code van het exponentgedeelte van de 'floating point' kan bepaald worden m.b.v. het aantal vermenigvuldigingen of delingen.
4) Sommeer deze binaire code met de 'Long Integer' van stap 1. Gebruik een shiftinstructie om de binaire code op de juiste plaats te zetten in de 'Long Integer' variabele.
5) De overgebleven fractie van het getal (m.a.w. het getal achter de komma van het getal uit stap 2) in binaire vorm schrijven. Een voorbeeld luidt als volgt: $0.570313 = 1/2^1 + 1/2^4 + 1/2^7 \rightarrow$ De mantisse bestaat uit 23 bits dus de binaire code van de overgebleven fractie is $10010010000000000000000_2$.
6) Sommeer de binaire code van stap 5 met de 'Long Integer' van stap 4.

Dit stappenplan houdt echter geen rekening met het MSB- en Writebin/Readbin-probleem van de EPSON-programmeeromgeving en kan dus alleen gebruikt worden indien alles binnen de programmeeromgeving werkt zoals het hoort. Dit stappenplan kon dus niet gebruikt worden in deze masterproef omdat de desbetreffende robotcontroller een RC+6.0 sturing is en leidt onder het MSB- en Writebin/Readbin-probleem.

Een tweede manier waarop het 'floating point' probleem aangepakt kan worden, wordt gerealiseerd met een zogenaamde 'DLL' (Dynamic-Link Library). Een DLL is een bibliotheek die code bevat dat door meerdere programma's tegelijkertijd gebruikt kan worden [18]. De EPSON-programmeeromgeving laat namelijk toe om functies die in een DLL zitten, op te roepen in het robotprogramma [16]. Dit gebeurt a.d.h.v. het 'Declare' robotcommando. Om het 'floating point' probleem op te lossen, kan dus een DLL aangemaakt worden met bijvoorbeeld de development tool Visual Studio. In deze DLL moet dan een functie geschreven worden (in de programmeertaal C of C++) die de floating point variabelen eenvoudig omzet naar Integers met dezelfde binaire code. Sectie 6.25 van de EPSON-sturing (RC+6.0) handleiding geeft aan hoe in Visual Studio een DLL aangemaakt kan worden. Ook toont de handleiding hoe een eenvoudige functie in de aangemaakte DLL geprogrammeerd kan worden en hoe de link wordt gelegd met de robotprogrammeeromgeving. Figuur 19 toont de datatypen van de robotprogrammeeromgeving en met welke DLL-dataypen deze overeenkomen.

EPSON RC+ 6.0	C/C++
Boolean	short
Byte	short
Short	short
Integer	short
Long	int
Real	float
Double	double
String	char [256] * Null included

Figuur 19: Correspondentie tabel [16]

Tijdens het schrijven van een DLL-functie is het belangrijk dat men rekening houdt met de correspondentietabel van figuur 19. Uit de tabel volgt dat men in de DLL-functie geen gebruik kan maken van de C/C++ 'byte' en 'structure' datatypen omdat hier geen SPEL+ datatype mee overeen kan komen.
Tijdens deze masterproef werd een DLL gemaakt waarin verschillende functies werden geprogrammeerd, die men kan oproepen met het 'Declare' commando in het robotprogramma. Visual Studio 2008 werd geïnstalleerd op de robotcontroller. Voordat men de DLL gaat gebruiken in het robotprogramma, moet het DLL-programma succesvol gecompileerd worden en moet de DLL-file in het SPEL+-project gekopieerd worden. Indien men de DLL-file wil maken met een andere computer (dus niet de robotcontroller), kan het zijn dat het robotprogramma de DLL-file niet kan vinden bij het compileren. Voor de zekerheid kan men best Visual Studio installeren op de controller zelf zodat alle nodige files voor het oproepen van een DLL op de controller zijn geïnstalleerd.

Buiten het schrijven van een robotprogramma dat de floating points om Dan is namelijk de sequentienummer die over de ethernetverbinding geschreven wordt negatief (meer hierover in de volgende sectie).
zet en het oproepen van een DLL, is er nog een derde mogelijke oplossing voor het floating point probleem. Het is namelijk mogelijk om met het Write commando ASCII karakters te schrijven naar de ROS-omgeving. Door de 'simple message' en/of 'industrial robot client' package uit te breiden is het voor de ROS-clients mogelijk om de berichten van de robotcontroller onder de vorm van ASCII karakters in te lezen. Deze berichten hebben dus nog niet de juiste binaire code! De ROS-packages moeten dan zodanig aangepast worden dat de omzetting van het datatype 'String' naar 'Long Integers' en 'floating points' in de ROS-omgeving zelf gebeurt. Deze omzetting kan bv. juist na het inlezen van het bericht gerealiseerd worden. Ook omgekeerd (ROS schrijft een bericht naar de robotcontroller) moet er aan de ROS-kant een omzetting gebeuren die de simple message omzet naar het datatype 'String', waardoor de robotcontroller dit bericht makkelijk kan inlezen met het 'Read' commando. Binnen de programmeeromgeving van de EPSON-robot bestaan er commando's die floating points omzetten naar het 'String' datatype en omgekeerd.

3.5.2 Potentiële oplossingen voor het MSB-probleem

Het MSB-probleem dat in sectie 3.4.2 beschreven wordt kan ook op verschillende manieren omzeild worden. Een eerste workaround voor dit probleem werd tijdens deze masterproef door de EPSON-fabrikant aan ons bezorgd. Figuur 20 toont een stukje code om de workaround te verduidelijken. In deze code zal de variabele 'byteval2' een hexadecimale waarde toegekend krijgen die een meest beduidende bit bevat die als hoog werd gedefinieerd. In de derde regel zal de robotprogrammeeromgeving een error genereren omdat op een indirecte manier een negatief getal aan de variabele werd toegekend. De 'byteval' variabele toont de workaround die aan ons werd bezorgt. Door het gebruiken van een minteken (zie tweede regel van de code), zal byteval gelijk zijn aan het getal -128 en geen error genereren.

```
Byte byteval,byteval2
byteval = -(&H80)
byteval2 = &H80
```

Figuur 20: Voorbeeld workaround EPSON-fabrikant

Om deze workaround te gebruiken moest er dus een onderscheid kunnen gemaakt worden tussen de data die geen meest significante bit als hoog hebben gedefinieerd en de data die wel zo zijn opgesteld. Indien dit onderscheid niet wordt gemaakt zal de data die normaal als positief is gedefinieerd, negatief worden. Hiermee is dus ook het Writebin/Readbin-probleem niet opgelost, waardoor de ontwikkeling van de EPSON-driver nog steeds belemmerd wordt.

Een tweede manier komt overeen met de oplossing voor het floating point probleem. Door de robotcontroller simple messages te laten schrijven onder de vorm van de ASCII codering kunnen in de ROS-packages aanpassingen gemaakt worden zodanig dat daar de juist omzetting (ASCII karakters naar Integers en floating points) gemaakt zal worden. Wanneer de robotcontroller berichten moet inlezen, zal de ROS-omgeving ook de juiste omzetting moeten maken zodanig dat de EPSON-programmeeromgeving met het 'Read' commando de data kan inlezen onder de vorm van ASCII karakters.

De vorige manieren hebben allen hun nadelen. Tijdens de ontwikkeling van de EPSON-driver werd echter voor een ander manier gekozen. Deze oplossing wordt in sectie 3.5.6 finale oplossing beschreven en is de eenvoudigste en meest praktische workaround om de meeste problemen die in sectie 3.4 werden besproken in één klap te omzeilen.

3.5.3 Potentiële oplossingen voor het Writebin/Readbin probleem

Het Writebin/Readbin-probleem kan ook op verschillende manieren omzeild worden. Een manier om dit probleem te vermijden is door geen gebruik te maken van de Writebin en Readbin commando's maar wel van 'Read' en Write'. Deze eerste potentiële oplossing komt grotendeels overeen met een oplossing van het MSB en floating point probleem. Zo kan o.a. de simple message en/of industrial robot client package van de ROS-omgeving aangepast/uitgebreid worden zodanig dat de robotcontroller de simple messages in de vorm van ASCII karakters kan inlezen en wegschrijven. Een

andere oplossing is mogelijk door alle data van de simple message in een DLL-file om te zetten naar ASCII karakters die in zijn geheel dezelfde binaire code bevatten. Op deze manier kan de simple message m.b.v. het 'Write'-commando naar de ROS-omgeving geschreven worden zonder dat er aanpassingen aan de ROS-kant nodig zijn. Ook wanneer de robot data inleest, gebeurt dit met het 'Read' commando waardoor in een DLL-file de ASCII karakters omgezet moeten worden naar Integers en floating points die dezelfde binaire code bevatten. De Write en Read robotcommando's interpreteren echter lege bytes, die vaak voorkomend zijn bij simple messages, als een 'End of Line terminator' waardoor de rest van het bericht niet ingelezen/geschreven wordt. Met deze kleine belemmering moet dus ook rekening gehouden worden.

Een andere oplossing voor het Writebin/Readbin-probleem gebruikt één van de bugs in zijn voordeel. Sectie 3.4.3 gaf aan dat bij het schrijven van integer variabelen d.m.v. 'Writebin', enkel de minst beduidende byte effectief over de ethernet poort wordt geschreven. Wanneer er nu voor elke byte van de simple message een Integer variabele (2 bytes) wordt gereserveerd, zal dus altijd de meest significante byte leeg blijven. Zoals sectie 3.4.3 vermeldt, zal Writebin en Readbin geen error genereren wanneer de meest significante byte van een integer variabele leeg is. Hierdoor kan ook eenvoudig het MSB-probleem vermeden worden omdat op deze manier de meest significante bit nooit als hoog gedefinieerd zal worden.

3.5.4 Oplossing voor het 'Trajectory stop' probleem

De gestandaardiseerde clients van ROS maken gebruik van een zogenaamde 'Watchdog Timer'. Deze timer controleert of binnen de één seconde, de robotcontroller zijn werkelijke configuratie naar de ROS-omgeving heeft geschreven (JOINT_POSITION simple message). Indien de werkelijke robotconfiguratie niet binnen de één seconde door de ROS-omgeving werd ingelezen/ontvangen, zal er een JOINT_TRAJECTORY_PT bericht naar de robotcontroller geschreven worden waarvan de sequentienummer gelijk is aan het getal -4 (zie opbouw TRAJECTORY_PT bericht sectie 2.2.1). Wanneer het sequentienummer gelijk is aan het getal -4, spreken we van een 'Trajectory Stop' bericht. Hieruit volgt dat er geen gegenereerde padplanningsberichten meer naar de robotcontroller geschreven zullen worden en de robot zijn geplande beweging niet zal afmaken. Dit probleem wordt aangepakt door ervoor te zorgen dat de robotcontroller met een periode kleiner dan één seconde de werkelijke robotconfiguratie naar de ROS-omgeving schrijft. Het is dus gewenst om zo weinig mogelijk delays in de ROS-driver te integreren. Indien de ontwikkelde driver langer dan één seconde nodig heeft om de robotconfiguratie aan de ROS-omgeving ter beschikking te stellen, kan de ROS-omgeving de robot niet laten bewegen.

3.5.5 Oplossing voor trage MoveIt!

De oplossing voor dit probleem wordt in hoofdstuk 4 behandeld.

3.5.6 Finale oplossing

Sectie 3.5.3 gaf aan dat één van de EPSON-bugs bij de ontwikkeling in ons voordeel gebruikt kan worden door voor elke byte van de simple message een integer variabele te reserveren. Door middel van deze werkwijze kan dan ook meteen het MSB-probleem vermeden worden. Tijdens de ontwikkeling van de EPSON-driver werd ervoor gezorgd dat de JOINT_POSITION berichten op een eenvoudige manier (zonder fouten en problemen) naar de ROS-omgeving geschreven worden. Het omzeilen van alle problemen binnen de EPSON-programmeeromgeving werd a.d.h.v. het volgende stappenplan gerealiseerd:

1) Eerst moeten de juiste waarden toegewezen worden aan o.a. de Prefix, de velden van de Header en het sequentieveld van de simple message. Alle waarden worden toegewezen aan een 'Long Integer' variabele.
2) De jointposities van de robot worden opgevraagd en omgerekend naar radialen. Daarna worden deze in floating point variabelen opgeslagen.
3) Een DLL-functie wordt opgeroepen vanuit het robotprogramma. Alle data van de simple message (Long integers en floating points) worden als parameter meegegeven aan de DLL-functie.
4) In de DLL-functie wordt alle data van de opgebouwde simple message afkomstig van het robotprogramma omgezet naar een array van integer variabelen. Elke integer variabele bevat de binaire code van één byte uit de simple message. Hieruit volgt dat de volgorde van deze bytes uiteraard correct moet zijn.
5) De integer array wordt d.m.v. het 'Writebin' commando naar de ROS-omgeving geschreven. Omdat de meest significante byte van de integer variabelen (die nu leeg is) niet mee wordt weggeschreven (omwille van de EPSON-bug) naar de ROS-omgeving, is de binaire code die over het ethernet wordt verzonden correct en ontvangt ROS de werkelijke robotconfiguratie.

De STATUS simple messages worden op een gelijkaardige manier naar ROS geschreven. Het enigste verschil is dat nu niet de jointposities opgevraagd moeten worden in het robotprogramma, maar wel de toestand van enkele kritische parameters. Er wordt dus geen gebruik gemaakt van floating point variabelen.

De trajectory server moet de JOINT_TRAJECTORY_PT berichten die ROS/MoveIt! genereert, omzetten naar bewegingen (Let op: er werd gebruik gemaakt van de streaming interface). Dit gebeurt a.d.h.v. het volgende stappenplan:

1) De simple messages worden ingelezen door middel van het 'Readbin' commando. Met behulp van dit commando wordt iedere ingelezen byte opgeslagen in een integer variabele (2 bytes).
2) Door middel van shiftinstructies worden de Prefix, de Headervelden en het sequentienummer berekend a.d.h.v. de binaire code van de ingelezen bytes.
3) De ingelezen data die omgevormd moeten worden naar floating point variabelen (de jointposities, velocity en duration velden van het bericht) worden als parameter aan een DLL-functie meegegeven. De DLL-functie zal de ingelezen bytes omzetten naar de correcte floating point variabelen.
4) De floating point variabelen die de gewenste waarden van de jointposities (in radialen) bevatten, worden omgezet naar graden.
5) In het robotprogramma wordt a.d.h.v. de verkregen jointposities een robotconfiguratie 'geteached'.

6) Een bewegingscommando wordt uitgevoerd.

Indien één van de long integer variabelen een negatief getal bevat (Prefix, Header of sequentienummer), zal het MSB-probleem weer voor belemmeringen zorgen. In dit geval kan bijvoorbeeld de workaround die de EPSON-fabrikant ons ter beschikking heeft gesteld, gebruikt worden. Deze situatie kan voorkomen wanneer de ROS-omgeving een 'Trajectory Stop' bericht heeft geschreven naar de robotcontroller. Dan is namelijk de sequentienummer die over de ethernetverbinding geschreven wordt negatief (meer hierover in de volgende sectie).

3.6 Foutafhandeling

Om de driver robuust te maken, moeten de robotservers adequate foutafhandeling voorzien. Zo moet de driver onverwachte communicatiefouten en onrealistische data op een veilige manier opvangen. Wegens tijdsgebrek werd het opvangen van een plots verbindingsverlies nog niet geïmplementeerd in de EPSON-driver. De servers van de EPSON-driver moeten namelijk automatisch terug een nieuwe verbinding initialiseren (zie specificaties ROS-driver) wanneer bv. de ethernetkabel werd uitgetrokken. Ook is het een vereiste dat de motoren worden afgeschakeld. Wel werd de nodige foutafhandeling voorzien wanneer de trajectory server corrupte data inleest. De EPSON-driver werd zodanig ontwikkeld dat bij het inlezen van JOINT_TRAJECTORY_PT berichten een error wordt gegenereerd wanneer de ingelezen Prefix en/of Header van het ingelezen bericht niet klopt. Hiervoor werd een zelf gedefinieerde error gemaakt. Figuur 21 toont de error code, label en bericht dat gegenereerd wordt door de EPSON-programmeeromgeving wanneer een corrupt bericht over de ethernetverbinding werd ingelezen.

```
nError=8000
sLabel="CORRUPT_MSG"
sMessage="JOINT_TRAJECTORY_PT message is corrupt"
```

Figuur 21: Error code, label en bericht bij een corrupte message

De EPSON-programmeeromgeving stelt een aantal commando's ter beschikking die 'Error Handling' mogelijk maken [16]. Op deze manier is het mogelijk om in het robotprogramma een bepaalde routine uit te voeren wanneer er een error optreedt. Wanneer de EPSON-driver corrupte data inleest zal één van de foutafhandelingen ervoor zorgen dat beide servers worden afgesloten. De gebruiker moet dan zelf opnieuw de verbinding initialiseren door eerst de robotservers en daarna de gestandaardiseerde clients van de ROS-omgeving op te starten. Ook wordt het sequentienummer van de simple message gecontroleerd om te verifiëren of er bv. een 'Trajectory Stop' bericht gegenereerd werd. De jointpositie velden van een 'Trajectory Stop' bericht worden namelijk leeg naar de robotcontroller geschreven. Om te vermijden dat de robot geen onverwachte beweging gaat maken van zijn huidige configuratie naar de HOME-positie van de robot, mag er bij het inlezen van de 'Trajectory Stop' bericht geen robotbeweging uitgevoerd worden. Het robotprogramma moet dus kunnen detecteren of het sequentienummer gelijk is aan -4. Bij de ontwikkeling van de streamende EPSON-driver werd deze implementatie gedeeltelijk uitgevoerd. In de streamende driver werd

namelijk enkel onderscheid gemaakt tussen een negatief of een positief sequentienummer omdat het MSB-probleem voor enkele belemmeringen zal zorgen wanneer dit een negatief getal blijkt te zijn. Dit nummer kan immers meerdere negatieve getallen aannemen zoals te zien in onderstaande figuur. Bij een streamende driver zal enkel de STOP TRAJECTORY sequentienummer mogelijk voorkomen. Vandaar werd bij de streamende driver enkel onderscheid gemaakt tussen negatief en positief.

Val	Name	Description
N		Index into current trajectory
-1	START_TRAJECTORY_DOWNLOAD	Downloading drivers only: signals start
-3	END_TRAJECTORY	Downloading drivers only: signals end
-4	STOP_TRAJECTORY	Driver must abort any currently executing motion

Figuur 22: Verschillende sequentienummers [8]

Bij de (later ontwikkelde) downloadende driver werd wel onderscheid gemaakt tussen de verschillende negatieve sequentienummers. Dit omdat de downloadende driver onderscheid moet kunnen maken tussen bv. De START TRAJECTORY en END TRAJECTORY berichten.
Een bijkomende controle op corrupte of onrealistische data kan geïmplementeerd worden door de ingelezen jointposities te vergelijken met de desbetreffende jointlimits. Indien één van de jointposities niet binnen de limits ligt, hebben we te maken met onrealistische data en kan de trajectory server zodanig geprogrammeerd worden dat er in dit geval geen robotbeweging wordt uitgevoerd. De controle op de ingelezen jointposities werd tijdens de ontwikkeling van de driver nog niet behandeld, maar kan in de toekomst makkelijk geïmplementeerd worden met bv. een aantal IF-functies. Ook lijken andere ROS-drivers voor industriële robots deze controle niet uit te voeren [19]. Dit is dus een extra controle waarmee onrealistische data zeker uitgesloten kan worden.

Om de nodige inspiratie te verwerven omtrent fouthafhandelingen, werd tijdens de ontwikkeling van de EPSON-driver, ook de ABB-driver bestudeerd. Bij de ABB-driver worden namelijk de ingelezen berichten gecontroleerd op hun compleetheid. Deze al bestaande driver doet dit op de volgende manier: allereerst zullen de eerste 4 bytes van het bericht worden ingelezen. Indien nu alles naar behoren werkt, zullen deze 4 bytes, de Prefix voorstellen die de lengte van de simple message aangeeft. Deze ingelezen Prefix wordt dan gebruikt om de rest van de simple message in te lezen. Daarna wordt gecontroleerd of het aantal ingelezen bytes overeenkomt met het aantal dat de Prefix van het bericht aangeeft. Indien het aantal niet overeenkomt, wordt er een error genereerd. De EPSON-driver doet deze controle niet maar zal a.d.h.v. het 'Readbin' commando data blijven inlezen totdat het juist aantal bytes goed werd ontvangen. Indien alle bytes van het bericht werden ingelezen worden pas de volgende regels in de trajectory server geëxecuteerd. Een manier om de controle op compleetheid uit te voeren in SPEL+, moet nog geïmplementeerd/gevonden worden voor de EPSON-driver.

Een andere foutafhandeling die geïntegreerd is in de EPSON-driver is bij de fout die optreedt indien men wenst een pad te plannen met de padplanningssoftware maar de motoren van de robot zijn uitgeschakeld. In dat geval zal het robotprogramma een MessageBox laten verschijnen met de vraag of u de motoren wilt aanzetten of de padplanningsdata wilt inlezen zonder de robotbeweging uit te

voeren. Indien voor de tweede optie gekozen wordt, moeten na het inlezen van alle data, de servers en clients opnieuw gestart worden.

Sectie 3.2.2 geeft aan dat de trajectory server als normale taak werd gedefinieerd in het robotprogramma. Dit betekent dat deze taak beëindigd zal worden wanneer bv. de noodstop wordt ingeduwd. Dit is een beveiliging binnen de EPSON-programmeeromgeving omwille van de robotbewegingen die kunnen uitgevoerd worden in een dergelijke taak. Wanneer nu de trajectory server afgesloten wordt door bv. het induwen van de noodstop, bestaat deze taak niet meer. Dit betekent dat de state server tijdens het opbouwen van de STATUS simple message, de taak waar in de error werd gegenereerd, niet meer kan vinden. Hierdoor wordt er een error gegenereerd in de state server die op de juiste manier behandeld wordt, zodat de ROS-omgeving hiervan op de hoogte kan zijn.

3.7 Besluit

Sectie 3.2 vermeldde dat de driver volledig in het robotprogramma werd ontwikkeld. De logische keuze was dan om de 2 servers in dit zelfde programma te programmeren. Zoals sectie 3.2.2 beschrijft, zullen deze 2 servers simultaan draaien. De ontwikkeling van deze driver is niet vlekkeloos verlopen. Zoals sectie 3.4 aanhaalt zijn er allerlei error's, bugs en fouten opgedoken die het schrijven van de driver aanzienlijk hebben vertraagd. Door middel van de oplossing, beschreven in sectie 3.5.6, konden deze verschillende problemen omzeild worden. Sectie 3.6 beschreef de foutafhandelingen die in de EPSON-driver werden geïntegreerd. Het gaat dan voornamelijk over het juist afhandelen van corrupte berichten, indien deze door de EPSON-driver worden ingelezen. Het juist afhandelen van een communicatieverlies werd nog niet geïmplementeerd in de EPSON-driver.

4 Integratie in MoveIt!

4.1 Inleiding

De vorige hoofdstukken beschreven onder andere hoe de communicatie tussen de robot en de ROS-omgeving verloopt. De EPSON-driver werd zodanig ontwikkeld dat deze JOINT_TRAJECTORY_PT berichten kan inlezen en omzetten naar robotbewegingen. Deze berichten kunnen door de ROS-omgeving niet uit het niets geschreven worden naar de EPSON-driver maar dit moet met een bepaalde padplanningssoftware gebeuren. De padplanningstool *MoveIt!* is een softwareprogramma van de ROS-omgeving die een pad kan plannen en deze kan uitvoeren. Vanaf het moment dat een gepland pad uitgevoerd wordt, zullen JOINT_TRAJECTORY_PT berichten gegenereerd worden die dan naar de EPSON-driver geschreven worden a.d.h.v. de gestandaardiseerde clients. Elk bericht zal dan een gewenste configuratie van de robot bevatten dat bij één trajectpunt van het volledig (geplande) traject hoort. Door het doorlopen van alle berichten zal de robot naar de genereerde configuraties van Moveit! bewegen zodanig dat het volledig traject door de robot wordt uitgevoerd.
Vooraleer men *MoveIt!* kan gebruiken voor het plannen van een pad om dat pad nadien uit te laten voeren door de robot, moeten nog een aantal ROS-packages voor de EPSON C3 A601S robot gemaakt worden. Moveit! heeft namelijk een 'moveit config' package nodig. Voordat men deze package kan aanmaken, moet eerst een ROS-industrial 'robot support' package voor de EPSON C3 A601S robot gemaakt worden. Alle ROS-drivers voor industriële robots stellen deze robotspecifieke packages ter beschikking; voor de EPSON C3 robots bestaan deze packages nog niet dus werden deze tijdens de ontwikkeling van de driver zelf gemaakt. Op deze manier kan de padplanningstool Moveit! met alle informatie van de robot rekening houden zoals joint limits, de links van de robot (botsingsdetectie), de draairichting van de joints, enzovoort.
Sectie 4.2 beschrijft de opbouw van een support package. Alle fabrikantspecifieke support packages worden namelijk volgens dezelfde structuur samengesteld. Hierdoor kunnen ROS-gebruikers makkelijk alle bestanden terugvinden. Ook wordt aangegeven wat zo een package allemaal ter beschikking moet stellen aan de ROS-gebruiker. Sectie 4.3 geeft kort aan hoe door middel van de 'support package' een 'moveit config' package gemaakt kan worden voor de EPSON-robot. Eveneens wordt aangehaald hoe d.m.v. een eenvoudig commando de gestandaardiseerde clients in samenwerking met Moveit! opgestart kunnen worden.

4.2 Support package voor de robot

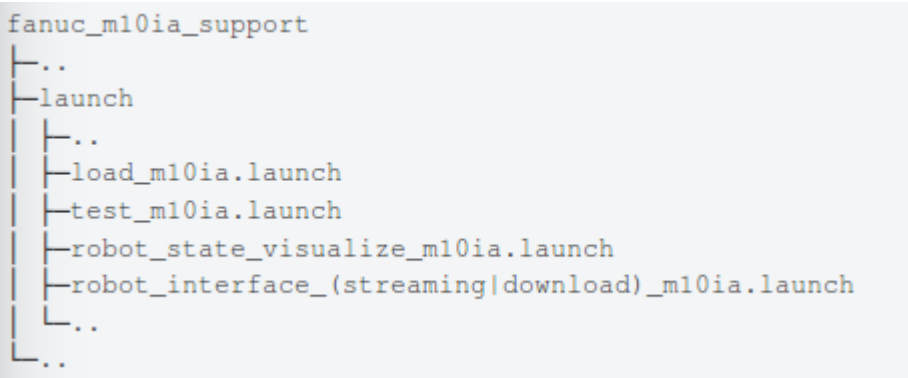
De ROS-industrial robot support package voor een industriële robot wordt onderverdeeld in een aantal mappen. Figuur 23 toont de structuur waaraan een support package moet voldoen bij een industriële robot.

Figuur 23:Onderverdeling robot support package [20]

- De 'config' map bevat bestanden om informatie, inzake de namen van de joints en/of RViz configuraties in op te slaan.
- De 'launch' map bevat een aantal launch files die gebruikelijk zijn bij 'support' packages voor industriële robots (deze sectie zal hier dieper op in gaan).
- In de 'meshes' folder worden de CAD-files die de onderdelen van de robot beschrijven, opgeslagen. Moveit! maakt namelijk gebruik van de visualisatiesoftware Rviz en gebruikt deze 'meshes' om de visualisatie van de robot te realiseren. Aan de hand van Moveit! kan dan de robot visueel naar een gewenste eindconfiguratie verplaatst worden, waardoor naderhand een pad gepland en uitgevoerd kan worden (van begin- tot eindconfiguratie). Ook bevat de 'meshes' folder CAD-files die gebruikt worden om collisiedetectie bij het padplannen mogelijk te maken.
- De 'test' map bevat een file waarmee de 'launch' files uit de 'launch' folder getest kunnen worden. Het gaat dan voornamelijk over het controleren van de launch files voor errors.
- Uiteindelijk is er nog de 'urdf' folder, hierin worden alle urdf- en xacro-files opgeslagen. Dit zijn files van het XML-type die gebruikt worden om het robotmodel voor te stellen. In deze files worden o.a. de meshes gebruikt om het robotmodel op te bouwen en correct te kunnen visualiseren. Ook worden de belangrijke kenmerken van de robot in deze files meegegeven zoals jointlimits, de draairichting van de joints en de maximale snelheid van de joints. Het is namelijk belangrijk dat Moveit! later bij het plannen van robotbewegingen op de hoogte is van deze robotkenmerken. De CAD-files die de tool van de robot beschrijven horen niet in deze urdf/xacro-files thuis. De packages moeten zodanig opgebouwd zijn dat een ROS-gebruiker makkelijk zelf een tool op de robot kan zetten binnen de ROS-omgeving. In de urdf/xacro-files wordt een zogenaamde 'tool0' gedefinieerd waaraan de ROS-gebruiker eventueel een zelfgemaakte tool kan 'plakken'. De ROS-gebruiker moet dan zelf een macro_xacro-file maken die de gewenste tool beschrijft. Met deze nieuwe macro_xacro file kan de tool dan aan het robotmodel 'geplakt' worden. Een voorbeeld kan men vinden bij de Tutorials van ROS-industrial, namelijk 'Working with ROS-Industrial Robot Support Packages. [20]

In sectie 3.4 werd vermeld dat tijdens de ontwikkeling van de driver werd ondervonden dat Moveit! zeer traag is. Eerst en vooral voldeden de files die ACRO ter beschikking stelde niet aan de structuur die figuur 23 aangeeft. Ook bevatte de ter beschikking gestelde urdf-file, code die niet van toepassing is bij het opstellen van een robotmodel voor een industriële robot. De meest significante fout die ervoor zorgde dat Moveit! zo traag was en zo slecht werkte was het feit dat de meshes die gebruikt werden voor de visualisering van de robot ook gebruikt werden voor de collisiedetectie a.d.h.v. Moveit!. Deze meshes hebben een zeer hoge resolutie. Voor de collisiedetectie heeft Moveit! geen meshes nodig met zo een hoge resolutie. Vandaar dat er ook een set van meshes werd gemaakt waarvan de resolutie beduidend werd verlaagd. Op deze manier werkt Moveit! al stukken sneller en beter.

Alle ROS-industrial support packages bevatten de launch files die figuur 23 aangeeft. De figuur toont de launch files voor de industriële M-10a robot van Fanuc. Bij het creëren van de EPSON-packages moeten de namen van deze launch files aangepast worden. Zo zal de load_m10a.launch file bijvoorbeeld de naam load_c3a601s.launch krijgen. Op deze manier kunnen de files van verschillende robots eenvoudig worden onderscheiden.



Figuur 24:Launch files in een support package van een industriële robot

De launch files hebben het volgende doel:

- De 'load' launch file dient om eenvoudig de urdf(of macro_xacro)-file van een specifiek robotmodel te kunnen uploaden naar de robot_description parameter op de parameterserver. De parameterserver is in feite een server die ROS-files/nodes kan gebruiken om parameters in op te slaan of op te vragen.
- Met behup van de 'test' launch file kan de ROS-gebruiker in de visualisatiesoftware RViz, de urdf of macro_xacro-file inspecteren en controleren. De juiste urdf/macro_xacro file wordt automatisch ingeladen d.m.v. de 'load' launch file. Ook worden enkele files binnen de ROS-omgeving gestart die het voor de ROS-gebruiker mogelijk maken om met het robotmodel een interactie uit te uitvoeren. Op deze manier kan men bijvoorbeeld eenvoudig controleren of de draairichting van de joints goed werden ingesteld in de urdf/macro_xacro file.
- Het 'robot_state_visualize' bestand kan gebruikt worden indien de ROS-gebruiker de huidige robotconfiguratie van de werkelijke robot wil controleren in de visualisatiesoftware RViz.

De laatste launch file (robot_interface_streaming of in het geval van een downloadende driver, robot_interface_download) dient om alle vereiste ROS-files te starten die een bi-directionele communicatie tussen de ROS-omgeving en de robotcontroller tot stand kunnen brengen. [20]

Tijdens de ontwikkeling van de EPSON-driver werd de support package voor de EPSON C3 robot (variant: A601S) zelf gemaakt. Dit gaat eenvoudig door alle files van de voorbeeld-package van Fanuc te kopiëren en aan te passen a.d.h.v. de kenmerken van de EPSON-robot. Het zijn dan ook de files die het robotmodel beschrijven (macro_xacro of urdf file), die het meeste tijd in beslag nemen tijdens het maken van de fabrikant specifieke packages. Deze files kunnen getest worden inzake correctheid aan de hand van de 'test' launch file. Op deze manier kunnen de draairichtingen van de robotassen gecontroleerd worden (en de opbouw van het robotmodel zelf natuurlijk). Ook kunnen de coördinaten van de werkelijke robot vergeleken worden met de robotcoördinaten binnen de ROS-omgeving. Deze test kan men uitvoeren met de 'robot_state_visualize' launch file.

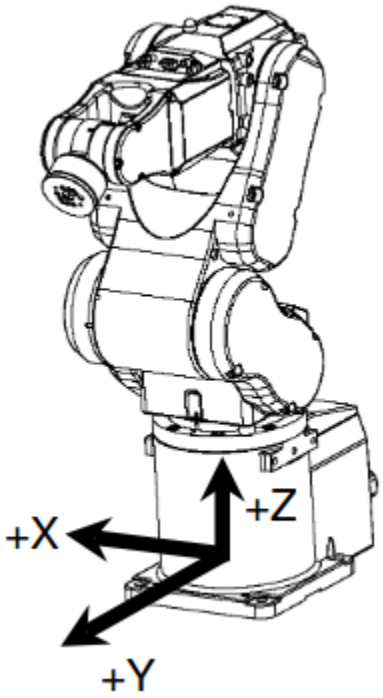
Tijdens het aanpassen van deze files werd er echter geen rekening gehouden met enkele ROS-conventies. ROS heeft namelijk graag dat het coördinatenstelsel van de robot binnen de ROS-omgeving voldoet aan de volgende conventies [21]

- De positieve x-richting moet overeenkomen met een voorwaartse robotbeweging
- De positieve y-richting komt overeen met een beweging naar links
- De positieve z-richting komt overeen met een beweging naar boven.

Het coördinatenstelsel van de EPSON-robot is echter anders. Onderstaande opsomming geeft het huidige coördinatenstelsel weer:

- De positieve x-richting komt overeen met een beweging naar rechts
- De positieve y-richting komt overeen met een voorwaartse beweging
- De positieve z-richting komt overeen met een beweging naar boven.

Ter verduidelijking toont figuur 25 het coördinatenstelsel van de EPSON-robot. Om aan de ROS-conventies te kunnen voldoen, moet de macro_xacro file van het robotmodel nog wat aangepast worden. Zo ligt het coördinatenstelsel binnen de ROS-omgeving -90° verdraaid t.o.v. het werkelijke coördinatenstelsel van de EPSON-robot. Tijdens de ontwikkeling van de driver werden deze aanpassingen nog niet gemaakt omdat de EPSON-driver ook zonder het maken van deze aanpassingen kan werken, om alles mooi in orde te maken moet dit wel nog gebeuren.



Figuur 25: Coördinatenstelsel EPSON-robot [22]

4.3 MoveIt! config package voor de robot

Eenmaal de support package voor de EPSON C3 robot klaar is en de cruciale robotkenmerken binnen de ROS-omgeving werden gecontroleerd, kan een 'moveit config' package gegenereerd worden. Deze package voorziet de nodige files om met Moveit! botsingsvrije padplanning voor de EPSON C3 robot mogelijk te maken. De 'moveit config' package kan eenvoudig gegenereerd worden door de zogenaamde 'moveit setup assistant'. De 'moveit setup assistant' is een grafische gebruikersinterface om een robot te configureren zodanig dat deze gebruikt kan worden met Moveit!. [23]. Binnen de gebruikersinterface van de setup assistant kan ook een zelf-collisie matrix gegenereerd worden. De zelf-collisie matrix generator zoekt paren van robotlinks waar MoveIt! geen collisie controle op hoeft te doen (bv. aangrenzende links, links die nooit met elkaar kunnen botsen, …). Aan de hand van de zelf-collisie matrix heeft MoveIt! minder tijd nodig om een volledig/botsingsvrij traject te plannen [24]. Ook kan een virtuele joint toegevoegd worden die zal fungeren als de verbinding tussen de robot met de 'wereld'. Voor een industriële robot betekent dit dat de voet van de robot vastgemaakt moet worden aan de grond, met andere woorden is de virtuele joint van het type 'fixed'. In een volgende stap moet een planningsgroep voor de arm van de robot toegevoegd worden, hierbij hoort een kinematische solver die binnen de gebruikersinterface gekozen kan worden. In het geval van de EPSON-robot werd gekozen voor de TRAC-IK solver. De KDL solver die standaard gebruikt wordt, werkt minder goed in de aanwezigheid van joint-limits. [25] Door de setup assistant te beëindigen kan de 'moveit config' package voor de EPSON c3 robot gegenereerd worden. Vanaf dit moment kan MoveIt! a.d.h.v. de zojuist gemaakte package, trajecten plannen en controle op collisie uitvoeren. De 'moveit config' package bevat echter nog niet genoeg informatie om met een werkelijke robot een interactie uit te voeren, hiervoor moeten nog enkele aanpassingen gemaakt worden [26]. Er moet namelijk voor gezorgd worden dat MoveIt! kan communiceren met de robotcontroller. Hoe dit gerealiseerd kan worden, beschrijft één van de ROS-industrial tutorials, namelijk 'Create a Moveit Package for an Industrial Robot' [26]. Wanneer de tutorial volledig werd doorlopen kan MoveIt! in samenwerking met de gestandaardiseerde clients opgestart worden met het volgende commando:

```
$ roslaunch <robot_name>_moveit_config moveit_planning_execution.launch sim:=false robot_
ip:=<robot's IP>
```

Figuur 26: Commando gestandaardiseerde clients

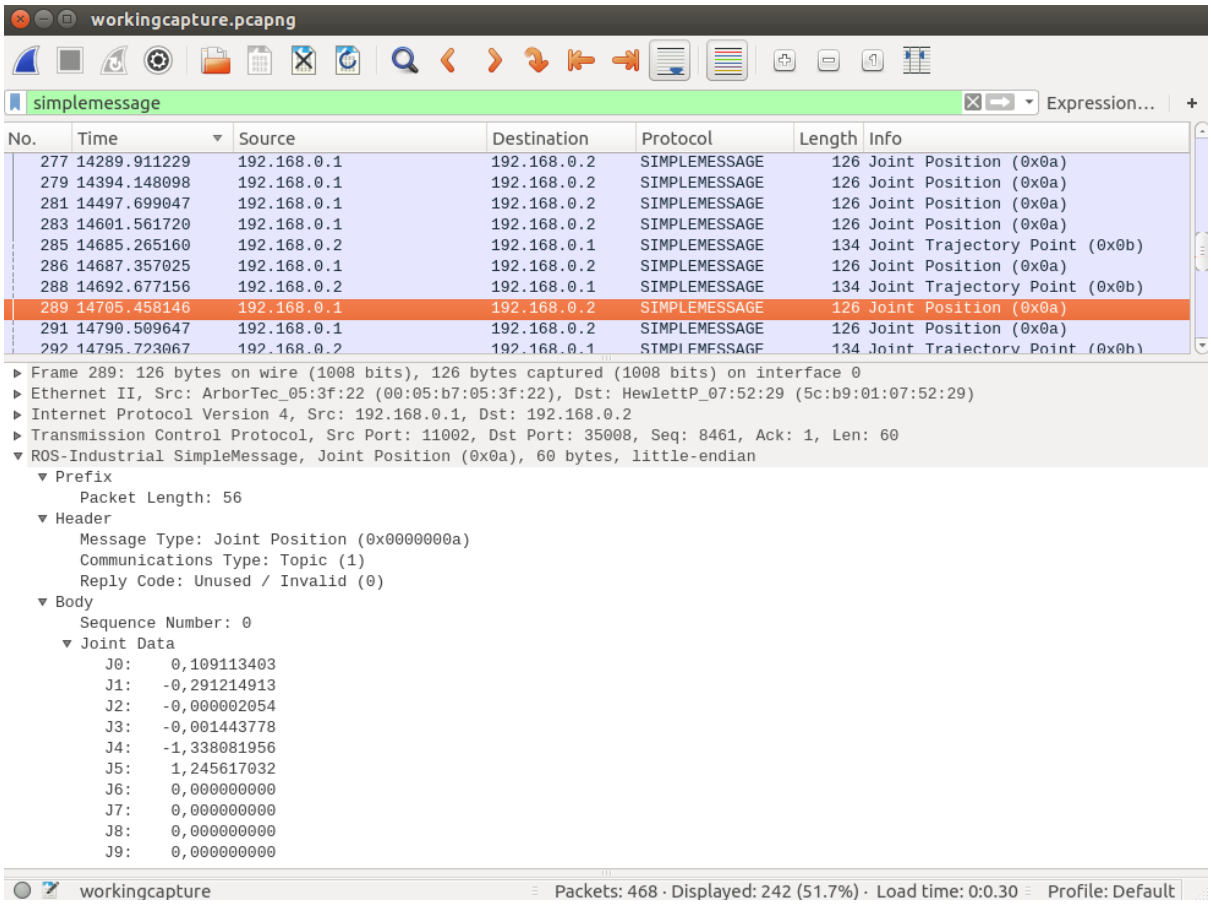
4.4 Besluit

In sectie 4.2 werd vermeld dat gedurende de ontwikkeling van de driver Moveit! zeer traag was. De eerste reden hiervoor was dat de 'support' en 'moveit_config' package niet helemaal volgens de richtlijnen van de ROS-community werden opgesteld. Ook waren de ter beschikking gestelde files te ACRO onoverzichtelijk en zaten vol met overbodige bestanden. De laatste reden die zorgde voor de trage Moveit! was dat de meshes die gebruikt werden voor de collisiedetectie, een veel te grote resolutie hadden. Na het volgen van deze richtlijnen, het opruimen van de overbodige bestanden en het verlagen van de resolutie van de meshes, kan besloten worden dat Moveit! aanzienlijk sneller opstart en soepeler werkt. Ook wordt nu de mapstructuur gebruikt die door de hele ROS-Industrial community wordt toegepast. Zo is het makkelijk voor andere ontwikkelaars om bestanden terug te vinden.

5 Experimentele resultaten

5.1 Inleiding

Tijdens de ontwikkeling van de EPSON-driver is het handig om het programma Wireshark te hanteren. Wireshark is een tool die netwerkverkeer kan analyseren. [27]Op deze manier is het bijvoorbeeld mogelijk om te controleren of de robotcontroller op de juiste manier het simple message protocol volgt tijdens het schrijven van berichten naar de ROS-omgeving. Ook kunnen de simple messages, gegenereerd door de gestandaardiseerde ROS-clients (JOINT_TRAJECTORY_PT berichten), geanalyseerd worden. Hierdoor is bij de ontwikkeling van de driver duidelijk hoe de JOINT_TRAJECTORY_PT berichten precies zijn opgebouwd. Zo kan de ontwikkelaar er voor zorgen dat de ingelezen berichten op de correcte manier worden ontleedt in de trajectory server. Ook wordt met Wireshark meteen duidelijk hoe de communicatie tussen de ROS-omgeving en de driver precies verloopt wat de ontwikkeling van de EPSON-driver aanzienlijk vergemakkelijkt.
Om de analyse van de EPSON-driver d.m.v. Wireshark beduidend te vereenvoudigen, stelt ROS-industrial een zogenaamde 'Wireshark Lua dissector' voor het simple message protocol ter beschikking. [28] Door middel van deze 'dissector' kan Wireshark berichten die aan het simple message protocol voldoen en over het netwerk worden geschreven, automatisch decoderen. Op deze manier kan zeer eenvoudig het netwerkverkeer tussen de EPSON-driver en de ROS-omgeving geanalyseerd worden. Figuur 27 toont een voorbeeld waar Wireshark m.b.v. de simple message dissector de communicatie tussen de ROS-omgeving en de EPSON-driver analyseert. Het bovenste venster uit de Wireshark-werkomgeving toont alle simple messages die de tool heeft opgenomen tijdens het luisteren naar het netwerkverkeer tussen de ROS-omgeving en robotcontroller. Zoals de figuur aangeeft, worden JOINT_POSITION simple messages van de robotcontroller (ip: 192.168.0.1) naar de ROS-omgeving (ip: 192.168.0.2) geschreven. Ook kan men meteen zien dat de JOINT_TRAJECTORY_PT berichten van de ROS-omgeving naar de EPSON-driver geschreven worden. Wireshark wijst ook meteen een 'timestamp' toe aan de berichten die door deze tool worden opgevangen. Op deze manier is het eenvoudig om het netwerkverkeer i.f.v. de tijd te analyseren. Sectie 5.2 beschrijft deze analyse.
Door op één van de berichten uit het bovenste venster van de Wireshark-omgeving te klikken, geeft het onderste venster de data van het bericht in kwestie weer . Figuur 27 toont dat er op een JOINT_POSITION bericht werd geklikt. Wireshark decodeert dit bericht en toont alle velden van de simple message volgens de juiste datatypen. Zo kan men in Wireshark o.a. de werkelijke robotconfiguratie (jointposities als floating points) op een bepaald tijdstip bestuderen. Hieruit volgt dat de robotbeweging i.f.v. de tijd eenvoudig geanalyseerd kan worden. Sectie 5.3 behandelt deze analyse.



Figuur 27: Wireshark capture van de simple messages die over het netwerk worden geschreven.

Tijdens deze masterproef werd ontdekt dat de simple message dissector die ROS-industrial ter beschikking stelt niet werkt wanneer de berichten byte per byte over het netwerk worden geschreven. Dit probleem werd tijdens de ontwikkeling van de EPSON-driver gemeld aan de ROS-industrial community. Deze bug werd ondertussen behandeld door ROS-industrial.

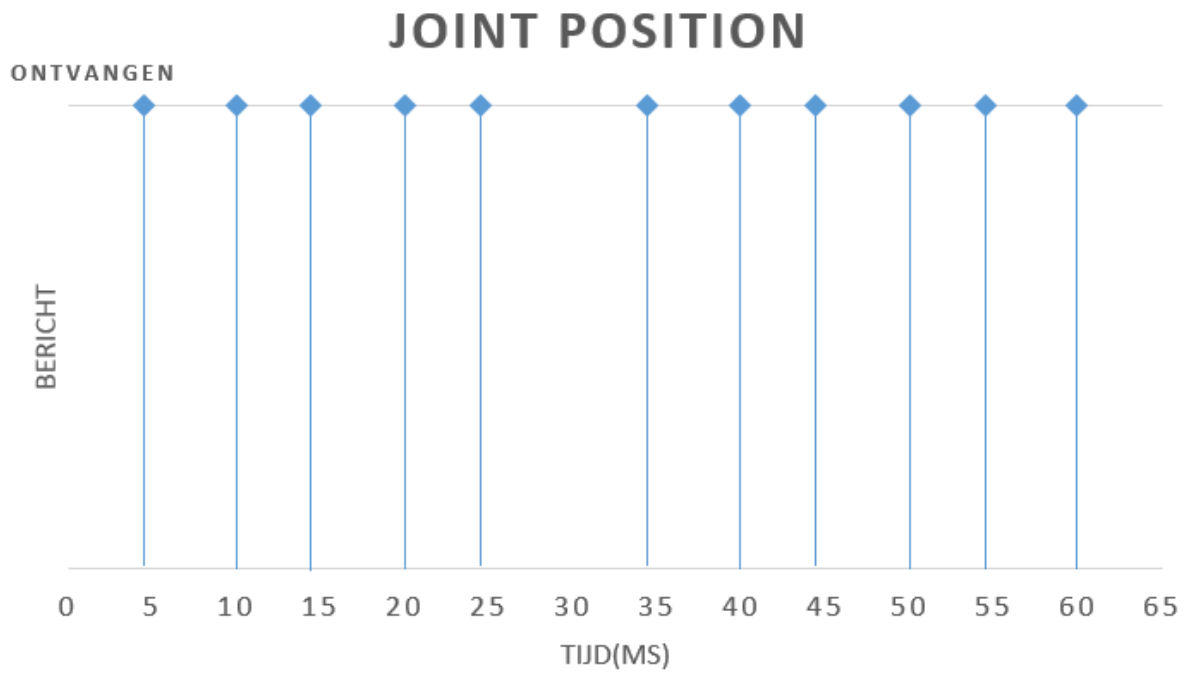
5.2 Analyse van het netwerkverkeer

Deze sectie beschrijft het netwerkverkeer tussen de EPSON-driver en de ROS-omgeving. Aan de hand van deze sectie kunnen eenvoudig de prestaties van de ontwikkelde EPSON-driver geanalyseerd worden. Sectie 3.2.2 gaf aan dat de EPSON-driver a.d.h.v. twee robotservers communiceert met de gestandaardiseerde ROS-clients. Deze sectie zal door middel van de verworven data van Wireshark de prestatie van deze twee robotservers evalueren.

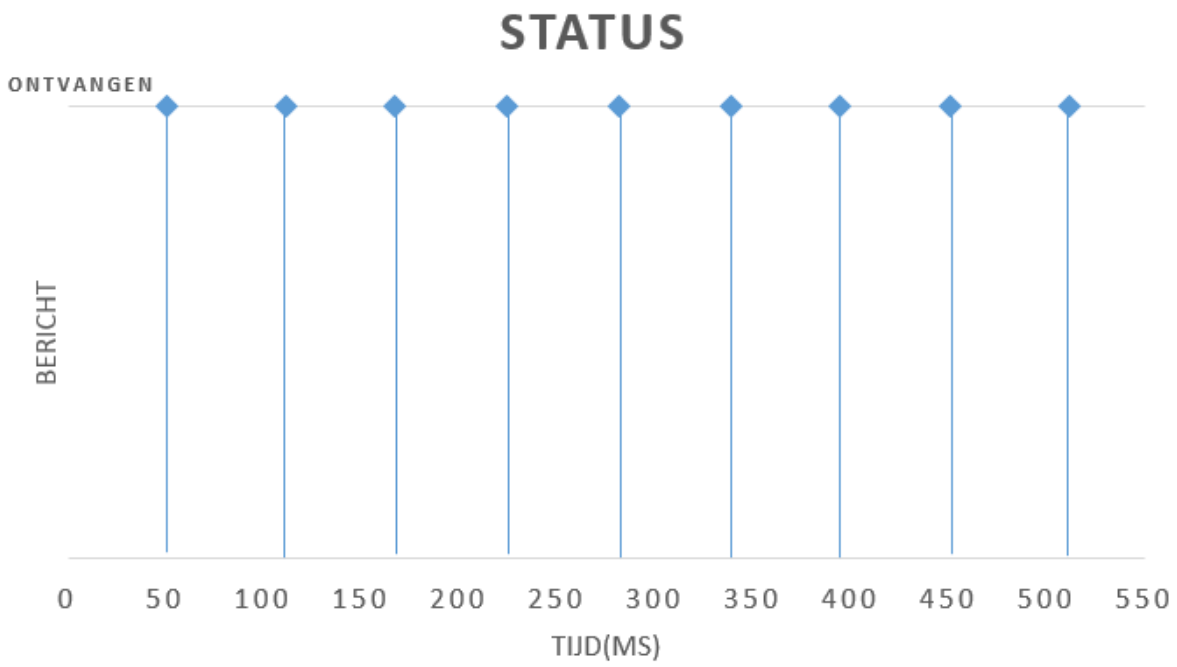
Eerst zal de ontwikkelde state server geëvalueerd worden. Zoals sectie 3.2.2 beschrijft, zal de state server STATUS en JOINT_POSITION berichten naar de ROS-omgeving schrijven. Voor de verduidelijking werd het netwerkverkeer voor elk soort simple message afzonderlijk bestudeerd. De simple message die als eerst aan bod komt, is het JOINT_POSITION bericht. Figuur 28 toont dat de EPSON-driver binnen de 60 milliseconden 11 JOINT_POSITION berichten naar de ROS-omgeving

heeft geschreven. Dit geldt enkel wanneer in de state server van de EPSON-driver geen 'delays' werden geprogrammeerd. Met andere woorden is dit de maximale snelheid dat het robotprogramma toelaat. Wanneer er dieper wordt ingegaan op figuur 28 kan verondersteld worden dat er om de 5 milliseconden een JOINT_POSITION bericht naar de ROS-omgeving wordt geschreven. Een uitzondering bevindt zich tussen het 25 en 35 milliseconden interval. Hier duurde het 10 milliseconden vooraleer een JOINT_POSITION bericht naar ROS werd geschreven. De reden hiervoor luidt als volgt:
De EPSON-driver werd zodanig ontwikkeld dat de state server om de 10 JOINT_POSITION berichten, één STATUS bericht naar de ROS-omgeving schrijft om ROS de toestand van enkele kritische parameters ter beschikking te stellen. De state server heeft namelijk een bepaalde tijd (+/- 5ms) nodig om het STATUS bericht weg te schrijven waardoor het robotprogramma het JOINT_POSITION bericht even niet kan opbouwen en wegschrijven. Al bij al wordt dit bericht tegen een voldoende hoge snelheid naar ROS geschreven. Hierdoor is de ROS-omgeving letterlijk constant op de hoogte van de werkelijke robotconfiguratie en kan MoveIt! de gemaakte robotbeweging controleren a.d.h.v. het geplande traject

Het STATUS bericht dat door de EPSON-driver naar de ROS-omgeving wordt geschreven gebeurt volgens figuur 29. Zoals de figuur aangeeft schrijft de EPSON-driver binnen de 550 milliseconden 9 STATUS simple messages. De EPSON-driver heeft namelijk iets meer dan 55 milliseconden nodig om dit bericht naar de ROS-omgeving te schrijven. (wachten op 10 JOINT_POSITION berichten komt overeen met 50 ms plus de 5 milliseconden dat het robotprogramma nodig heeft om het statusbericht zelf naar ROS te schrijven is 55 ms).

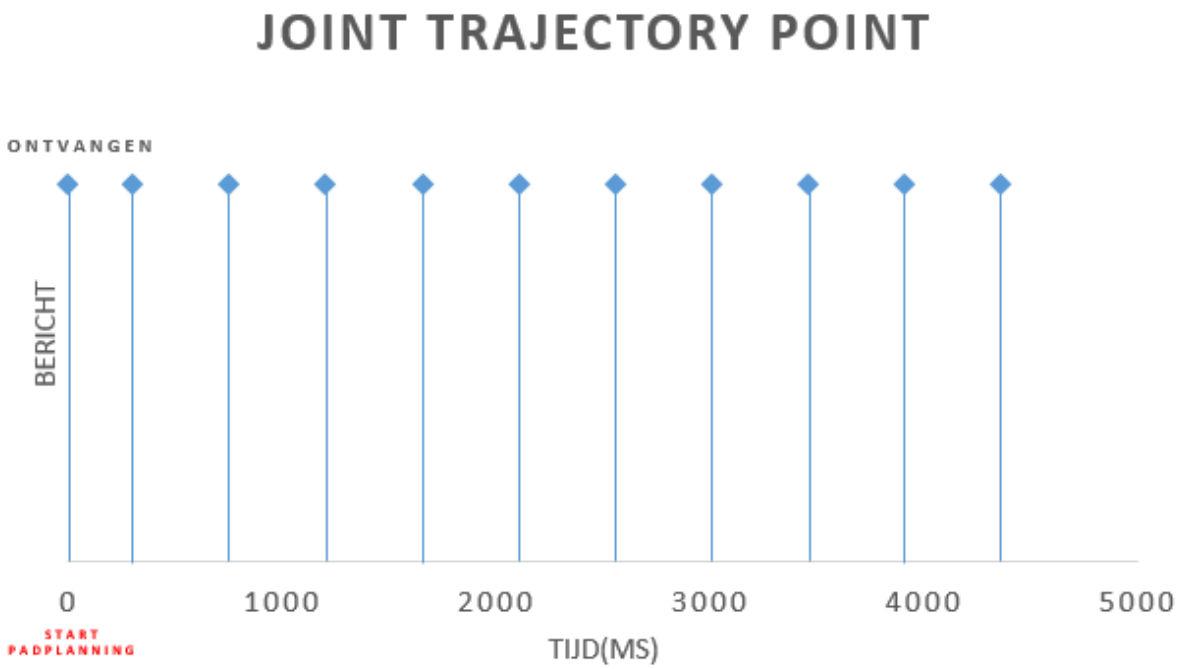


Figuur 28: Snelheid netwerkverkeer Joint position simple messages



Figuur 29: Snelheid netwerkverkeer status simple messages

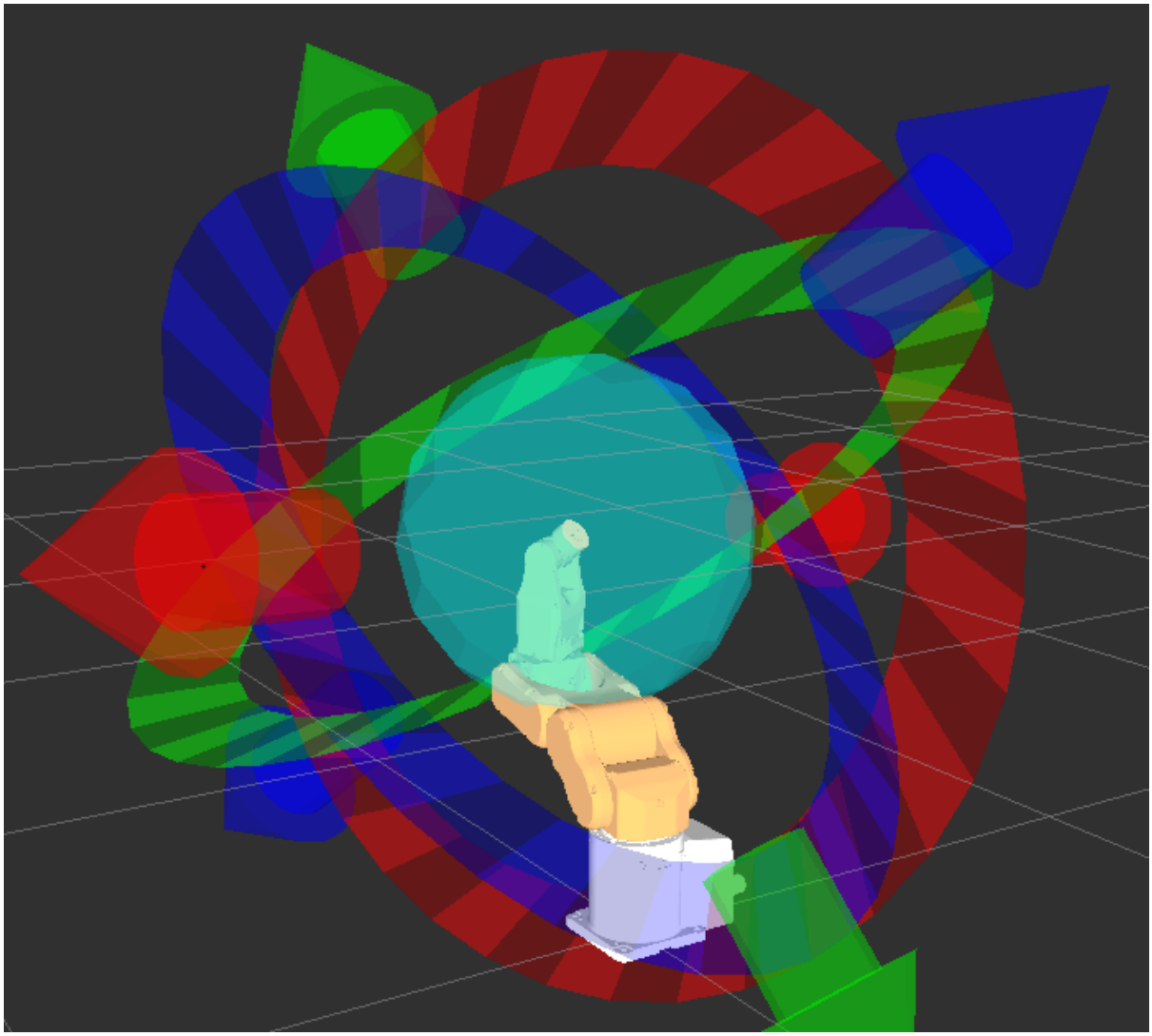
Figuur 30 beschrijft de snelheid waarmee de JOINT_TRAJECTORY_PT berichten door de EPSON-driver worden ingelezen. Zoals de figuur aangeeft, worden er binnen 4,5 seconden 11 JOINT_TRAJECTORY_PT berichten door de EPSON-driver ontvangen. Deze snelheid ligt dus beduidend lager dan de vorige berichten die in deze sectie werden besproken. De reden hiervoor luidt als volgt:
De 'streamende' driver werd zodanig ontwikkeld dat per bericht dat wordt ingelezen, ook meteen een robotbeweging volgt. Dit betekent dat de robot meteen een beweging zal uitvoeren naar het trajectpunt dat het pas ingelezen bericht beschrijft. Met andere woorden moet de EPSON-driver wachten totdat de robotbeweging is uitgevoerd vooraleer een nieuw bericht komende van de ROS-omgeving ingelezen kan worden. Met deze veronderstelling in het achterhoofd kan men uit de grafiek ook meteen afleiden dat de EPSON-driver zodanig werd ontwikkeld dat de robot ongeveer een halve seconde nodig heeft om naar een volgend trajectpunt te bewegen. De gewenste acceleratie, deceleratie en maximale snelheid van de robotbeweging werden hard gecodeerd in het robotprogramma ingesteld, Door deze instellingen aan te passen zullen deze berichten tegen een andere snelheid ingelezen kunnen worden. Het tijdsinterval tussen het eerste en het tweede bericht lijkt ook beduidend lager te zijn dan de andere intervallen. Dit komt omdat het eerste JOINT_TRAJECTORY_PT bericht dat werd ingelezen de huidige robotconfiguratie als data bevat. Hierdoor hoeft de robotbeweging geen beweging te maken waardoor het volgende bericht sneller ingelezen kan worden.



Figuur 30: Snelheid netwerkverkeer trajectory pt simple messages

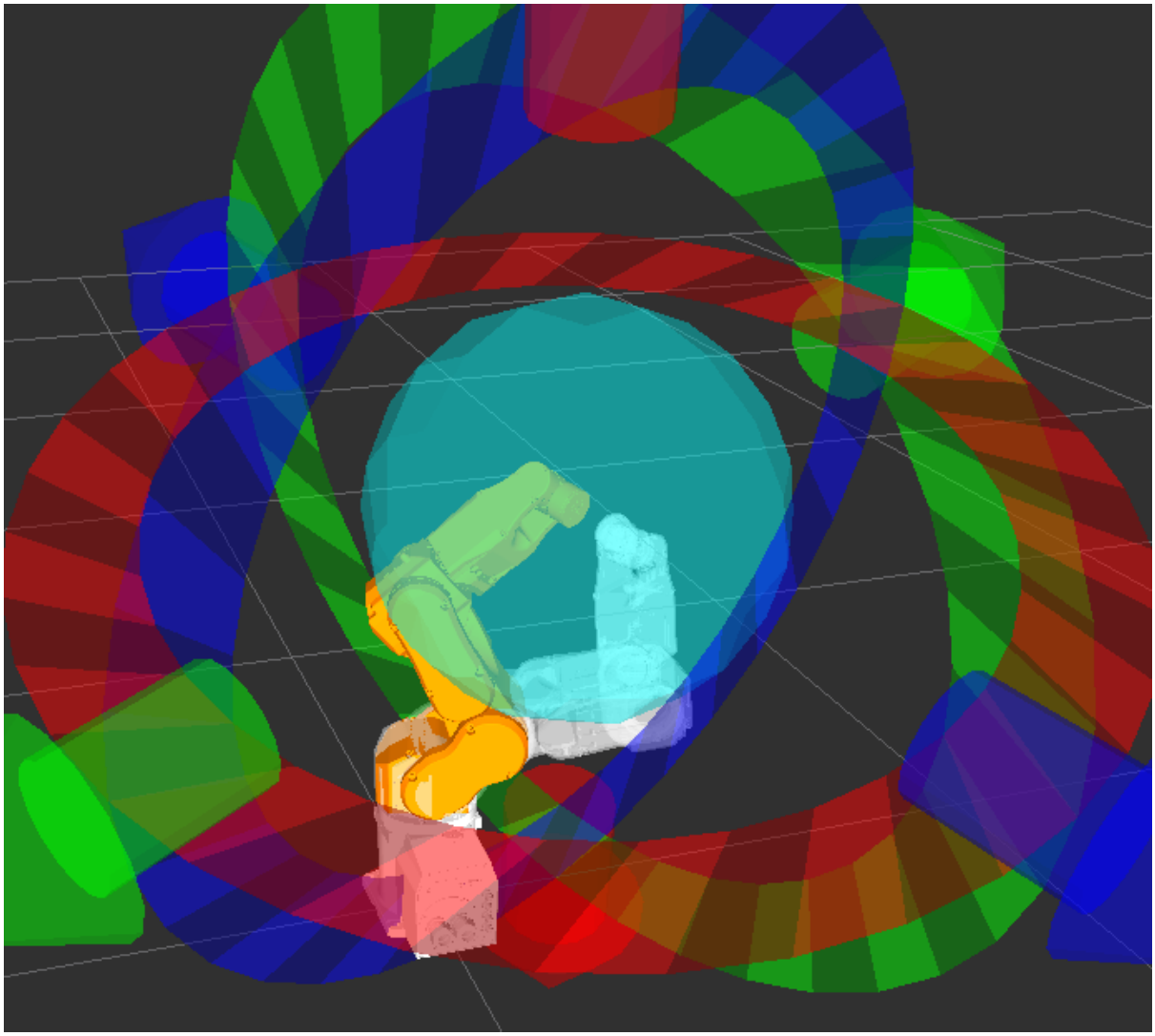
5.3 Analyse van de robotbeweging

De analyse van de robotbeweging aan de hand van Wireshark gebeurt door de JOINT_POSITION simple messages i.f.v. de tijd te bestuderen. Omdat de werkelijke robotconfiguratiedata (jointposities) van deze berichten uiteindelijk door de 'joint_state_publisher' in de ROS-omgeving worden gepubliceerd, is het eenvoudiger om de analyse van de robotbeweging binnen de ROS-omgeving uit te voeren. Dit omdat op deze manier de werkelijke jointposities van de robot i.f.v. de tijd eenvoudig geplot kunnen worden a.d.h.v. de 'rqt_plot' tool.
Om de robotbeweging te analyseren werd met MoveIt! een traject gepland. De beginconfiguratie van de robot wordt d.m.v. de JOINT_POSITION simple messages aan MoveIt! ter beschikking gesteld. Figuur 31 toont de beginconfiguratie van de robot.



Figuur 31: Beginconfiguratie van de robot bij de robotbewegingsanalyse

Wanneer MoveIt! volledig werd opgestart en de werkelijke robotconfiguratie door MoveIt! gevisualiseerd werd zoals figuur 31 aangeeft, kan binnen de visualisatieomgeving van MoveIt!, met een simpele muisklik, de robot naar een gewenste eindconfiguratie gesleept worden. Figuur 32 toont de gewenste eindconfiguratie van de robot die gekozen werd tijdens de analyse van de robotbeweging. De robotconfiguratie die in het wit is aangeduid, is de beginconfiguratie van de robot en komt overeen met de configuratie uit figuur 31. De oranje robotconfiguratie is de gewenste (ingestelde) robotconfiguratie waar de robot naar toe zal bewegen tijdens de analyse van de robotbeweging.



Figuur 32: Eindconfiguratie van de robot bij de robotbewegingsanalyse

Door in MoveIt! nu een traject te plannen kan men binnen de visualisatieomgeving zien hoe de robot zal bewegen indien de robot het geplande traject zou uitvoeren. Dit geplande traject kan door de robot uitgevoerd worden door eenvoudig op de 'Execute' button in de Moveit-omgeving te klikken. Figuur 33 toont het verloop van de robotbeweging i.f.v. de tijd. De werkelijke jointposities van de robot werden i.f.v. de tijd geplot a.d.h.v. de 'rqt_plot' tool die de ROS-omgeving ter beschikking stelt. De jointposities van de robot worden namelijk d.m.v. JOINT_POSITION simple messages naar de ROS-omgeving geschreven. De 'joint_states' topic stelt deze data dan ter beschikking binnen de ROS-omgeving onder de vorm van een array die bij element '0' begint. Dat element bevat dan de hoekpositie (in radialen) van de eerste as van de EPSON-robot. Element 1 zal dan de hoekpositie van de tweede as aanduiden enzovoort. Door de specificaties van de array in rekening te nemen, kan eenvoudig uit figuur 33 het verloop van alle jointposities i.f.v. de tijd geëvalueerd worden a.d.h.v. de legende.

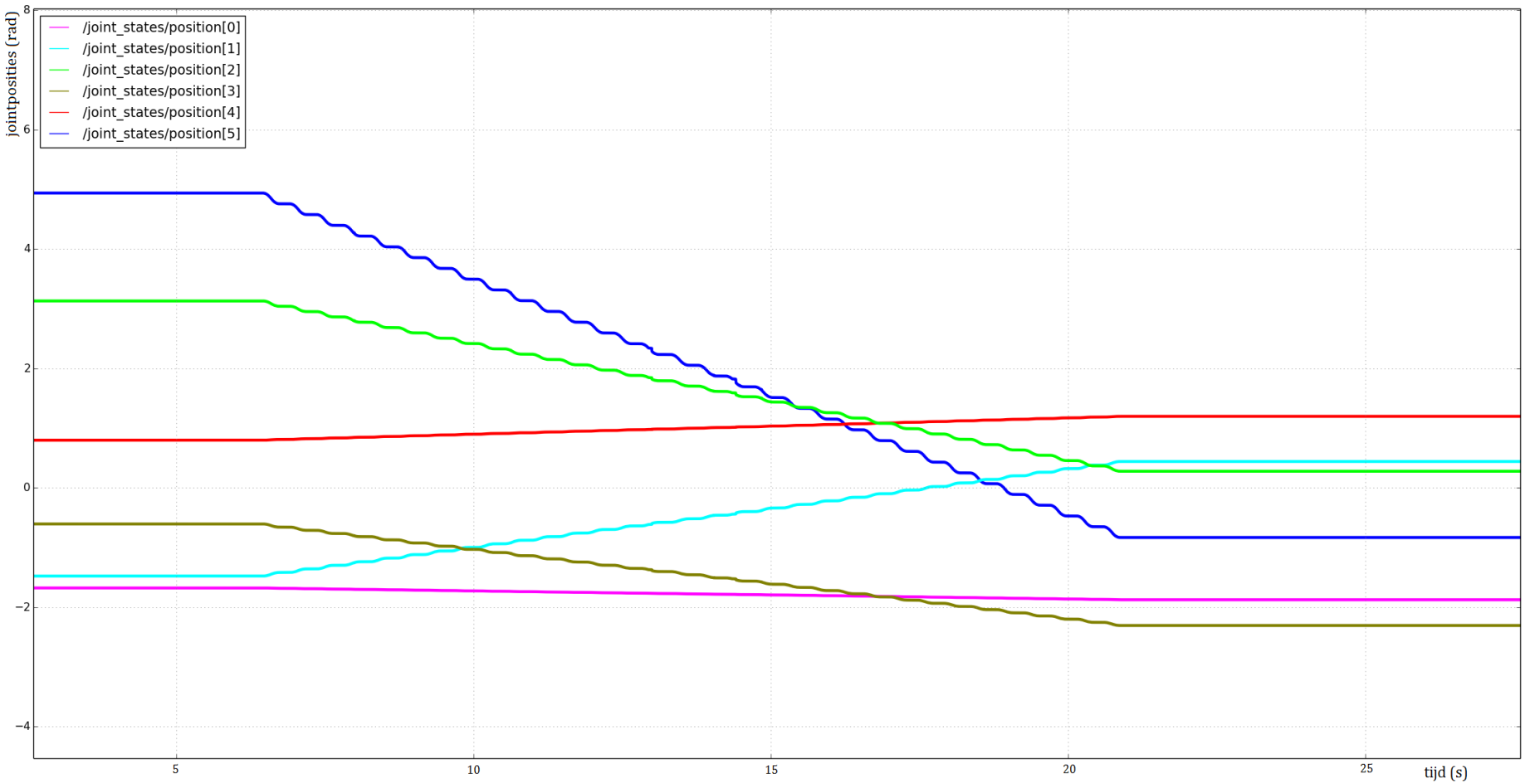
Aan de hand van figuur 33 kan al meteen opgemerkt worden dat de robotjoints in kwestie een schokkende beweging maken om de gewenste eindconfiguratie te behalen. Omdat de EPSON-driver volgens de streaminginterface werkt zal deze steeds een bericht inlezen en daarna direct een robotbeweging uitvoeren naar het trajectpunt dat het ingelezen bericht beschrijft. Hieruit volgt dat de robotsturing geen weet heeft van de robotconfiguratie die het volgend punt uit het traject

beschrijft. Met andere woorden kent de robotsturing niet het volgend punt van het te doorlopen traject tijdens het bewegen naar het daarnet ingelezen trajectpunt. Hierdoor is een vloeiende beweging bij 'streamende' ROS-drivers voor een industriële robot wat moeilijker te realiseren dan bij 'downloadende' drivers. Een downloadende driver leest alle berichten op voorhand in en voert daarna pas de beweging uit die met MoveIt! werd gedefinieerd. Hierdoor kent de robotsturing op voorhand alle robotconfiguraties beschreven door de trajectpunten van het volledig traject zodat het geplande pad op voorhand 'geteached' kan worden d.m.v. het gepaste robotcommando. Hierdoor is een vloeiende beweging door middel van een downloadende driver eenvoudiger te realiseren.

Een streamende driver heeft binnen de ROS-industrial community toch de voorkeur. De meest voor de hand liggende reden is dat de robot meteen kan bewegen na het ontvangen van een commando. Tot nu toe is ROS-Industrial er in geslaagd om voor de meeste drivers een redelijk vloeiende beweging van de robot te realiseren door het toepassen van allerlei robotspecifieke handigheden. Ook bestaat meestal de mogelijkheid om bij een streamende driver met een kleine buffer te werken, waardoor de latency van het netwerk minder een rol gaat spelen. Al bij al zorgen streamende drivers vaak voor een beduidend hogere prestatie t.o.v. de downloadende versies. De downloadende drivers kampen namelijk met de volgende nadelen:

- Een welbepaalde tijd moet doorlopen zijn vooraleer de robot zijn beweging kan starten
- De lengte van het geplande traject is gelimiteerd. De robotcontroller kan namelijk maar een bepaalde hoeveelheid configuraties 'teachen' binnen een robotproject/programma. Dit betekent dus een downloadende driver goed kan werken bij eenvoudige bewegingen (zonder al te veel obstakels). Indien de driver wordt gebruikt voor wat lastigere trajecten kan het goed zijn dat de limiet overschreden wordt.

Een streamende driver krijgt vanwege bovenstaande redenen de voorkeur binnen de ROS-industrial community.



Figuur 33: Verloop robotbeweging i.f.v. de tijd

5.4 Analyse van de dode tijd bij een downloadende driver

In de vorige hoofdstukken werd altijd vermeld dat de ontwikkelde EPSON-driver aan de hand van de streaminginterface werkt. Er werd echter ook een downloadende versie van de EPSON-driver gemaakt. Deze werd ontwikkeld om te controleren hoeveel dode tijd het systeem bevat. Met andere woorden werd aan de hand van de downloadende driver geanalyseerd hoeveel tijd deze nodig heeft om alle berichten in te lezen om zo het geplande traject te kunnen uitvoeren. Om dit te realiseren werden natuurlijk een downloadende versie van de 'moveit config' en 'support' packages gemaakt. Op deze manier kan de ROS-omgeving een 'Trajectory Start' en een 'Trajectory End' bericht naar de EPSON-driver schrijven. Hierdoor kan de driver op een eenvoudige manier te weten komen wanneer het volledig traject werd ingelezen. Op het moment dat het volledig traject werd ingelezen start de robotbeweging.
De analyse van de dode tijd werd uitgevoerd door timers in het robotprogramma te gebruiken. Op deze manier kan men eenvoudig de tijd meten die nodig is om één bericht van de ROS-omgeving in te lezen. De robotbeweging zal niet meer het inlezen van berichten verlangzamen omdat we nu gebruik maken van de downloadende driver. Ook werden alle mogelijke delays uit het robotprogramma gelaten om de maximale prestatie van de driver te kunnen opmeten.
Uit de analyse van de dode tijd bleek dat men voor elk ingelezen bericht een gemiddelde van 11 à 12 milliseconden moet rekenen. Zoals ondertussen wel duidelijk is, geeft één bericht de robotconfiguratiedata weer van één trajectpunt uit het geplande traject. Hieruit volgt dat de dode tijd grotendeels afhankelijk is van de grootte/moeilijkheidsgraad van het geplande traject. Wanneer het geplande traject van MoveIt! bijvoorbeeld 1000 genereerde trajectpunten bevat, dan zal men een dode tijd van 11 à 12 seconden bekomen, wat zeker niet als verwaarloosbaar beschouwd kan worden. Bij een eenvoudige pick and place opstelling zoals bij het random bin picking systeem te ACRO, is het geplande traject meestal eenvoudig en niet groot. Hierdoor zal de dode tijd beduidend verlagen en kan de robotbeweging na een dode tijd van één seconde meestal al gestart worden.
Zoals in de vorige paragraaf werd vermeld moet men 11 à 12 milliseconden per ingelezen bericht rekenen. Dit lijkt op het eerst zicht nog al veel ten opzichte van de snelheid dat de state server JOINT_POSITION berichten naar de ROS-omgeving schrijft. De reden hiervoor luidt als volgt:
Bij elk bericht dat de trajectory server inleest, moet ook door deze server een service reply naar de ROS-omgeving geschreven worden om zo duidelijk te maken dat het bericht goed werd ontvangen. Zo een service reply bericht is meestal een lege JOINT_POSITION simple message waarvan de header juist is opgebouwd. Hierdoor verliest men in feite al 5 milliseconden.

5.5 Besluit

In sectie 5.3 werd vermeld dat bij industriële toepassingen vaak de voorkeur gaat naar een streamende driver, dit omwille van het feit dat de robot direct zijn beweging kan starten na het ontvangen van een commando. Het zal echter wel moeilijker zijn om met een streamende driver een vloeiende beweging te verkrijgen waardoor er nog zeker ruimte voor verbetering is bij deze soort drivers. Indien men toch een vloeiende beweging wenst te verkrijgen, kan er nog altijd worden overgeschakeld naar een downloadende driver. Bij deze soort drivers is een vloeiende beweging makkelijker te verkrijgen maar dat brengt echter ook enkele nadelen met zich mee (zie sectie 5.3).

In sectie 5.2 werd beschreven hoe het netwerkverkeer werd geanalyseerd d.m.v. de handige tool Wireshark. Aan de hand van de gemaakte grafieken uit deze sectie, kan geconcludeerd worden dat de verschillende simple messages met een voldoende hoge snelheid door de ROS-omgeving naar de EPSON-controller worden gestuurd (en omgekeerd). Sectie 5.4 beschreef de dode tijd die zal optreden bij een downloadende EPSON-driver. De resultaten zijn schappelijk voor kleine trajecten maar bij ingewikkeldere trajecten zal de dode tijd een te grote rol spelen.

5.6 Toekomstig werk

Tijdens deze masterproef werd de ROS-driver voor de EPSON C3 A601S afgewerkt voor random bin picking systemen. Om in de toekomst deze driver online te kunnen publiceren in de ROS-community, moeten echter nog verschillende dingen worden aangepast of verbeterd. Onderstaande opsomming geeft de belangrijkste aandachtspunten weer:

- Communicatieverlies tussen de EPSON-driver en de ROS-omgeving op de juiste manier behandelen
- Een vloeiende beweging bekomen met de streamende driver.
- De ROS driver laten voldoen aan de conventies die ROS voorlegt.

Buiten de bovenstaande opsomming zijn er natuurlijk nog kleine aanpassingen die gemaakt kunnen worden omtrent het detecteren van corrupte data of de controle op de lengte van ingelezen berichten. Om de ontwikkelde EPSON-driver te kunnen gebruiken in de random bin picking opstelling van ACRO kunnen ook best alle obstakels geïntegreerd worden in MoveIt!. Ook moet de grijpconfiguratie komende van Halcon aan het padplanningspakket nog ter beschikking gesteld worden.

Literatuurlijst

[1] J. Beuls, „Experimentele evaluatie van botsingsvrije trajectgeneratie voor 3D random bin picking".

[2] „ROS MoveIt!," [Online]. Available: moveit.ros.org.

[3] „ROS wiki Homepage," [Online]. Available: http://wiki.ros.org.

[4] „ROS wiki Industrial," [Online]. Available: http://wiki.ros.org/Industrial.

[5] „ROS industrial description," [Online]. Available: http://rosindustrial.org/about/description/.

[6] „ROS wiki Packages," [Online]. Available: http://wiki.ros.org/Packages.

[7] „ROS wiki Simple Message," [Online]. Available: http://wiki.ros.org/simple_message.

[8] „Github Gijs van der Hoorn REP simple message," [Online]. Available: https://github.com/gavanderhoorn/rep-ros-i/blob/retrospective_simple_msg/rep-ixxxx.rst.

[9] „ROS wiki Industrial Robot Client," [Online]. Available: http://wiki.ros.org/industrial_robot_client.

[10] „ROS wiki Industrial Robot Client Generic Implementation," [Online]. Available: http://wiki.ros.org/industrial_robot_client/generic_implementation.

[11] „ROS answers Port Questions," [Online]. Available: http://answers.ros.org/question/198229/port-questions-and-initial-feedback-from-controller-via-simple-message-ros-i/.

[12] „ROS wiki Industrial Robot Driver Specifications," [Online]. Available: http://wiki.ros.org/Industrial/Industrial_Robot_Driver_Spec.

[13] „ROS wiki Industrial Supported Hardware," [Online]. Available: http://wiki.ros.org/Industrial/supported_hardware.

[14] „Github ROS industrial motoman 93," [Online]. Available: https://github.com/ros-industrial/motoman/pull/93.

[15] S. E. CORPORATION, SPEL+ Language Reference RC+ 6.0 Ver.6.2, 2012.

[16] S. E. CORPORATION, User's Guide RC+ 6.0 Ver.6.2, 2011.

[17] „Github Gijs van der Hoorn EPSON Experimental," [Online]. Available: https://github.com/gavanderhoorn/EPSON_experimental/blob/spel_plus_driver_dev/EPSON_spelp_driver/spel%2B/ros_industrial/EPSON_spelp_driver/ros_state.prg.

[18] „Microsoft Support," [Online]. Available: https://support.microsoft.com/en-us/kb/815065.

[19] „Github Ros industrial fanuc," [Online]. Available: https://github.com/ros-industrial/fanuc.

[20] „ROS wiki Industrial Tutorials Working with Robot Support Packages," [Online]. Available: http://wiki.ros.org/Industrial/Tutorials/WorkingWithRosIndustrialRobotSupportPackages.

[21] „ROS REP," [Online]. Available: http://www.ros.org/reps/rep-0103.html.

[22] „EPSON C3 Robot Manual," [Online]. Available: [http://robots.EPSON.com/admin/uploads/product_catalog/files/EPSON_C3_Robot_Manual%28R7%29.pdf].

[23] „ROS Docs Moveit setup assistant Tutorial," [Online]. Available: http://docs.ros.org/indigo/api/moveit_setup_assistant/html/doc/tutorial.html.

[24] „ROS docs moveit setup assistant tutorial," [Online]. Available: http://docs.ros.org/hydro/api/moveit_setup_assistant/html/doc/tutorial.html.

[25] „Traclabs Track-ik," [Online]. Available: http://traclabs.com/projects/trac-ik/] .

[26] „ROS wiki Industrial Tutorials Creata a MoveIt Pkg for an Industrial Robot," [Online]. Available: http://wiki.ros.org/Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot.

[27] „Wireshark Homepage," [Online]. Available: https://www.wireshark.org/.

[28] „Github ROS Industrial Packet Simple Message," [Online]. Available: [https://github.com/ros-industrial/packet-simplemessage].

[29] „ROS wiki Homepage," [Online]. Available: http://wiki.ros.org/.

[30] „ROS wiki Supported Hardware," [Online]. Available: http://wiki.ros.org/Industrial/supported_hardware.

Bijlagen

Bijlage A: trajectory server van de streamende EPSON C3 ROS-driver

```
' Copyright 2015 TU Delft Robotics Institute
'
' Licensed under the Apache License, Version 2.0 (the "License");
' you may not use this file except in compliance with the License.
' You may obtain a copy of the License at
'
'     http://www.apache.org/licenses/LICENSE-2.0
'
' Unless required by applicable law or agreed to in writing, software
' distributed under the License is distributed on an "AS IS" BASIS,
' WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
' See the License for the specific language governing permissions and
' limitations under the License.
'
' Author: G.A. vd. Hoorn (TU Delft Robotics Institute)
'         L.Castelli (Student at UHasselt/KULeuven, ACRO)
'         W. vd. Aelst (Student at UHasselt/KULeuven, ACRO)

'TODO: fix the seq nr problem (negative number bug)
'TODO: get smooth motion of the robot taking into account the streaming interface

'********************************************************************************
'trajectory server (no perfect smooth motion just yet)
'streaming: movements of the robot in real time (on the fly)
'********************************************************************************

#include "simple_message.inc"
#include "robotinfo.inc"

'in seconds. Smallest 0.01seconds
#define SLEEP_TIME 0.01

'hardcoded for now
#define SOCKET_TAG 202

Function robot_trajectory_main_v2

        'var decls
        Integer bytesAvail, status, counter
        Boolean ShutdownRequested

        'var init
        bytesAvail = 0
        status = 0
        counter = 0
        ShutdownRequested = False

        Power Low
        Speed 50
        Accel 50, 50

        Print "Init done"
        Do While Not ShutdownRequested
                Print "Waiting for client .."
                'socket accept
                CloseNet #SOCKET_TAG
                SetNet #SOCKET_TAG, "192.168.0.2", 11000
```

```
                OpenNet #SOCKET_TAG As Server
                WaitNet #SOCKET_TAG, 50
                Wait SLEEP_TIME
                status = ChkNet(SOCKET_TAG)
                If status < 0 Then
                        Print "Error with socket:", status
                        Exit Function
                EndIf
                Print "ChkNet:", status
                Print "Connected"

                Do While Not ShutdownRequested

                        'client loop, check connection
                        status = ChkNet(SOCKET_TAG)
                        If status < 0 Then
                                Print "Error with socket:", status
                                Exit Function
                        EndIf
                        'Print "ChkNet:", status

                        Call readSimpleMessageAndMoveTheRobot()
                        Call sendServiceReply()

                        'sleep a bit (100ms)
                        Wait SLEEP_TIME '* 100

                        'debug: exit after single iter
                        'Exit Function

                'inner client handling
                Loop

                Print "error?"
                Wait 1.0

        'outer loop
        Loop
        Print "Exit"
Fend

Function readSimpleMessageAndMoveTheRobot()

        'Joint trajectory pt Simple message structure
        '-------------------------------------------------
        'sm_length -> 4 bytes (int32)
        'sm_hdr(0) = sm_id -> 4 bytes (Int32)
        'sm_hdr(1) = sm_comm_type -> 4 bytes (Int32)
        'sm_hdr(2) = sm_reply_type -> 4 bytes (Int32)
        'sequence nr -> 4 bytes (Int32)
        'joint_data -> 40 bytes (10 joints) (Real)
        'velocity -> 4 bytes (Real)
        'duration -> 4 bytes (Real)
        '-------------------------------------------------

        Long sm_length
        Long sm_hdr(2)
        Long seq
        Real joint_data(9)
        Real velocity
        Real duration

        'read first 4 bytes of the simple message (simple message length)
        Integer read_sm_length(3)
        ReadBin #SOCKET_TAG, read_sm_length(), UBound(read_sm_length) + 1
```

```
'Convert the 4 bytes to a Long integer which gives the SM length
sm_length = read_sm_length(0) + read_sm_length(1) * &H100 + read_sm_length(2) * &H10000 + read_sm_length(3)
* &H1000000

Print "sm_length: ", sm_length

'length of a joint_trajectory_pt message = 64 bytes
If sm_length = 64 Then
	'read header of the simple message (including sequence nr part)
	Integer read_hdr(15)
	ReadBin #SOCKET_TAG, read_hdr(), UBound(read_hdr) + 1

	'get values of the simple message header (including sequence part of the sm)
	Call getHeaderValues(ByRef read_hdr(), ByRef sm_hdr(), ByRef seq)
	Print "sm_id: ", sm_hdr(0)
	Print "sm_comm_type: ", sm_hdr(1)
	Print "sm_reply_type: ", sm_hdr(2)
	Print "sequence nr: ", seq

	'this part still needs to be tested
	If sm_hdr(0) <> SM_MSG_TYPE_JOINT_TRAJ_PT Or sm_hdr(1) <>
	SM_COMM_TYPE_SERVICE_REQUEST Or sm_hdr(2) <> SM_REPLY_TYPE_INVALID Then

		OnErr GoTo ErrorHandler
		Error CORRUPT_MSG

	EndIf

	'read the remaining bytes
	Integer read_remaining_bytes(47)
	ReadBin #SOCKET_TAG, read_remaining_bytes(), UBound(read_remaining_bytes) + 1

	'get floatingpoint values(joint_data,velocity,duration) of the sm by
	'converting the remaining bytes using a DLL
	'only do this when we are sure we received a joint trajectory pt message
	'only do this when we are sure we didn't receive a trajectory stop message
	'trajectory stop message has a sequence nr = -4
	If sm_hdr(0) = 11 And seq >= 0 Then
		Short getFloatingpoints

		getFloatingpoints = convertToFloatingpoint(ByRef read_remaining_bytes(), ByRef joint_data(),
		ByRef velocity, ByRef duration)

		Integer k
		For k = 0 To 9
			Print "joint", k, ": ", joint_data(k)
			joint_data(k) = RadToDeg(joint_data(k))
			Print "joint", k, ": ", joint_data(k), "°"
		Next k

		'Move the Robot
		P2 = AglToPls(joint_data(0), joint_data(1), joint_data(2), joint_data(3), joint_data(4), joint_data(5))

		OnErr GoTo ErrorHandler
		Go P2 CP

	ElseIf seq = -4 Then
		Print "received a trajectory stop message, no movement of the robot is allowed"
	EndIf

Else
	OnErr GoTo ErrorHandler
	Error CORRUPT_MSG
EndIf
```

```
	ErrorHandler:

	'+++++++ Error 4031: Motors in off state while trying to make a motion +++++++
		If Err = 4031 Then
			String message4031$
			Integer answer
			Integer helpmovement

			Print "in Errorhandler"

			If Motor = On Then
				helpmovement = 1
			Else
				helpmovement = 0
			EndIf

			If helpmovement = 0 And seq = 0 Then
				message4031$ = "Error 4031: The robot can't move because the motors are off. If you
				want to try again with the motors on click Retry. Else click Cancel"
				MsgBox message4031$, MB_RETRYCANCEL, "Error 4031: motors are in off state",
				answer

				If answer = 4 Then '4 = retry
					Motor On
					Go P2 CP

				ElseIf answer = 2 Then '2 = cancel
					'read data from ROS but don't move
					Print "no movement of the robot --> only reading data..."
					Print "After reading all trajectory points, please close the robotservers and the
					ROS clients."
					Print "In case you want to read more trajectory pt messages. start the servers
					and clients again (after closing them)."
					Wait 3.0
				EndIf
			EndIf
		EndIf

	'+++++++ Error 8000: JOINT_TRAJECTORY_PT message is corrupt +++++++
		If Err = 8000 Then
			'read all data and do nothing with it
			String message8000$
			message8000$ = "JOINT_TRAJECTORY_PT message is corrupt. Hence robot motion is not
			allowed. All robotservers will shut down. Please close the ROS clients"
			MsgBox message8000$, MB_OK, "Error 8000: JOINT_TRAJECTORY_PT message is corrupt"
			Quit All
		EndIf
Fend

Function getHeaderValues(ByRef read_hdr() As Integer, ByRef sm_hdr() As Long, ByRef seq As Long)
	Integer i
	For i = 0 To UBound(sm_hdr()) + 1
		If i < 3 Then
			sm_hdr(i) = read_hdr(i * 4) + read_hdr((i * 4) + 1) * &H100 + read_hdr((i * 4) + 2) * &H10000 +
			read_hdr((i * 4) + 3) * &H1000000
```

```
                Else
                        'when seq = -4 we get an error here (epson bug)
                        'temporary fix for this problem... (we will never have a seq nr higher than 0x01000000 unless we
                        get a negative seq nr (-4 = 0xFFFFFFFC)
                        If (read_hdr((i * 4) + 3) > 0) Then
                                seq = -4
                        Else
                        seq = read_hdr(i * 4) + read_hdr((i * 4) + 1) * &H100 + read_hdr((i * 4) + 2) * &H10000 +
                        read_hdr((i * 4) + 3) * &H1000000
                        EndIf
                EndIf
        Next i
Fend

Function sendServiceReply()
        Integer i
        Integer SMreply(59) 'array to be written to generic client
        For i = 0 To 59
                If i = 0 Then
                        SMreply(i) = 56
                ElseIf i = 4 Then
                        SMreply(i) = SM_MSG_TYPE_JOINT_POSITION
                ElseIf i = 8 Then
                        SMreply(i) = SM_COMM_TYPE_SERVICE_REPLY
                ElseIf i = 12 Then
                        SMreply(i) = SM_REPLY_TYPE_SUCCES
                Else
                        SMreply(i) = 0
                EndIf
        Next i
        'write reply to ROS
        WriteBin #202, SMreply(), UBound(SMreply) + 1
Fend
```

Bijlage B: state server van de EPSON C3 ROS-driver

```
' Copyright 2015 TU Delft Robotics Institute
'
' Licensed under the Apache License, Version 2.0 (the "License");
' you may not use this file except in compliance with the License.
' You may obtain a copy of the License at
'
'     http://www.apache.org/licenses/LICENSE-2.0
'
' Unless required by applicable law or agreed to in writing, software
' distributed under the License is distributed on an "AS IS" BASIS,
' WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
' See the License for the specific language governing permissions and
' limitations under the License.
'
' Author: G.A. vd. Hoorn (TU Delft Robotics Institute)
'         L.Castelli (Student at UHasselt/KULeuven, ACRO)
'         W. vd. Aelst (Student at UHasselt/KULeuven, ACRO)

Declare JointPositionSMtoInts, "DLLsForEpsonDriver.dll"(sm_len As Long, sm_id As Long, sm_comm As Long, sm_reply As
Long, seq As Long, joint1 As Real, joint2 As Real, joint3 As Real, joint4 As Real, joint5 As Real, joint6 As Real, joint7 As Real,
joint8 As Real, joint9 As Real, joint10 As Real, ByRef intvalues() As Integer) As Short

Declare RobotStatusSMtoInts, "DLLsForEpsonDriver.dll"(sm_len As Long, sm_id As Long, sm_comm As Long, sm_reply As
Long, ByRef robot_status_body() As Long, ByRef intvalues() As Integer) As Short

#include "simple_message.inc"
```

```
#include "robotinfo.inc"

'in seconds. Smallest 0.01seconds
#define SLEEP_TIME 0.01

'hardcoded for now
#define SOCKET_TAG 201

Function robot_state_mainv3()


		'var decls
		Integer bytesAvail, status, counter
		Boolean ShutdownRequested

		'var init
		bytesAvail = 0
		status = 0
		counter = 0
		ShutdownRequested = False

		Print "Init done"

		Do While Not ShutdownRequested

			Print "Waiting for client .."

			'socket accept
			CloseNet #SOCKET_TAG
			SetNet #SOCKET_TAG, "192.168.0.2", 11002
			OpenNet #SOCKET_TAG As Server
			WaitNet #SOCKET_TAG, 50
			Wait SLEEP_TIME
			status = ChkNet(SOCKET_TAG)
			If status < 0 Then
				Print "Error with socket:", status
				Exit Function
			EndIf
			'Print "ChkNet:", status

			'should have a client here (?)
			Print "Connected"

			Do While Not ShutdownRequested

				'client loop, check connection
				status = ChkNet(SOCKET_TAG)
				If status < 0 Then
					Print "Error with socket:"
					Exit Function
				EndIf
				'Print "ChkNet:", status

				OnErr GoTo ErrorHandler
				Call SendRobotStateToROSv3()
				counter = counter + 1
				If counter = 10 Then
					counter = 0

					OnErr GoTo ErrorHandler
					Call SendRobotStatusSM
				EndIf
```

```
                    'sleep a bit (100ms)
                    Wait SLEEP_TIME '* 100

                    'debug: exit after single iter
                    'Exit Function

                'inner client handling
                Loop

                Print "error?"
                'Wait 1.0

                ErrorHandler:
                        Quit All

        'outer loop
        Loop
        Print "Exit"

Fend

Function GetJointValuesV3(ByRef data() As Real)
        'assume 6 axis robot for now, single group, only revolute joints
        data(0) = DegToRad(Agl(1))
        'Print data(0)
        data(1) = DegToRad(Agl(2))
        'Print data(1)
        data(2) = DegToRad(Agl(3))
        'Print data(2)
        data(3) = DegToRad(Agl(4))
        'Print data(3)
        data(4) = DegToRad(Agl(5))
        'Print data(4)
        data(5) = DegToRad(Agl(6))
        'Print data(5)
Fend


Function SendRobotStateToROSv3()
        'Joint Position Simple message structure
        '-------------------------------------------------
        'sm_hdr(0) = message length -> 4 bytes (int32)
        'sm_hdr(1) = sm_id -> 4 bytes (Int32)
        'sm_hdr(2) = sm_comm_type -> 4 bytes (Int32)
        'sm_hdr(3) = sm_reply_type -> 4 bytes (Int32)
        'sequence nr -> 4 bytes (Int32)
        'joint_data -> 40 bytes (10 joints) (Real)
        '-------------------------------------------------

        Long sm_hdr(3)
        Long seq
        Real joint_data(9)

        sm_hdr(0) = ((UBound(sm_hdr) + 1) * 4) + ((UBound(joint_data) + 1) * 4) ' = 56 bytes
        sm_hdr(1) = SM_MSG_TYPE_JOINT_POSITION
        sm_hdr(2) = SM_COMM_TYPE_TOPIC
        sm_hdr(3) = SM_REPLY_TYPE_INVALID
        seq = 0

        GetJointValuesV3(ByRef joint_data())

        Integer intvaluesSM(255)
        Integer getIntValuesSM
```

```
		getIntValuesSM = JointPositionSMtoInts(sm_hdr(0), sm_hdr(1), sm_hdr(2), sm_hdr(3), seq, joint_data(0),
		joint_data(1), joint_data(2), joint_data(3), joint_data(4), joint_data(5), joint_data(6), joint_data(7),
		joint_data(8), joint_data(9), ByRef intvaluesSM())


		WriteBin #SOCKET_TAG, intvaluesSM(), 60

Fend

Function SendRobotStatusSM
		'Robot Status Simple message structure
		'-----------------------------------------------------
		'robot_status_hdr(0) = message length (Int32)
		'robot_status_hdr(1) = message type/id (Int32)
		'robot_status_hdr(2) = comm type (Int32)
		'robot_status_hdr(3) = reply type (Int32)
		'robot_status_body(0) = drives powered (Int32)
		'robot_status_body(1) = e stopped (Int32)
		'robot_status_body(2) = error code (Int32)
		'robot_status_body(3) = in error(Int32)
		'robot_status_body(4) = in motion (Int32)
		'robot_status_body(5) = mode (Int32)
		'robot_status_body(6) = motion possible (Int32)
		'-----------------------------------------------------
		Long robot_status_hdr(3)
		Long robot_status_body(6)

		'setup prefix & header
		robot_status_hdr(0) = (UBound(robot_status_hdr) * 4) + ((UBound(robot_status_body) + 1) * 4)
		robot_status_hdr(1) = SM_MSG_TYPE_STATUS
		robot_status_hdr(2) = SM_COMM_TYPE_TOPIC
		robot_status_hdr(3) = SM_REPLY_TYPE_INVALID

		'drives powered
		If Motor = On Then
			robot_status_body(0) = SM_TRI_STATE_TRUE
		Else
			robot_status_body(0) = SM_TRI_STATE_FALSE
		EndIf

		'e stopped
		If EStopOn = True Then
			robot_status_body(1) = SM_TRI_STATE_TRUE
		Else
			robot_status_body(1) = SM_TRI_STATE_FALSE
		EndIf

		'error code
		If Ert > 0 Then
			OnErr GoTo Error2261 'error: specified task number does not exist
					'(happens when the emergency button aborted the trajectory server task)

			robot_status_body(2) = Err(Ert)
		Else
			robot_status_body(2) = 0
		EndIf

		'in error
		If ErrorOn = -1 Then
			robot_status_body(3) = SM_TRI_STATE_TRUE
		Else
			robot_status_body(3) = SM_TRI_STATE_FALSE
		EndIf

		'in motion
		'TODO: find a way to check this part of the status msg!
		robot_status_body(4) = SM_TRI_STATE_UNKNOWN
```

```
'robot mode
If CtrlInfo(7) = 0 Then
	'Running project(Run Window F5) = Manual mode
	robot_status_body(5) = SM_ROBOT_MODE_MANUAL
ElseIf CtrlInfo(7) = 1 Then
	'test auto mode (shift F5) = Auto mode
	robot_status_body(5) = SM_ROBOT_MODE_AUTO
EndIf

'motion possible
'TODO: find a way to check this part of the msg!
robot_status_body(6) = SM_TRI_STATE_UNKNOWN

'Write robot status message to ROS
Integer intvaluesSM(43)
Short dllFinished
dllFinished = RobotStatusSMtoInts(robot_status_hdr(0), robot_status_hdr(1), robot_status_hdr(2),
robot_status_hdr(3), ByRef robot_status_body(), ByRef intvaluesSM())
WriteBin #SOCKET_TAG, intvaluesSM(), 44

Error2261:
	robot_status_body(2) = 2261

Fend
```

Bijlage C: Main programma (opstarten van de servers)

```
Declare convertToFloatingpoint, "DLLsForEpsonDriver.dll"(ByRef inputintegers() As Integer, ByRef output_joint_data() As
Real, ByRef output_velocity As Real, ByRef output_duration As Real) As Short

Function main

Xqt robot_state_mainv3, NoEmgAbort
Xqt robot_trajectory_main_v2, Normal
'Call robot_trajectory_main_smooth

Fend
```

Bijlage D: simple message protocol include file

```
' Copyright 2015 TU Delft Robotics Institute
'
' Licensed under the Apache License, Version 2.0 (the "License");
' you may not use this file except in compliance with the License.
' You may obtain a copy of the License at
'
'     http://www.apache.org/licenses/LICENSE-2.0
'
' Unless required by applicable law or agreed to in writing, software
' distributed under the License is distributed on an "AS IS" BASIS,
' WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
' See the License for the specific language governing permissions and
' limitations under the License.

' Constants used in the ROS-Industrial Simple Message protocol
'
' Author: G.A. vd. Hoorn (TU Delft Robotics Institute)
'
```

```
#ifndef SIMPLE_MESSAGE_H__
#define SIMPLE_MESSAGE_H__

'Standard lengths
#define SM_LEN_PREFIX 4
#define SM_LEN_HDR (3 * 4)

'Standard Message Set IDs (from REP-I0004)
#define SM_MSG_TYPE_INVALID 0
#define SM_MSG_TYPE_JOINT_TRAJ_PT 11
#define SM_MSG_TYPE_JOINT_TRAJ 12
#define SM_MSG_TYPE_STATUS 13
#define SM_MSG_TYPE_JOINT_TRAJ_PT_FULL 14
#define SM_MSG_TYPE_JOINT_FEEDBACK 15

'Deprecated message types
#define SM_MSG_TYPE_JOINT_POSITION 10
#define SM_MSG_TYPE_JOINT 10
#define SM_MSG_TYPE_READ_INPUT 20
#define SM_MSG_TYPE_WRITE_OUTPUT 21

'Header comm type values
#define SM_COMM_TYPE_INVALID 0
#define SM_COMM_TYPE_TOPIC 1
#define SM_COMM_TYPE_SERVICE_REQUEST 2
#define SM_COMM_TYPE_SERVICE_REPLY 3

'Header Reply type values
#define SM_REPLY_TYPE_INVALID 0
#define SM_REPLY_TYPE_SUCCES 1
#define SM_REPLY_TYPE_FAILURE 2


'Robot Status - Robot Mode Values
#define SM_ROBOT_MODE_UNKNOWN -1
#define SM_ROBOT_MODE_MANUAL 1
#define SM_ROBOT_MODE_AUTO 2

'Robot Status - Tristate values
#define SM_TRI_STATE_UNKNOWN -1
#define SM_TRI_STATE_TRUE 1
#define SM_TRI_STATE_FALSE 0

#endif
```

Bijlage E: robot info include file

```
#ifndef ROBOTINFO_H__
#define ROBOTINFO_H__

'Number of Joints
#define NR_OF_JOINTS 6

#endif
```

Bijlage F: geschreven DLL-functies

```
// DLLsForEpsonDriver.cpp : Defines the exported functions for the DLL application.

#include "stdafx.h"
#include "DLLsForEpsonDriver.h"
```

```
DLLSFOREPSONDRIVER_API short JointPositionSMtoInts(int sm_length,int sm_id ,int sm_comm_type,int sm_reply_type, int seq,float joint1,float joint2,float joint3,float joint4,float joint5 , float joint6, float joint7, float joint8,float joint9, float joint10,short* outputIntegers)
{
   float joint_data[10];
   joint_data[0]= joint1;
   joint_data[1]= joint2;
   joint_data[2]= joint3;
   joint_data[3]= joint4;
   joint_data[4]= joint5;
   joint_data[5]= joint6;
   joint_data[6]= joint7;
   joint_data[7]= joint8;
   joint_data[8]= joint9;
   joint_data[9]= joint10;

            unsigned char buffer[60];
            memcpy(buffer,&sm_length,sizeof(sm_length));
            memcpy(buffer + 4,&sm_id,sizeof(sm_id));
            memcpy(buffer + 8,&sm_comm_type,sizeof(sm_comm_type));
            memcpy(buffer + 12,&sm_reply_type,sizeof(sm_reply_type));
            memcpy(buffer + 16,&seq,sizeof(seq));
            memcpy(buffer + 20, &joint_data, sizeof joint_data);

   for (int i=0;i<sizeof(buffer);i++)
   {
                        outputIntegers[i]=buffer[i];
   }

   return 1;
}

DLLSFOREPSONDRIVER_API short RobotStatusSMtoInts(int sm_length,int sm_id ,int sm_comm_type,int sm_reply_type,int* robot_status_body, short* outputIntegers)
{
            unsigned char buffer[44];
            memcpy(buffer,&sm_length,sizeof(sm_length));
            memcpy(buffer + 4,&sm_id,sizeof(sm_id));
            memcpy(buffer + 8,&sm_comm_type,sizeof(sm_comm_type));
            memcpy(buffer + 12,&sm_reply_type,sizeof(sm_reply_type));
            memcpy(buffer + 16, robot_status_body,28);



   for (int i=0;i<sizeof(buffer);i++)
   {
                        outputIntegers[i]=buffer[i];
   }

   return 1;
}

DLLSFOREPSONDRIVER_API short convertToFloatingpoint(short* inputintegers,float* joint_data,float velocity,float duration)
{
   unsigned char jointbuffer[40];
   unsigned char velocitybuffer[4];
   unsigned char durationbuffer[4];

    // filter the most significant bytes out of the input array
    for(int i=0;i<48;i++)
   {
      if(i<40)
      {
         jointbuffer[i] = static_cast<unsigned char>(inputintegers[i]);
      }
      if(i>=40 && i<44)
      {
```

```
            velocitybuffer[i-sizeof(jointbuffer)] = static_cast<unsigned char>(inputintegers[i]);
        }
        if (i>=44 && i<48)
        {
            durationbuffer[i-sizeof(jointbuffer)-sizeof(velocitybuffer)] = static_cast<unsigned char>(inputintegers[i]);
        }
    }

    //copy bufferdata to the concerned memory block
    memcpy(joint_data,&jointbuffer,sizeof(jointbuffer));
    memcpy(&velocity,&velocitybuffer,sizeof(velocitybuffer));
    memcpy(&duration,&durationbuffer,sizeof(durationbuffer));

    return 1;
}
```

Bijlage G: def file voor de geschreven DLL-functies

```
LIBRARY         "DLLsForEpsonDriver"
        EXPORTS
                JointPositionSMtoInts
                convertToFloatingpoint
                RobotStatusSMtoInts
```

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Ontwikkeling van een ROS-driver voor Epson C3 robots

Richting: **master in de industriële wetenschappen: energie-automatisering**
Jaar: **2016**

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,

Castelli, Luca **Van der Aelst, Wim**

Datum: **6/06/2016**