



# Zesassige robotarm

Lowiek Van den Stockt

5 juni 2019

6IW nr.10

Mentor: L. Grammens

Vrij Technisch Instituut Aalst 2018-2019

# Woord vooraf

Mijn naam is Lowiek Van den Stockt. Ik zit in het laatste jaar industriële wetenschappen in het VTI te Aalst. Ik vind dat een geïntegreerde proef een project moet zijn waar je jezelf mee kan uitdagen op verschillende vlakken. Alleen zo leer je zelfstandig informatie verwerken, hoe je kritisch met bronnen moet omgaan en hoe je nieuwe kennis en vaardigheden ontwikkelt om een project tot een goed einde te brengen. Bovendien leer je ook nog hoe je een goede planning opstelt.

De geïntegreerde proef zou je de kers op de taart kunnen noemen van onze middelbare schoolcarrière. In de voorbije zes jaar hebben we een goede basis van kennis en vaardigheden opgebouwd. Je moet echter bij deze proef bewijzen dat je in staat bent om zelfstandig op deze basis verder te bouwen. Dit is een belangrijke vaardigheid om later te hebben, zowel in het hoger onderwijs als in het professionele leven. Deze proef laat je toe om die vaardigheid te ontwikkelen in een vrij vergevingsgezinde omgeving, namelijk de middelbare school.

Ik heb gekozen om een zesassige robotarm te ontwikkelen. Dit was op elk vlak een uitdaging voor mij. Gelukkig laat ik me niet afschrikken door een uitdaging. Want hoe kan je leren van een project dat voor jou amper iets voorstelt?

Ik heb gekozen om dit project alleen te doen, maar uiteraard kon ik dit niet tot een goed einde brengen zonder hulp van derden. Daarom wil ik in de eerste plaats mijn ouders bedanken voor de financiële middelen, het vertrouwen en de vrijheid die ze mij gegeven hebben.

Verder wil ik meneer Grammens bedanken als mentor, voor alle hulp die hij gegeven heeft voor het zwaar wiskundig gedeelte van mijn geïntegreerde proef.

Ook plaats ik graag een woordje van dank richting alle onbekende derden. Hiermee bedoel ik alle mensen die een bijdrage geleverd hebben aan alle resources die ik geraadpleegd heb, al dan niet via het internet. Ik heb mijn best gedaan om zo veel mogelijk referenties te plaatsen waar nodig, maar ik zal er hoe dan ook een aantal vergeten zijn.

Ook wens ik Stijn De Mil van FabLab Factory te bedanken, enerzijds voor het openstellen van het FabLab zodat ik gebruik kon maken van bijvoorbeeld de lasersnijder, maar anderzijds ook voor de vele technische ervaring dat ik al opgedaan heb als jobstudent bij FabLab Factory. Zonder deze ervaring zou ik nooit gedurfd hebben om aan dit project te beginnen.

Tenslotte ben ik in het bijzonder İlhan Ünal dankbaar. Hij leende me allerlei belangrijk materiaal uit, zoals een ESC (Electronic Speed Controller) om mijn motoren en assen te testen toen ik mijn echte hardware nog niet had.

# Inhoudstafel

<b>Inleiding</b>	<b>5</b>
<b>1 Motoren</b>	<b>7</b>
1.1 Inleiding . . . . .	7
1.2 Encoders . . . . .	7
1.3 BLDC-motoren . . . . .	9
1.3.1 Opbouw . . . . .	9
1.3.2 Werking . . . . .	10
1.3.3 Turnigy Aerodrive SK3 4250 - 350Kv . . . . .	13
1.3.4 Propdrive v2 3536 910Kv . . . . .	14
<b>2 Eerste iteratie: conceptueel</b>	<b>15</b>
2.1 Inleiding . . . . .	15
2.2 As 1 . . . . .	16
2.3 As 2 . . . . .	16
2.4 As 3 . . . . .	17
2.5 As 4 . . . . .	18
2.6 Differentieel . . . . .	18
2.7 As 5 en as 6 . . . . .	19
<b>3 Tweede iteratie: de doorbraak</b>	<b>20</b>
3.1 Inleiding . . . . .	20
3.2 Planeetwielmechanisme . . . . .	20
3.3 Compound Planetary Gear Box . . . . .	21
3.4 As 1 . . . . .	23
3.5 As 2 . . . . .	24
3.6 As 3 . . . . .	25
3.7 As 4 . . . . .	25
3.8 As 5 en as 6 . . . . .	26
<b>4 Derde iteratie</b>	<b>27</b>
4.1 Inleiding . . . . .	27
4.2 Bodemplaat . . . . .	27
4.3 As 1 . . . . .	29
4.4 As 2 . . . . .	30
4.5 As 3 . . . . .	30
4.6 As 4 . . . . .	32
4.7 As 5 . . . . .	33
4.8 As 6 . . . . .	33
4.9 End effector . . . . .	34
4.10 De grijper . . . . .	35
<b>5 Elektronica behuizing</b>	<b>36</b>
5.1 Inleiding . . . . .	36
5.2 Plaatsing elektronica . . . . .	36
5.3 Stroomvoorziening . . . . .	37
5.4 Voorpaneel . . . . .	38
<b>6 Toekomstvisie praktische uitwerking</b>	<b>38</b>

<b>7 Denavit-Hartenbergconventie</b>	<b>39</b>
7.1 Inleiding . . . . .	39
7.2 Denavit-Hartenbergassenstelsels . . . . .	39
7.3 Denavit-Hartenbergparameters . . . . .	40
7.4 Denavit-Hartenbergtransformatiematrix . . . . .	41
7.5 De inverse van de DH-transformatiematrix . . . . .	42
<b>8 Kinematica</b>	<b>44</b>
8.1 Inleiding . . . . .	44
8.2 Positie van de end-effector . . . . .	44
8.3 Oriëntatie van de end-effector: eulerhoeken . . . . .	44
8.4 Oriëntatie van de end-effector: as-hoek methode . . . . .	46
8.5 Quaternionen . . . . .	47
8.5.1 Complexe getallen . . . . .	47
8.5.2 Uitbreidning naar de quaternionen . . . . .	49
8.6 Voorwaartse kinematica . . . . .	50
8.7 Inverse kinematica . . . . .	52
8.8 Jacobimatrixmethode . . . . .	52
8.8.1 De Jacobimatrix . . . . .	52
8.8.2 Toegepast op inverse kinematica . . . . .	53
8.9 Gradient descentmethode . . . . .	54
8.10 IKFast . . . . .	56
<b>9 PID-loops</b>	<b>57</b>
9.1 Inleiding . . . . .	57
9.2 Werking . . . . .	58
9.3 Tunen van de parameters . . . . .	59
<b>10 ODrive</b>	<b>60</b>
10.1 Inleiding . . . . .	60
10.2 Hardware . . . . .	60
10.2.1 Microcontroller (1) . . . . .	61
10.2.2 Driefasige gate driver (2) . . . . .	61
10.2.3 N-channel MOSFETs (output stage) (3) . . . . .	62
10.2.4 N-channel MOSFETs (vermogensweerstand) (4) . . . . .	63
10.2.5 Connectoren, IO en communicatie (5-11) . . . . .	64
10.3 Firmware . . . . .	64
10.3.1 Controlelus . . . . .	64
10.4 Trapeziumvormige trajectory planner . . . . .	65
<b>11 ROS</b>	<b>66</b>
11.1 Inleiding . . . . .	66
11.2 Fundamentele concepten . . . . .	67
11.2.1 ROS Packages . . . . .	67
11.2.2 Nodes . . . . .	68
11.2.3 Topics . . . . .	68
11.2.4 Messages . . . . .	71
11.2.5 Services . . . . .	72
11.2.6 Master . . . . .	72
11.2.7 Parameter Server . . . . .	74
11.3 MoveIt . . . . .	75

11.3.1 move_group . . . . .	75
11.3.2 User interfaces . . . . .	76
11.3.3 Robotic interfaces . . . . .	76
11.3.4 MoveIt! Setup Assistant . . . . .	77
11.4 Trajectory generator . . . . .	77
11.4.1 Het plannen van een pad . . . . .	78
11.4.2 Het parsen van een pad . . . . .	79
<b>12 ODriveControl</b>	<b>80</b>
12.1 Inleiding . . . . .	80
12.2 Interfaces . . . . .	81
12.2.1 ODriveControl - hardware . . . . .	81
12.2.2 Communicatie ODriveControl - ROS . . . . .	82
12.2.3 Communicatie ODriveControl - parameters.xml . . . . .	84
12.3 Calibratiemenu . . . . .	85
12.4 Plotters . . . . .	86
<b>13 Praktische uitwerking van de software</b>	<b>87</b>
13.1 Inleiding . . . . .	87
13.2 Het opstarten van de software . . . . .	88
<b>14 Toekomstvisie van de software</b>	<b>89</b>
<b>Besluit</b>	<b>90</b>
<b>Figurenlijst</b>	<b>91</b>
<b>Tabellenlijst</b>	<b>93</b>
<b>Bijlagen</b>	<b>96</b>
A. De inverse van de DH-transformatiematrix . . . . .	96
B. Bewijs van de overbrengingsverhouding van de hoofdoverbrenging . . . . .	99
C. Wiskundige uitwerking van een trapeziumvormige trajectory planner . . . . .	102
D. Ma commande pour mon épreuve intégrée . . . . .	105
E. My integrated project . . . . .	106
F. Geschatte kostprijsberekening . . . . .	107
G. Afmetingen van de Turnigy SK3 Aerodrive 4250 - 350Kv . . . . .	108
H. Afmetingen van de algemene end-effector interface . . . . .	109
I. Elektrisch schema van een ODrive . . . . .	110

# Inleiding

Mijn eindwerk is het ontwerpen, realiseren en onderzoeken van een robotarm. Niet zomaar een robotarm, maar een zesassig exemplaar dat gebruikmaakt van een technologie op een industrieel niveau. Voor de articulaties gebruik ik namelijk driefasige, borstelloze motoren. Deze motoren zijn sneller, krachtiger én efficiënter. Het grote nadeel is dat zo'n motor aansturen niet eenvoudig is, vanwege de extra externe positiefeedback die je nodig hebt. Het is trouwens nog niet zolang dat deze technologie betaalbaar is voor hobbyisten. In tegenstelling tot de industrie, waar bijna elke robotarm gebruikmaakt van servo's. Wat een verzamelnaam is van een DC- of borstelloze motor en het positiefeedbacksysteem.

Concreet wordt mijn robotarm bestuurd door de coördinaten van de end-effector in te geven. Het kinematicamodel berekent op basis daarvan expliciet de benodigde stand van elke articulatie. Het kinematicamodel dat ik hiervoor hanteer spuwt meestal meerdere oplossingen uit, al dan niet praktisch uitvoerbaar. Daarom zijn er ook nog een aantal supplementaire en toch wel gesofisticeerde optimalisatiemodellen en -agents nodig. Dan denk ik bijvoorbeeld aan een collision detection algoritme en een motion planner. De motion planner moet op basis van (onder andere) het collision detection algoritme ervoor zorgen dat tijdens de motion planning zowel veiligheid als efficiëntie gemaximaliseerd wordt.

In december 2017 had ik een bestaande robotarm gerepareerd en geherprogrammeerd voor mijn werkgever, FabLab Factory. Die robot heeft maar vijf assen, wat je geen volledige bewegingsvrijheid biedt. Ook gebruikt deze robotarm stappenmotoren. Deze motoren zijn van nature niet snel noch krachtig. De grote overbrengingsverhouding die je als gevolg nodig hebt, heeft een nog grotere invloed op de snelheid. Dit is een van de redenen waarom ik een zesassige versie wil ontwerpen, met motoren die snel én krachtig zijn.

In dit project komt ook alles aan bod: fysica, mechanica, elektriciteit, softwareontwikkeling, zelfstudievaardigheden en wiskunde. Het laatste is mijn favoriete vak, dus leek het me verstandig om dat er veel in te verwerken. Ook daag ik mijzelf uit door dit project te kiezen. Alleen door jezelf uit te dagen kan je namelijk weten wat je wel of niet graag doet, en waar je werkpunten liggen. Daar bovenop zal ik hoogstwaarschijnlijk ook een aantal keer mijzelf tegenkomen. Ik zal dus meer te weten komen over mezelf en hoeveel ik wil gaan om projecten te realiseren. Met deze informatie kan je een betere keuze maken naar het hoger onderwijs toe. Waar het begrip 'onderwijs' naar een ander, en in mijn ogen beter, niveau wordt getild.

Na mijn secundaire schoolcarrière, verdwijnt dit project niet zomaar op zolder. Op lange termijn is het de bedoeling dat deze robotarm een open-source platform wordt. Dit zorgt ervoor dat iedereen het kan en mag gebruiken en aanpassen voor niet-commerciële doeleinden. Dit is dé manier om bij te dragen aan de open-source community. Moest deze community niet bestaan, dan zou ik nooit in staat geweest zijn om dit project tot een goed einde te brengen.

Het dossier is nogal theoretisch gericht, met voornamelijk besprekingen van theoretische structuren. Om het dossier leesbaar en boeiend te houden voor iedereen, heb ik bijvoorbeeld de bewijzen van wiskundige modellen als bijlage toegevoegd. Wat ik ook heb beschreven, zijn mislukte ideeën, alternatieven en aandachtspunten voor de

volgende iteratie.

De taken van Frans en Engels zijn te vinden in de bijlagen, net zoals de geschatte kostprijsberekening van dit project. Al het beeldmateriaal van dit project is te vinden op een openbare Google Photos map, beschikbaar via de volgende link: <https://tinyurl.com/fotosgip>

# 1 Motoren

## 1.1 Inleiding

De motoren zijn uiteraard het meest belangrijke onderdeel van de robotarm. Zonder motoren zal er niet veel bewegen. Ik had een aantal keuzes voor het type motor. De meest voor de hand liggende motor is de stappenmotor. Dit is de makkelijkste en goedkoopste oplossing omdat het in de makerwereld het meest gebruikte type motor is. Ook hebben stappenmotoren niet noodzakelijk een feedbacksysteem nodig om aan positiecontrole te doen, al is dit wel aangeraden. Uiteindelijk heb ik toch niet voor de stappenmotor gekozen omdat deze veel trager en minder krachtig is. Ook bestaan er al veel open-source robotarmen die gebruik maken van stappenmotoren.

Een andere optie is om gebruik te maken van DC-motoren. Deze kunnen sneller en krachtiger zijn dan stappenmotoren van dezelfde grootte, zijn ook relatief goedkoop en makkelijk aan te sturen. Wel zou ik nu gebruik moeten maken van een feedbacksysteem en één of meerdere PID-controllers om aan positiecontrole te kunnen doen.

Omdat DC-motoren niet echt efficiënt zijn voor grote vermogens vanwege de koolstofborstels, heb ik dan uiteindelijk gekozen voor BLDC-motoren. BLDC staat voor 'brushless DC'. Deze naam is redelijk verwarringend omdat de motoren dankzij de borstelloze constructie aangedreven moet worden door een driefasige wisselspanning. De driefasige wisselspanning wordt 'gesimuleerd' door een DC-spanning op een complexe manier te commuteren over de drie fases. Daarom dat het een borstelloze DC-motor genoemd wordt. Omdat deze motoren geen koolstofborstels gebruiken, kan het veel efficiënter een groter vermogen overbrengen.

Als ik deze motoren wil besturen op basis van positiecommando's, moet ik een extern feedbacksysteem toevoegen. Ik heb geopteerd voor het plaatsen van optische incrementele encoders. Voor de hardware dat verantwoordelijk is voor het commuteren van de motor op basis van positiefeedback van een encoder en PID-controllers maak ik gebruik van drie ODrives. (Zie hoofdstuk 10)

## 1.2 Encoders

Een encoder is een elektronisch toestel met een roterende as dat pulsen geeft als die as zich een bepaald aantal graden verdraaid heeft. Op basis van die informatie kan je berekenen onder welke hoek die as staat, alsook snelheid en richting. Er zijn verschillende types encoders. De voornaamste types zijn incrementale en absolute encoders. Absolute encoders geven hun positie vanaf een vast referentiepunt, terwijl incrementale encoders gewoon pulsen sturen wanneer er een verandering in hoek is. Bij de incrementale encoder is er dus software nodig dat zelf een 'absolute' positie bijhoudt en eventueel een homingprocedure aanbiedt. Aangezien ik enkel incrementale encoders gebruik heb, zal ik ook alleen maar dat type bespreken.



Fig. 1: De incrementale encoders die ik gebruik

Bron: Kaspars Dambis, *Maytech 5055 75kV BLDC outrunner motor rotor and stator*

Een incrementale encoder heeft minstens vier draden. Twee van die draden zijn voor de stroomvoorziening. Mijn encoders hebben 5-24V DC nodig. De twee andere draden zijn de pulstreinen A en B. Dit zijn de draden waar de pulsen opkomen. Zoals je op onderstaande afbeelding ziet geven de twee pulstreinen pulsen met een faseverschil van  $90^\circ$ .

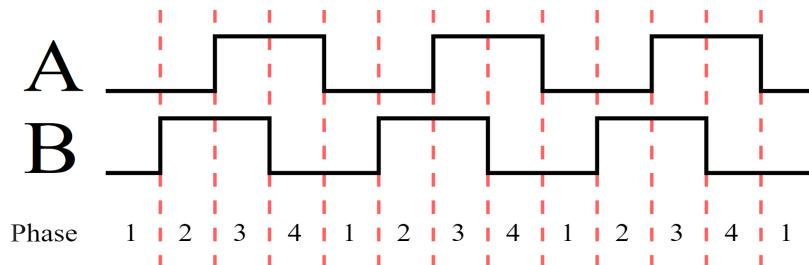


Fig. 2: De voorstelling van de pulsen op A en B

Bron: Wikipedia, *Incremental encoder*

Mijn encoders hebben een PPR (Pulse Per Revolution) van 600. Dat wil zeggen dat elke pulstrein 600 keer per revolutie een puls zal sturen. Omdat de twee pulstreinen  $90^\circ$  uit fase zijn en we beide pulstreinen gebruiken, krijgen we zo een CPR (Counts Per Revolution) van 2400. Dat wil zeggen dat de software in staat is om 2400 keer per revolutie een verandering te meten in de rotatie van de encoderaas. Met andere woorden kan het een hoekverschil merken van  $360^\circ/2400 = 0.15^\circ$ . Afhankelijk van welke pulstrein eerst een puls geeft kan de software ook bepalen in welke richting de encoder aan het draaien is.

Soms zal er ook een Z-pulstrein aanwezig zijn. Deze geeft maar één keer per revolutie en op een vaste locatie een puls. Dit voert dus een soort van referentiepunt in en kan het bijhouden van de hoek en het homingproces makkelijker maken.

Naar: Wikipedia, *Incremental encoder*

## 1.3 BLDC-motoren

### 1.3.1 Opbouw

BLDC-motoren zijn dus driefasige motoren die aangestuurd worden door een DC-spanning te commuteren over drie fases. Er zijn twee types BLDC-motoren: de 'outrunners' en de 'inrunners'. Het enige verschil tussen de twee is dat de rotor langs de buitenkant of langs de binnenkant zit.



Fig. 3: Een gedeassembleerde BLDC-motor van het type 'outrunner'

Bron: Kaspars Dambis, *Maytech 5055 75kV BLDC outrunner motor rotor and stator*

Op de bovenstaande afbeelding kan je een typische opbouw van een outrunner zien. Rechts heb je de stator dat opgebouwd is uit gelamineerd staal om de wervelstromen te beperken. Verder heb je ook nog de rotor. De binnenkant van de rotor is voorzien van een aantal permanente magneten, geplaatst met een wisselende polariteit. Het aantal poolparen van deze motor is het aantal polen (of aantal permanente magneten) gedeeld door twee. Ook is de rotor nog verbonden aan de interne rotatieas die met behulp van kogellagers ondersteund wordt.

Elke statorarm bevat een wikkeling uit dun koperdraad dat op basis van het aantal wikkelingen in staat is om een magnetisch veld op te wekken als er stroom door vloeit. Deze wikkelingen zijn op een speciale manier in drie fases ingedeeld en daarna in ster met elkaar verbonden, zoals je kan zien in de onderstaande afbeelding. Vanaf de wikkelingen worden dan de drie fasedraden afgetakt. De nulleider moet niet meegetrokken worden omdat de drie fases identiek zijn en er dus geen stroom door de nulleider zal vloeien.

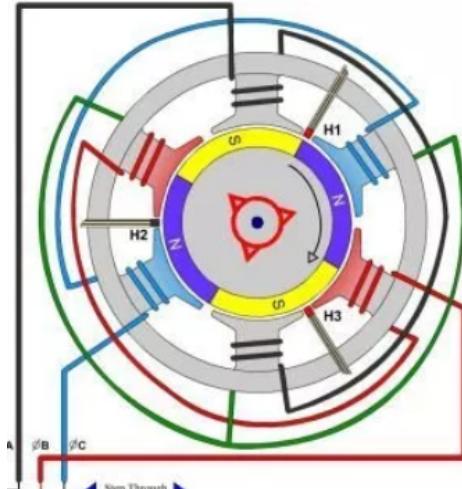


Fig. 4: Schematische voorstelling van een stator. (type: inrunner)

Bron: Elliott Wertheimer, *How do electric motors in ebikes work?*

### 1.3.2 Werking

Eerst en vooral moeten we een aantal termen definiëren. De rotatie door twee polen (rotatie door twee permanente magneten van de rotor zodanig dat de relatieve rotatie tussen de rotor en de stator opnieuw hetzelfde wordt) noemen we een elektrische cyclus. Voor een 14-polige motor (7 poolparen) moeten we dus zeven elektrische cycli uitvoeren voor één mechanische revolutie.

Commutatie is het proces dat bepaalde stromen door de drie fases van de motor stuurt om een beweging te veroorzaken. De simpelste vorm van commutatie is gewoon fase na fase onder spanning zetten. Stel dat je fase A onder spanning zet, dan zal die fase een magnetisch veld oprichten en zullen de permanente magneten van de rotor zich daar naar richten. Als je vervolgens fase C onder spanning zet, dan zal het magnetisch veld van richting veranderen, waardoor de rotor opnieuw gaat draaien om zichzelf met dat veld uit te lijnen. Door dit herhaaldelijk te doen krijg je een roterend magnetisch veld en dus een doorlopende draaibeweging.

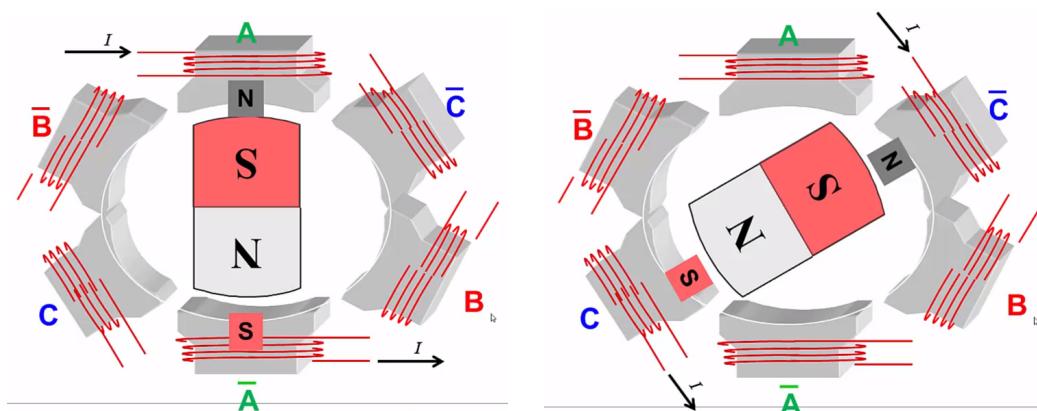


Fig. 5: De simpelste vorm van commutatie

Bron: Texas Instruments, *Commutation techniques for three phase brushless DC motors*

**Trapeziumvormige- of  $120^\circ$ -commutatie** Dit is een redelijk eenvoudige vorm van commutatie. De stroom door een fase is blokvormig, zoals je kan zien in de onderstaande linkerafbeelding. Ook staat een fase maar voor  $\frac{2}{3}$  van een volledige elektrische periode onder spanning, namelijk in twee blokken van telkens  $120^\circ$  (vandaar de naam). In de andere afbeelding zie je de eenvoudige blokspanningen die over de fases gezet moeten worden om dit effect te verkrijgen. Omdat de fases niet de hele tijd onder spanning staan, is het geleverd moment niet constant. Dit zorgt voor veel trillingen, geluid en een slecht rendement.

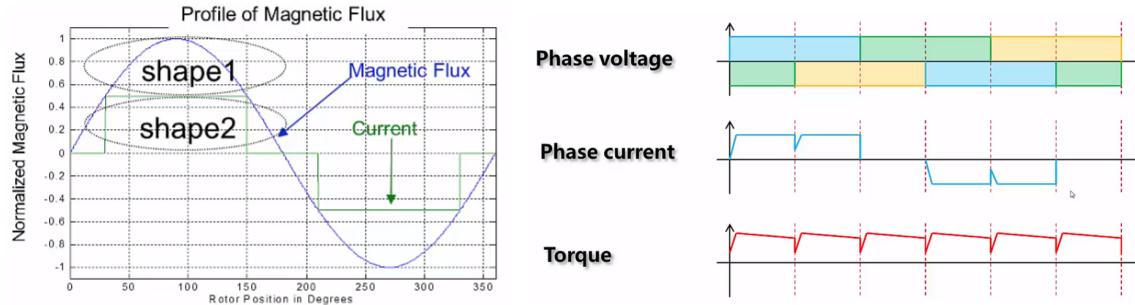


Fig. 6: Trapeziumvormige- of  $120^\circ$ -commutatie

Bron: Texas Instruments, *Commutation techniques for three phase brushless DC motors*

**Sinusvormige- of  $180^\circ$ -commutatie** Een veel betere methode om te commuteren is de sinusvormige- of  $180^\circ$ -commutatie. Hier moet je er voor zorgen dat de stroom op de fases sinusvormig is.

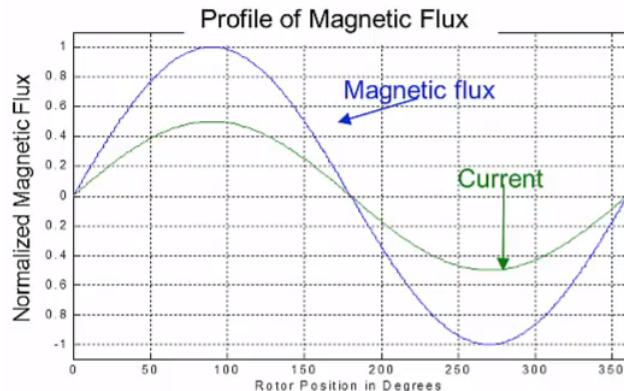
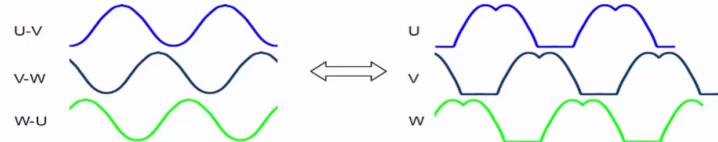


Fig. 7: Sinusvormige- of  $180^\circ$ -commutatie

Bron: Texas Instruments, *Commutation techniques for three phase brushless DC motors*

Stel dat we de drie fasedraden u, v en w noemen, dan zouden de spanningen over de fases er uit moeten zien zoals onderstaande afbeelding aan de linkerkant. Dit zou betekenen dat je niet alleen een positieve spanning moet voorzien, maar ook de negatieve tegenhanger. Om dit te versimpelen zodat je enkel de positieve spanning moet voorzien kan je alles verschuiven zodanig dat het hele systeem 0V als referentie gebruikt. Als gevolg moeten de spanningen op de fasedraden (die dus allemaal 0V als referentie hebben) er uit zien zoals op de onderstaande afbeelding aan de rechterkant. Als je daar de spanning op u en v bij elkaar telt (dus de spanning u-v meet), krijg je eigenlijk hetzelfde als een sinusvormige spanning, maar dan zonder negatieve spanning te moeten voorzien.



We can generate this voltage waveform using a PWM driver  
PWM duty cycle ~ applied voltage

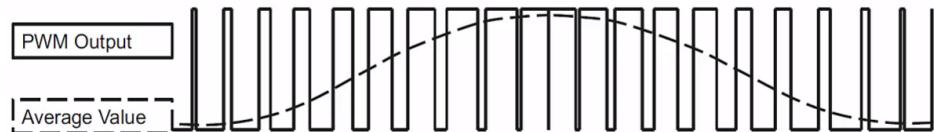


Fig. 8: De referentieshift bij sinusvormige commutatie

Bron: Texas Instruments, *Commutation techniques for three phase brushless DC motors*

Deze speciale spanning op een fasedraad kan door een microcontroller benaderd worden door gebruik te maken van een wisselend PWM-signal. Als je namelijk de duty-cycle met de juiste hoeveelheid en met correcte timing aanpast, zal de gemiddelde spanning de vorm aannemen van een sinusspanning.

Ook al lijkt deze methode veel moeilijker, het is zeker niet onmogelijk en zorgt bovendien voor een constant moment, minder trillingen, minder geluid en een redelijk hoog rendement.

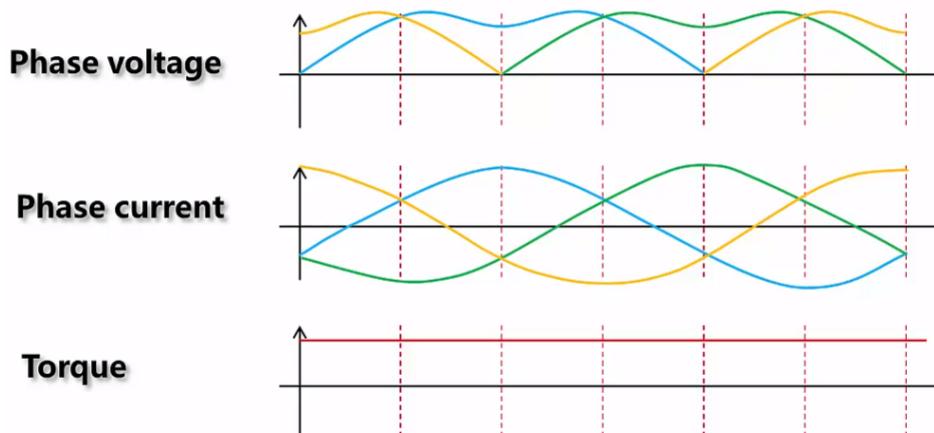


Fig. 9: Overzicht van de karakteristieken, afkomstig van sinusvormige commutatie

Bron: Texas Instruments, *Commutation techniques for three phase brushless DC motors*

*Naar:* Texas Instruments, *Commutation techniques for three phase brushless DC motors*

**Feedbackmechanismes** Als je kijkt naar de trapeziumvormige commutatie, kan je onmiddelijk zien dat het zeer belangrijk is om op het juiste moment de fasespanningen te veranderen. Stel dat je de fasespanningen wijzigt vooraleer de rotor op de juiste plaats gekomen is, zal die rotor haperen en waarschijnlijk helemaal niet draaien. Bij sinusvormige commutatie is dit ook belangrijk, maar omdat de spanningen geleidelijk aan veranderen (volgens een sinus) is het minder kritisch.

Als je enkel wilt dat de motor op een constante snelheid draait (dus dat de hardware zich gedraagt als een frequentieregelaar) dan hoef je geen externe feedback zoals een encoder aan te sluiten, maar dan werk je het beste met de trapeziumvormige commutatie. Bij dat soort commutatie is het namelijk zo dat er telkens één van de fases niet onder spanning staat. Maar als een permanente magneet langs de spanningsloze wikkeling passeert, dan zal die in de wikkeling zelf een spanning opwekken. Dit noemen we terug-EMK. Op deze manier kan de hardware in beperkte mate meten wat de relatieve rotatie is tussen de rotor en de stator en zo bepalen of dat het al dan niet tijd is om de fasespanningen te wijzigen. Het enige nadeel bij deze methode is dat het enkel werkt bij hoge snelheden. Dit komt omdat de amplitude van die opgewekte spanning (terug-EMK) recht evenredig is met de snelheid waarmee de rotor veldlijnen snijdt. Als de snelheid te laag is, zal die terug-EMK ook onmeetbaar laag zijn.

Aangezien ik op elke motor een incrementele encoder gezet heb, kan de hardware met een hoge precisie (in mijn geval  $0.15^\circ$ ) meten wat de relatieve rotatie is van de rotor en de stator. Dit is zeer belangrijk als je de sinusvormige commutatiemethode wil gebruiken en tegelijkertijd aan closed-loop positiecontrole wil doen. Dit zorgt er ook voor dat ik zowel op hoge als op zeer lage snelheden kan werken en beter rekening kan houden met dynamische belastingen.

**Calibratie** Het enige nadeel van het gebruiken van een encoder is het feit dat je die bij elke start moet calibreren. De hardware weet bij de start immers niet wat de begintoestand is van relatieve rotatie tussen de rotor en de stator. Als je bijvoorbeeld vraagt om de motor 5 rotaties te laten draaien, weet de hardware dus ook niet met welke fasespanningen hij moet beginnen. De hardware die ik hiervoor gebruik biedt een calibratiefunctie aan waarbij niet alleen die relatieve rotatie bepaald wordt, maar ook gekeken wordt ofdat de encoder op een correcte manier werkt. De commutatiemethode gedurende de calibratie is nog minder efficiënt dan trapeziumvormige commutatie, wat wil zeggen dat de motor eigenlijk niet belast mag worden (of dat de belasting op zijn minst constant moet blijven in elke draairichting). Daarom laat ik mijn robotarm altijd in zijn verticale toestand starten.

### 1.3.3 Turnigy Aerodrive SK3 4250 - 350Kv

Deze outrunner van Turnigy heb ik gebruikt voor as 1, as 2 en as 3. De motor heeft een kostprijs van ongeveer 40 euro en wordt uitsluitend verkocht bij Hobbyking. Voor die prijs is het wel een goede motor en eigenlijk geldt dit voor de hele Aerodrive SK3 reeks.

Deze motor heeft de volgende specificaties:

Gewicht	423g	Kv	350 rpm/V
Poolparen	7	Piekvermogen	1190W
Piekstroom	53A	Piekspanning	19V

Tab. 1: Specificaties van Turnigy Aerodrive SK3 4250 - 350Kv

De afmetingen zijn te vinden in bijlage G.

Deze motor is met een buitendiameter van 42mm de grootste die ik in mijn robotarm gebruik. De drie gebruikte motoren zijn dan ook verantwoordelijk voor de drie zwaarst

belaste assen van de robotarm. Deze motor moet je langs de bovenkant bevestigen en heeft maar één echte uitgaande as, maar toch kan je langs beide kanten een beweging afnemen.



Fig. 10: Turnigy Aerodrive SK3 4250 - 350Kv

Bron: HobbyKing, *Turnigy Aerodrive SK3 - 4250-350KV Brushless Outrunner Motor*

*Naar: HobbyKing, Turnigy Aerodrive SK3 - 4250-350KV Brushless Outrunner Motor*

#### 1.3.4 Propdrive v2 3536 910Kv

Deze outrunner van Propdrive heb ik gebruikt voor as 4, as 5 en as 6. De motor heeft een kostprijs van ongeveer 22 euro en wordt uitsluitend verkocht bij Hobbyking. Deze motor is van mindere kwaliteit als de Turnigy Aerodrive SK3 reeks, maar voor de prijs werkt hij zeer goed.

Deze motor heeft de volgende specificaties:

<b>Gewicht</b>	216g	<b>Kv</b>	910 rpm/V
<b>Poolparen</b>	7	<b>Piekvermogen</b>	350W
<b>Piekstroom</b>	38A	<b>Piekspanning</b>	17V

Tab. 2: Specificaties van Propdrive v2 3536 910Kv

Deze motoren zijn verantwoordelijk voor de minder belaste assen van de robotarm. De hogere assen (as 4, 5 en 6) zouden ook een hogere snelheid moeten hebben dan de andere assen. Daarom dat ik gekozen heb voor een motor die een Kv-waarde heeft van 910 rpm/V. De Kv-waarde geeft theoretisch weer hoe snel de motor zal draaien per aangelegde volt. Dat gaat natuurlijk ten koste van het maximaal moment dat uitgeoefend kan worden, maar dat is geen probleem aangezien de hogere assen minder zwaar belast worden.

Deze motor moet je net zoals de grotere motor langs de bovenkant bevestigen. Bij deze motor is het ook mogelijk om langs beide kanten een beweging af te nemen.



Fig. 11: Propdrive v2 3536 910Kv

Bron: HobbyKing, *PROPDRIVE v2 2830 800KV Brushless Outrunner Motor*

Naar: HobbyKing, *PROPDRIVE v2 2830 800KV Brushless Outrunner Motor*

## 2 Eerste iteratie: conceptueel

### 2.1 Inleiding

In dit deeltje zal ik het hebben over de eerste iteratie van de robotarm. De eerste iteratie kan gezien worden als een conceptuele iteratie. Er werd geen aandacht besteed aan de plaatsing van de motoren en de encoders en ik wist nog niet zeker welke overbrenging ik ging toepassen. Maar dat was ook niet de bedoeling van deze iteratie. Hiermee kon ik namelijk verschillende soorten overbrengingen testen om te zien dat ze wel praktisch haalbaar zouden zijn. Ook kon ik zo een beeld krijgen van hoe de robotarm er uit zal zien, of hoe ik de assen moet tekenen om zo veel mogelijk bewegingsvrijheid te verkrijgen.

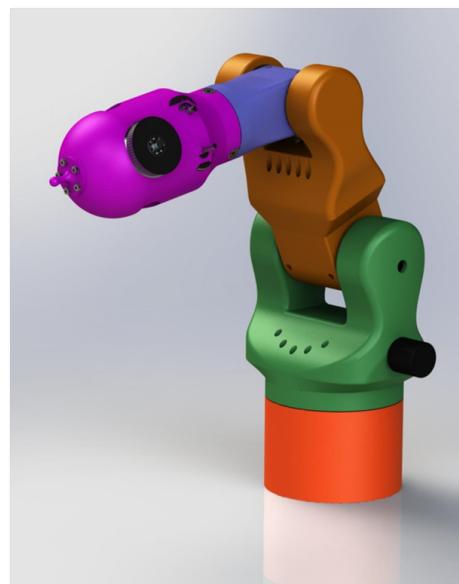


Fig. 12: De assembly van de eerste iteratie

## 2.2 As 1

Voor de eerste as had ik origineel het idee om met inwendige vertanding te werken. Dit heeft het grote voordeel dat de tweede as (het groene stuk in onderstaande afbeelding) langs alle kanten ondersteund kan worden door kogellagers. Dit zou voor een hele grote stabiliteit zorgen voor een toch wel redelijk zwaar belaste as.

Ook is het middelste stuk van de as open. Dit heeft als voordeel dat ik de bekabeling van de vijf andere assen langs deze weg kan wegleiden. Dit is niet alleen mooier, maar geeft me ook meer rotatievrijheid en spaart kabel uit.

Om die zogenaamde stabiliteit te garanderen, wordt het middenstuk zijdelings ondersteund door een grote kogellager, namelijk het type 16014 met een binnendiameter van 70mm. Verder zou de bovenkant ondersteund worden door kogellagers van het type 605-ZZ, en de onderkant met kogellagers van het type 623-ZZ.

Je kan zien aan de tekening dat ik sterk overwogen heb om dit ontwerp in de echte robotarm te steken. Er was al rekening gehouden met de schroeven die moeten gebruikt worden en de eerste as was ook al opgesplitst in verschillende onderdelen om het realiseerbaar te maken. In de derde iteratie zal je zien dat het idee van de kogellagers grotendeels gebleven is en zelfs is doorgetrokken naar de andere assen.

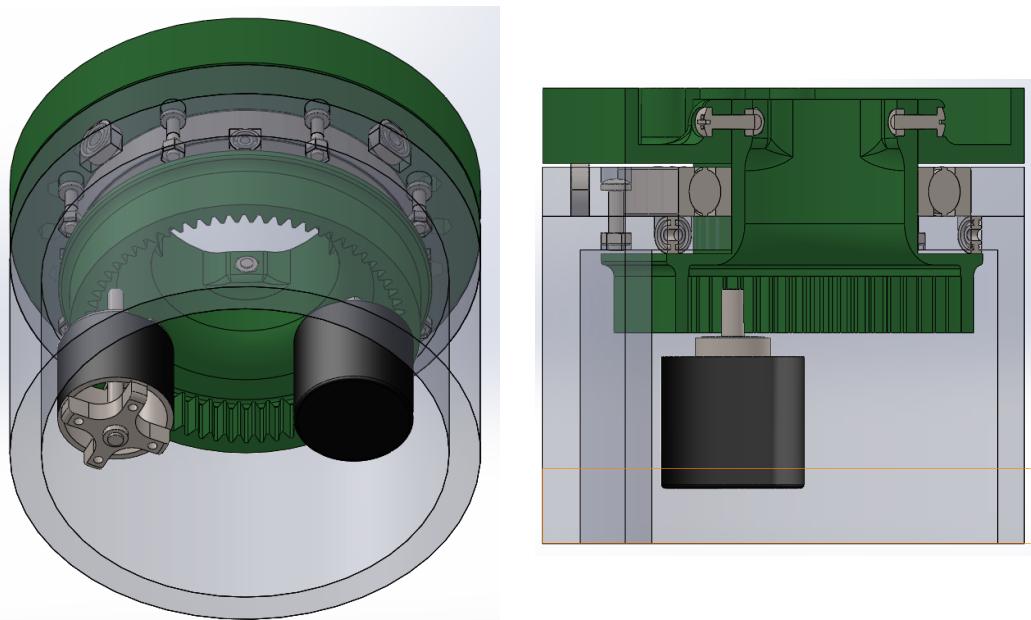


Fig. 13: De eerste as van de eerste iteratie

De voornaamste reden waarom dat ik deze overbrenging uiteindelijk niet gebruikte is omdat ik met inwendige vertanding een te kleine overbrengingsverhouding kon krijgen, zeker wanneer je de benodigde ruimte in rekening brengt.

## 2.3 As 2

In de tweede as zie je dat ik het idee had om de derde as langs beide kanten te ondersteunen met kogellagers van het type 6812-2RS. De motor die ik hiervoor zou gebruiken kan langs beide kanten een beweging overbrengen (zie hoofdstuk 1.3.3). Deze

beweging zou dan via beide kanten overgebracht worden met een riemoverbrenging.

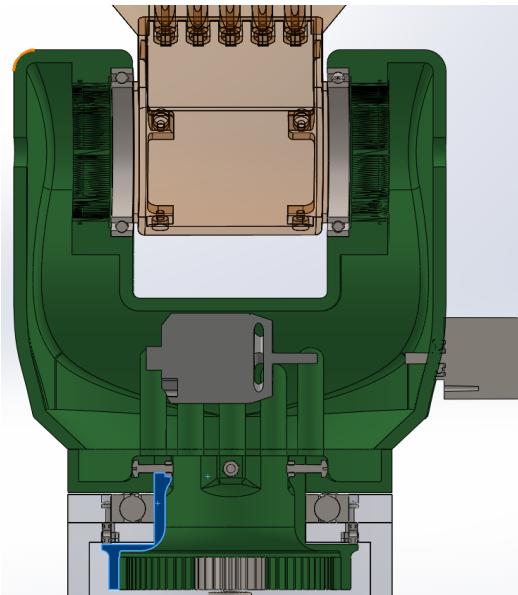


Fig. 14: Doorsnede van de tweede as van de eerste iteratie

Dat zou wel betekenen dat de encoder langs de buitenkant gehangen moest worden. Het was echter niet echt haalbaar om die riemoverbrenging in de behuizing te steken. Ook had ik mijn twijfels over hoe groot ik de overbrengingsverhouding kon maken. Daarom heb ik niet geopteerd voor dit ontwerp.

## 2.4 As 3

De derde as is zeer soortgelijk aan as 2. Het zou gebruik maken van dezelfde motor en dezelfde technieken. Alleen zijn de kogellagers die de vierde as ondersteunen kleiner. Dit ontwerp is niet gekozen om dezelfde redenen dan bij as 2. Het langs beide kanten ondersteunen van de vierde as is ook een grote beperking op de bewegingsvrijheid van de derde as. Om dat op te lossen zou ik deze as veel langer moeten maken, maar dan zou er een nog grotere belasting op de tweede as komen.

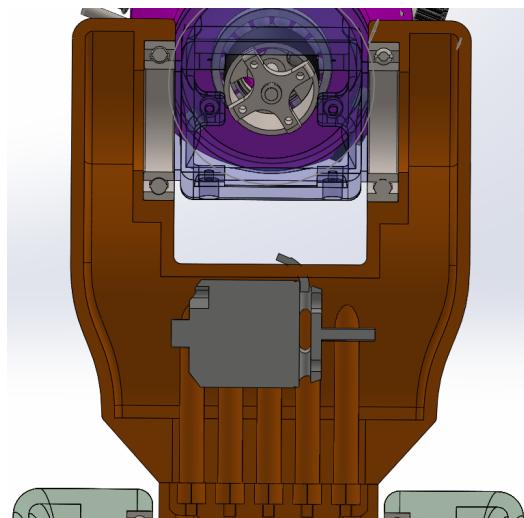


Fig. 15: Doorsnede van de derde as van de eerste iteratie

## 2.5 As 4

De vierde as wou ik op hetzelfde principe baseren als as 1. Maar de interne vertanding heeft zeer veel plaats nodig. Plaats die ik bij de vierde as zeker niet heb. Ik was er nog niet uit hoe ik dit ging oplossen, maar in deze as heb ik toch al een deel van het stabilisatiesysteem van de eerste as kunnen toepassen. Meerbepaald de vier kleine kogellagers en de grotere centrale kogellager dat de vijfde as ondersteund.

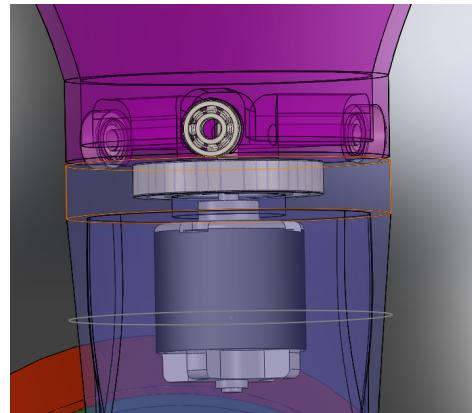


Fig. 16: Doorsnede van de vierde as van de eerste iteratie

## 2.6 Differentieel

Een differentieel is een stelsel van tandwielen waarmee met één aandrijvende as meerdere rotaties met verschillende snelheden afgetakt kunnen worden. Wel is het zo dat de gemiddelde snelheid van de aangedreven assen gelijk zal zijn aan de gemiddelde snelheid van de aandrijvende as.

**Rechtdoor rijdende auto** In onderstaand differentieel wordt het ringwiel (paars) door de motor aangedreven. De paarse kooi draait mee. Het in de kooi gemonteerde vleugel wiel (groen) geeft het vermogen door aan beide zijwielen (rood en geel), die verbonden zijn met de assen en dus met de achterwielen (of voorwielen). Rijdt de auto rechtaan, dan draaien deze wielen met dezelfde snelheid. Het vleugelwiel draait met de kooi mee, maar draait niet om zijn as.

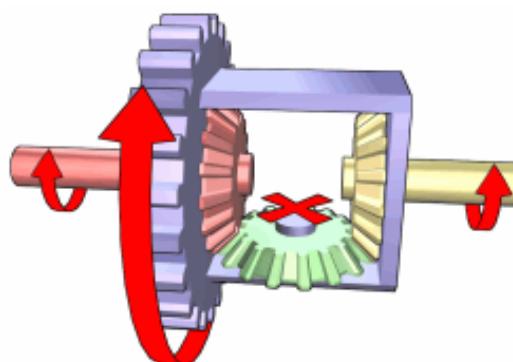


Fig. 17: Differentieel bij rechtaan rijden

Bron: Wikipedia, *Differentieel (werktuigbouwkunde)*

**Wanneer één wiel van de auto geblokkeerd is** In onderstaand differentieel zal het vleugelwiel (groen) gedwongen over de tanden van het geblokkeerde tandwiel gaan lopen. Hierdoor zal het niet geblokkeerde (gele) wiel met dubbele snelheid worden aangedreven, namelijk de snelheid waarmee de kooi (paars) rondwentelt plus de snelheid van het om zijn as draaiend vleugelwiel.

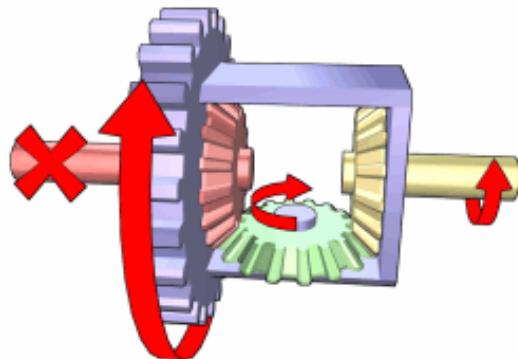


Fig. 18: Differentieel wanneer één wiel van de auto geblokkeerd is

Bron: Wikipedia, *Differentieel (werktuigbouwkunde)*

Naar: Wikipedia, *Differentieel (werktuigbouwkunde)*

## 2.7 As 5 en as 6

Voor as 5 en 6 had ik een apart idee. Ik wou namelijk het 'omgekeerde' van een differentieel toepassen. De motoren zouden aan de assen van de 'wielen' bevestigd moeten worden via een riemoverbrenging. De as dat normaal gezien het aandrijvende werk verricht (de drijver) dankzij de automotor, wordt nu de as die aangedreven wordt (de volger). Afhankelijk van hoe de twee motoren relatief tot elkaar draaien, zal de aangedreven as (de grijze cilinder die uit het paars gedeelte komt op onderstaande foto) rond zichzelf draaien (namelijk A6) of op en neer gaan langs het paarse stuk (namelijk A5). Dit is een zeer robuuste methode om die twee bewegingen te realiseren. De overbrengingsverhouding is misschien niet zo groot, maar je hebt wel twee samenwerkende motoren die dus samen een groter vermogen leveren.

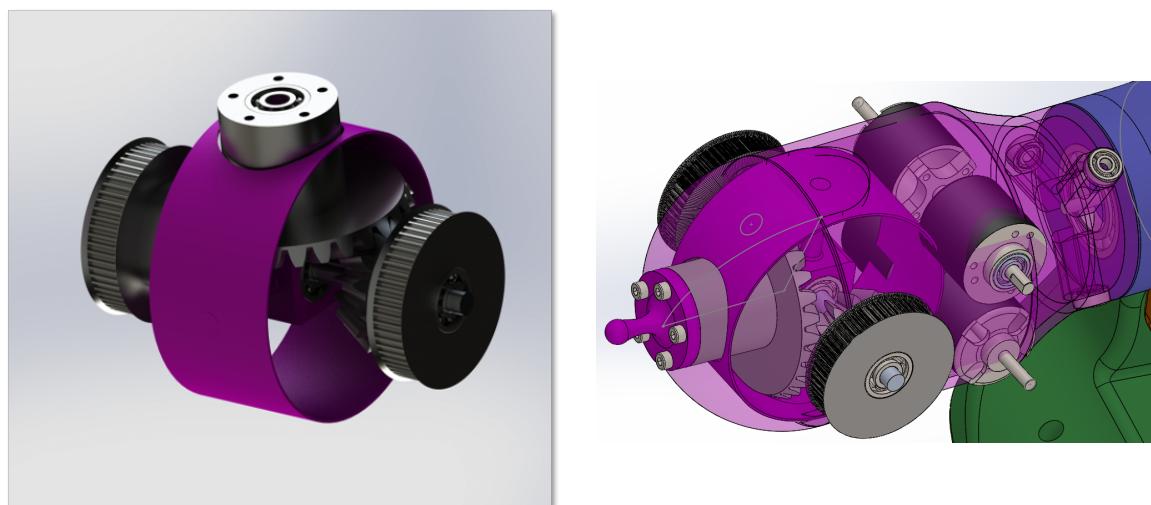


Fig. 19: As 5 en 6 van de eerste iteratie

Een veelbelovend concept dus, maar je hebt enorm veel plaats nodig om de twee motoren en de twee encoders te plaatsen, waardoor het uiteinde van de robotarm er niet alleen belachelijk uitziet, maar ook veel te zwaar en lomp is.

## 3 Tweede iteratie: de doorbraak

### 3.1 Inleiding

De tweede iteratie was een heuse doorbraak. Hier heb ik besloten welk type overbrenging ik ging gebruiken en heb ik deze ook kunnen doortrekken naar alle assen. In deze iteratie is er ook veel meer aandacht besteed aan de praktische realiseerbaarheid en ook aan vormgeving.

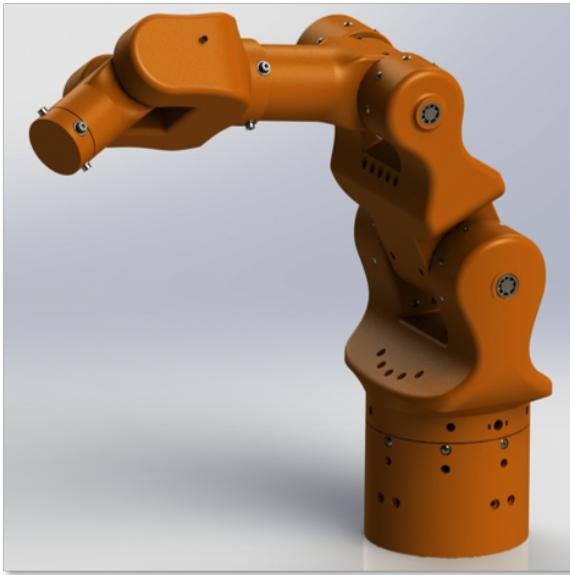


Fig. 20: De assembly van de tweede iteratie

Aangezien de tweede iteratie snel vervangen was door de derde iteratie, heb ik niet elke as uitgewerkt.

### 3.2 Planeetwielmechanisme

Een planeetwielmechanisme is een stelsel van tandwielen waarbij de aangedreven as op dezelfde lijn ligt als de aandrijvende as. Dit systeem heeft het grote voordeel om redelijk compact grote overbrengingsverhoudingen te voorzien. Omdat er ook meerdere raakvlakken zijn, kan het ook grote vermogens overbrengen.

Het heeft zijn naam te danken aan de overeenkomsten met ons zonnestelsel. Het gele tandwiel in onderstaande figuur wordt het zonnewiel genoemd, de blauwe tandwielen zijn de planeetwielen die allen bevestigd zijn op een planeetwieldrager en het rode tandwiel met innerlijke vertanding is een satellietwiel.

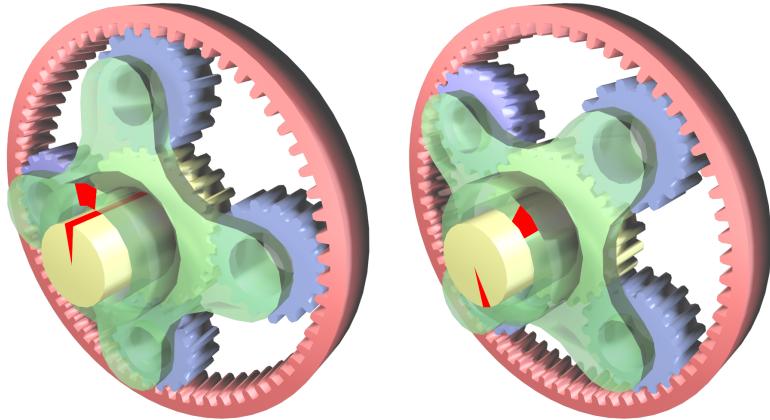


Fig. 21: Een simpel planeetwielmechanisme

Bron: Wikipedia, *Planeetwielmechanisme*

Afhankelijk van welke tandwielen je aandrijft en welke tandwielen aangedreven worden krijg je verschillende overbrengingsverhoudingen:

- Als je het zonnewiel gebruikt als aandrijvende as, het satellietwiel vasthoudt en de planeetwieldrager als aangedreven as gebruikt, krijg je een snelheidsreductie.
- Als je de planeetwieldrager aandrijft, het zonnewiel vasthoudt en het satellietwiel als aangedreven as gebruikt, krijg je een versnelling.
- Het aandrijven van het zonnewiel terwijl de planeetwieldrager vastgehouden wordt en het satellietwiel als aangedreven as gebruikt wordt heeft een omkering van de beweging als gevolg.
- Het tesamen aandrijven van het zonnewiel en de planeetwieldrager met het satellietwiel als aangedreven as zorgt voor een eenvoudige 1:1 overbrengingsverhouding.

Hoe groot de overbrengingsverhouding is bij een snelheidsreductie of versnelling is afhankelijk van de verhouding tussen de groottes van de tandwielen. De exacte formules voor het bepalen van de overbrengingsverhoudingen zijn te vinden in bijlage B.

*Naar: Wikipedia, Planeetwielmechanisme*

### 3.3 Compound Planetary Gear Box

Uiteindelijk heb ik besloten om alle assen te voorzien van een speciaal planeetwielmechanisme, gebaseerd op een ontwerp van Gear\_Down\_For\_What. Dit tandwielmechanisme is eigenlijk een combinatie van twee planeetwielmechanismen. Eén van de ringen moet vastgezet worden, de andere ring is de aangedreven as en afhankelijk van welk zonnewiel je als aandrijvende as gebruikt, zal de overbrengingsverhouding 50.4:1, -49.4:1, -38:1 zijn of 39:1. In bijlage B vind je een theoretische beschrijving van dit mechanisme en een bewijs voor de overbrengingsverhoudingen.

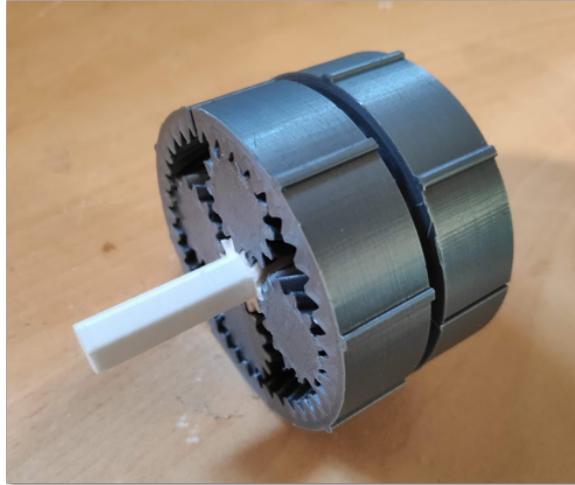


Fig. 22: Het ontwerp van Gear\_Down\_For\_What

Dankzij het ontwerp kan je deze overbrenging ook maar langs één kant gebruiken. Het is dus onmogelijk om de aangedreven ring te gebruiken als aandrijvende as en het zonnewiel als aangedreven as. Dit heeft het grote voordeel dat ik geen houdkoppel nodig heb van de motoren, aangezien deze overbrenging dit voorziet. Daarom blijft de warmteontwikkeling van de motoren redelijk beperkt. Alleen was de grote vraag dat dit mechanisme wel sterk genoeg is om dit houdkoppel te voorzien, zeker bij de kleinere naar beneden geschaalde varianten van as 4, 5 en 6.

Dit mechanisme bestaat echter uit tandwielen met V-vertanding. Hierdoor neemt het rendement toe en ook de maximale kracht die overgebracht kan worden. Dit gecombineerd met het voordeel van planeetwielmechanismes dat er zeer veel raakvlakken zijn zorgt ervoor dat dit mechanisme sterk genoeg is.

Het mechanisme is volledig geprint in PLA en heeft normaal gezien een steekdiameter van 50mm. Ik heb echter aangepaste versies gemaakt met een steekdiameter van 70mm en 40mm.

Het enige probleem van dit mechanisme is de hoeveelheid speling. Speling is de ruimte tussen bewegende onderdelen. Dat is nodig om de bewegende onderdelen correct met elkaar te laten werken, maar te veel aan speling kan een nadelige impact hebben op de accuraatheid. Speling kan ook vergroten als gevolg van slijtage. Dit mechanisme heeft van nature al redelijk veel speling. Dit komt omdat niet elke 3D-printer dezelfde toleranties heeft, niet elke rol PLA van dezelfde kwaliteit is en andere factoren zoals luchtvochtigheid, printtemperatuur en printsnelheid ook een grote invloed kunnen hebben. De overbrengingen gaan ook gedurende lange periodes aan meer dan 4000RPM moeten draaien, dus het valt af te wachten hoe veel ze afslijten op lange termijn.

De speling is op te lossen door de twee ringen op een bepaalde afstand van elkaar te fixeren. Maar dit is geen efficiënte of stabiele manier. De beste methode om speling op te lossen is door een groot experiment uit te voeren waarbij ik met mijn 3D-printer, met het door mij gebruikte merk PLA, verschillende toleranties in de tekeningen toepas en daarna uittest. Echter had ik hiervoor geen tijd noch zin, dus heb ik (tijdelijk) geopteerd voor de minder efficiënte methode. In de volgende hoofdstukken zal ik naar deze overbrenging verwijzen als de hoofdoverbrenging.

### 3.4 As 1

De hoofdoverbrenging van de eerste as is een geschaalde versie van het samengesteld planeetwielmechanisme. Het is meerbepaald 1.167 groter dan het origineel, namelijk met een buitendiamter van 70mm. Het idee was dat de motor (links) aan de zijkant bevestigd zou worden en zijn beweging zou doorgeven via een normale tandwieloverbrenging. De encoder zou in het midden zitten. Uiteindelijk ben ik hier van afgestapt uit schrik dat de tandwieloverbrenging tanden zou overslaan dankzij het grote vermogen dat overgebracht moet worden.

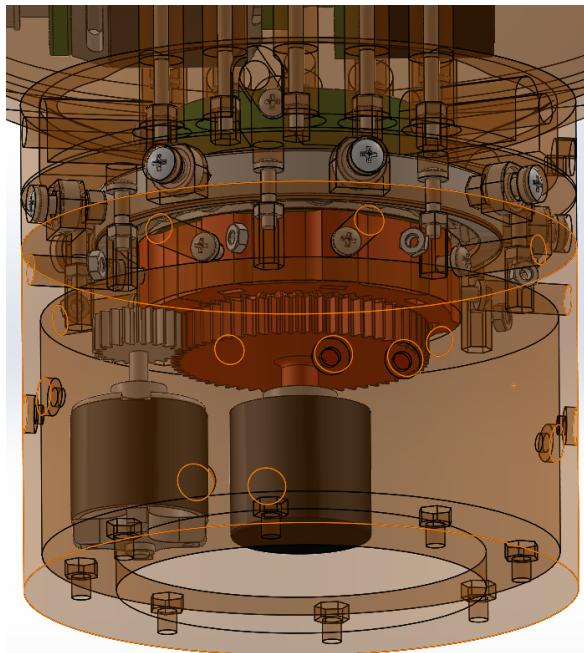


Fig. 23: As 1 van de tweede iteratie

Wel heb ik het stabilisatiesysteem verder uitgewerkt. Dit systeem dat je op onderstaande afbeelding kan zien is nog steeds van toepassing in de nieuwste iteratie.

As 1 en de verbinding tussen as 1 en as 2 bestaat uit verschillende delen. Het bovenste oranje stuk is de basis voor de tweede as op te monteren. Het middelste oranje stuk is verantwoordelijk voor de stabilisatie van de as. Het onderste oranje stuk is de basis van de eerste as. Het groene gedeelte is de aangedreven as van de hoofdoverbrenging en het rode gedeelte is de vastgezette ring.

Zoals je ziet wordt de groene ring door de grote kogellager niet alleen zijdelings ondersteund maar wordt de groene ring ook verticaal ingeklemd. Dit zorgt ervoor dat niet alleen de wringing maar ook de opwaartse bewegingen opgeheven wordt. Om de wringing nog beter te compenseren wordt het bovenste oranje deel ondersteund door acht kleinere kogellagers (type 605-ZZ) die verwerkt zitten in het middelste oranje deel.

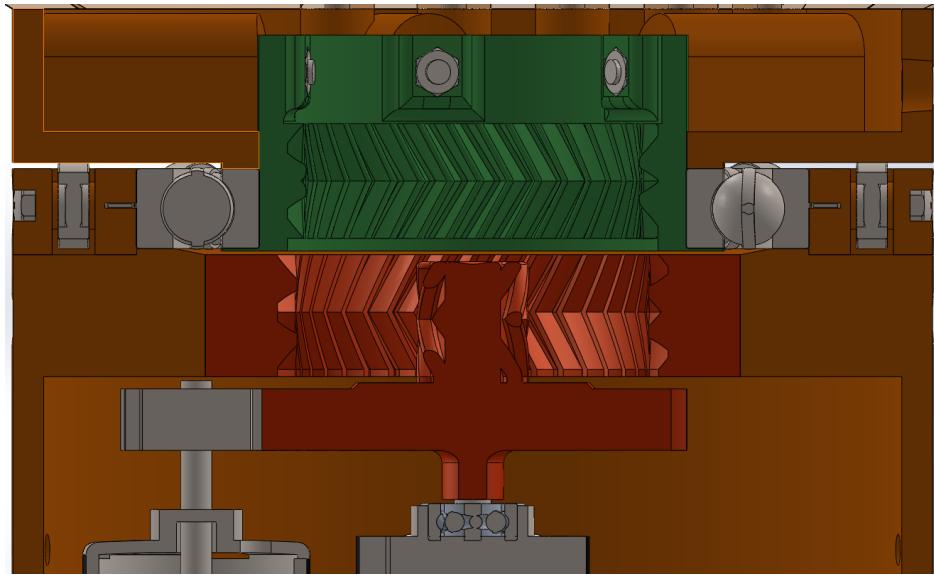


Fig. 24: Doorsnede van as 1 van de tweede iteratie

### 3.5 As 2

De derde as is nog steeds langs beide kanten ondersteund door de tweede as. Maar nu zijn de hoofdoverbrengingen langs binnen aangedreven door de blauwe zonnewielen die bevestigd zijn op een algemene metalen as. Deze metalen as wordt dan via een riemoverbrenging aangedreven door de motor. De encoder is op de achterkant van de motor bevestigd. Je kan ook zien dat er aan de derde as randjes voorzien zijn die zijwaartse druk zetten op de kogellagers. Dit is om speling te beperken. Deze techniek zit nog altijd in de huidige iteratie en werkt zoals verwacht. De andere zonnewielen van de planeetwielmechanismes (niet de blauwe zonnewielen) hebben een gat in het midden dat er voor zorgt dat de metalen as door kan.

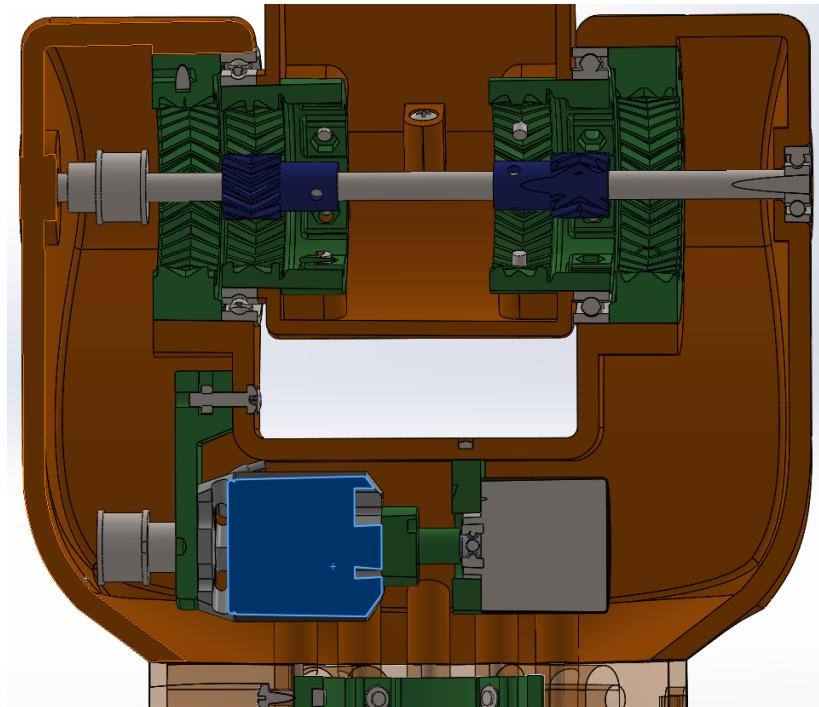


Fig. 25: Doorsnede van as 2 van de tweede iteratie

### 3.6 As 3

De derde as is sterk gebaseerd op de tweede as. Alleen zijn de dubbele planeetwielmechanismen kleiner, net zoals de kogellagers. Toch heb ik uiteindelijk niet geopteerd voor dit ontwerp aangezien de dubbele uitvoering van de hoofdoverbrenging er voor zorgt dat de bewegingsvrijheid beperkt is. Ook had ik twijfels bij de praktische uitvoerbaarheid van dit ontwerp.

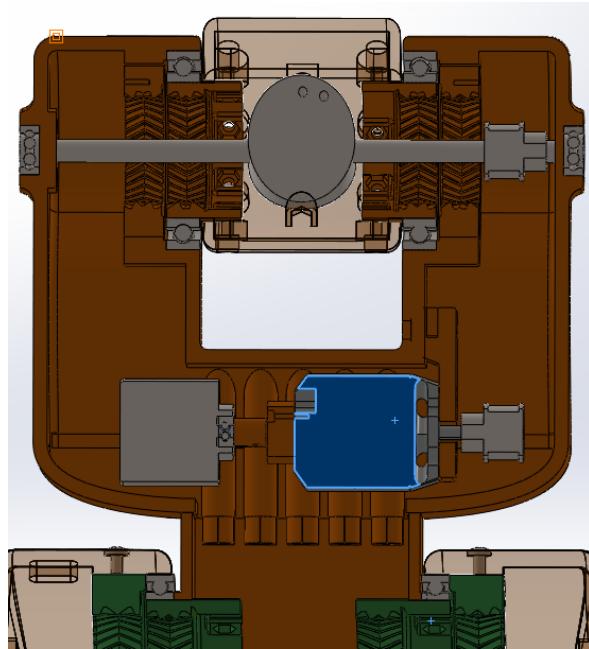


Fig. 26: Doorsnede van as 3 van de tweede iteratie

### 3.7 As 4

Het plaatsprobleem van de vierde as is opgelost omdat ik de hoofdoverbrenging in lijn kon plaatsen met de rotatieas, de motor en de encoder. Dat was ik dus ook van plan. Maar ik had in de tweede iteratie nog niet de tijd genomen om deze al uit te werken. Wel zie je dat ik opnieuw gebruik wil maken van het stabilisatiesysteem van de eerste as, alleen kleiner.

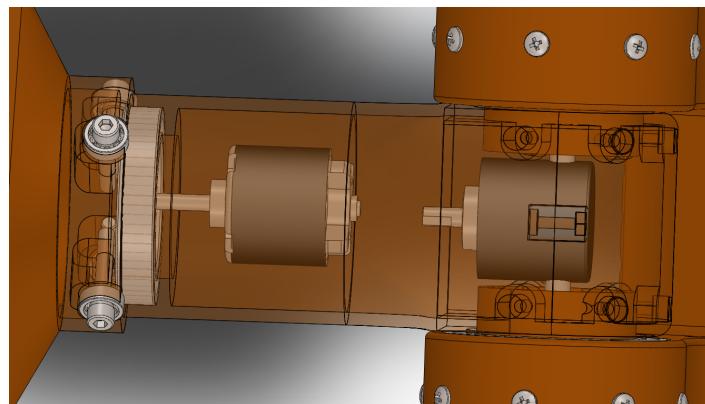


Fig. 27: Doorsnede van as 4 van de tweede iteratie

### 3.8 As 5 en as 6

Origineel wou ik de vijfde as laten werken op dezelfde principes als de derde as. Maar aangezien ik daar nog minder ruimte had, was het zo goed als onmogelijk. Daarom heb ik geen uitwerking gemaakt in de tweede iteratie.

Wel zat het idee om as 5 en as 6 te combineren nog steeds in mijn hoofd. Daarom probeerde ik om twee motoren in de behuizing van de vijfde as te steken en één van de twee motoren toch verantwoordelijk te stellen voor de aandrijving van as 6, wat trouwens vaak toegepast wordt bij echte industriële robotarmen. Echter had ik al snel beslist dat dit niet de beste oplossing was en heb ik er dus ook geen tijd ingestoken.

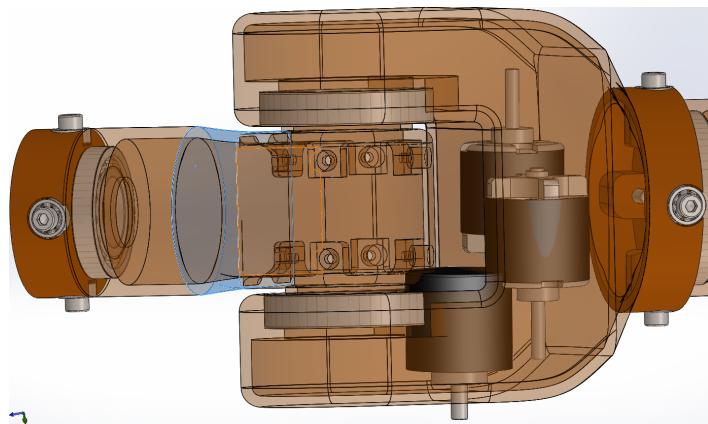


Fig. 28: Overzicht van as 5 en 6 van de tweede iteratie

## 4 Derde iteratie

### 4.1 Inleiding

De derde iteratie is voor de eerste twee assen vooral een herziening van de tweede iteratie op vlak van vormgeving. Voor de andere assen zijn er toch wel grote veranderingen doorgevoerd, zowel op vlak van vormgeving als op vlak van functionaliteit, met het oog op praktische uitvoerbaarheid en bewegingsvrijheid. Deze iteratie is uiteindelijk ook degene die ik in praktijk gebracht heb.

De zesde as heb ik moeten herontwerpen omdat het op vlak van inverse kinematica beter uitkwam, maar het komt uiteindelijk neer op een simpele asverschuiving. De eerste as heb ik ook een aantal keer moeten herontwerpen dankzij een aantal doorgebrande motoren in de prototypes. Uiteindelijk ben ik tot de conclusie gekomen dat ik de belasting op de eerste as fout heb ingeschat en heb ik gekozen om er een grotere motor in te steken. De robotarm is hierdoor ongeveer 2cm langer geworden dan normaal gezien de bedoeling was.

Ook heb ik in deze iteratie werk gemaakt van een baseplate om de robotarm effectief op te bevestigen, een elektronica behuizing, een end-effector interface en een tool. De tool is zuiver om een voorbeeld te geven, want je kan eigenlijk elke tool gebruiken, zolang het maar aan de conventies van mijn end-effector interface voldoet.

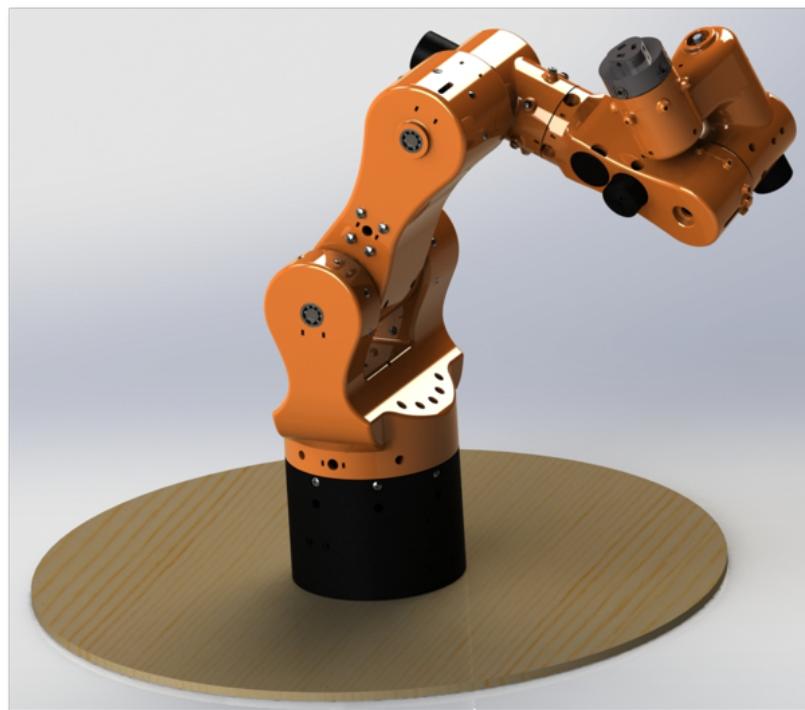


Fig. 29: De assembly van de derde iteratie

### 4.2 Bodemplaat

De bodemplaat bestaat uit twee cirkels. De bovenste cirkel bevat twaalf gaten met een diameter van 4mm die overeenkomen met de bevestigingsgaten in de onderkant van de eerste as. Ook heeft deze bovenplaat een gleuf met als bedoeling de bekabeling van de eerste as langs die weg naar buiten te krijgen.

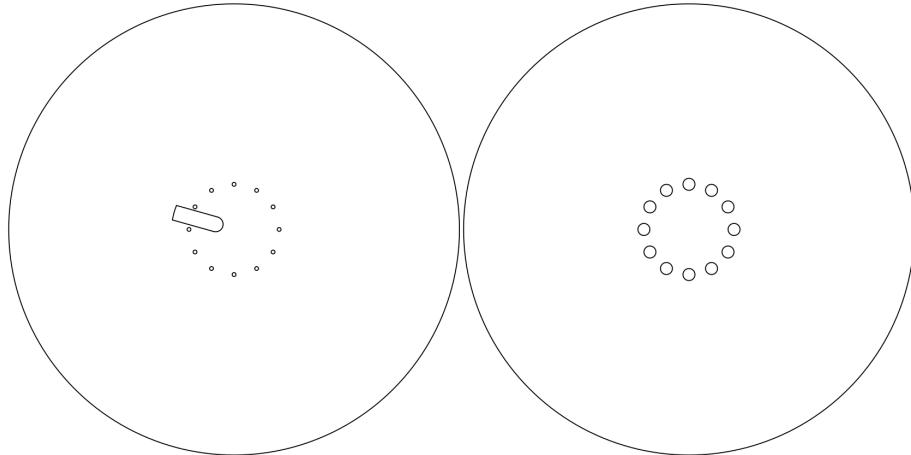


Fig. 30: De bodemplaat: links heb je de bovenplaat, rechts heb je de onderplaat

De onderplaat heeft ook 12 gaten (op dezelfde plaats), maar deze keer met een diameter van 16mm. Dit zorgt ervoor dat ik de kop van de schroeven kan wegwerken in de onderste plaat en de robotarm er dus niet op steunt. Ook kan ik zo rondellen plaatsen dat me in staat stelt om een schroef strakker aan te draaien.

Elke plaat heeft een buitendiameter van 60cm en is met de lasersnijder uitgesneden uit 8mm dik populier. De lasersnijder die ik hiervoor gebruikte is de BRM van FabLab Factory.

Eerst heb ik bovenstaande tekening geëxporteerd naar een dxf-bestand, daarna heb ik dat bestand geïmporteerd in Laserworks, wat op basis van een aantal parameters een tekening omzet naar gcode dat de lasersnijder kan uitvoeren. Vervolgens heb ik het gcode-bestand naar de lasersnijder zelf verplaatst met een USB-stick en uitgelaserd.

Voor de veiligheid heb ik na het uitlaseren een grote waarschuwing op de voorkant gekleefd waarop de maximale werkradius vermeld staat. Deze waarschuwing is met een online tool gegenereerd en een beetje aangepast met Photoshop.

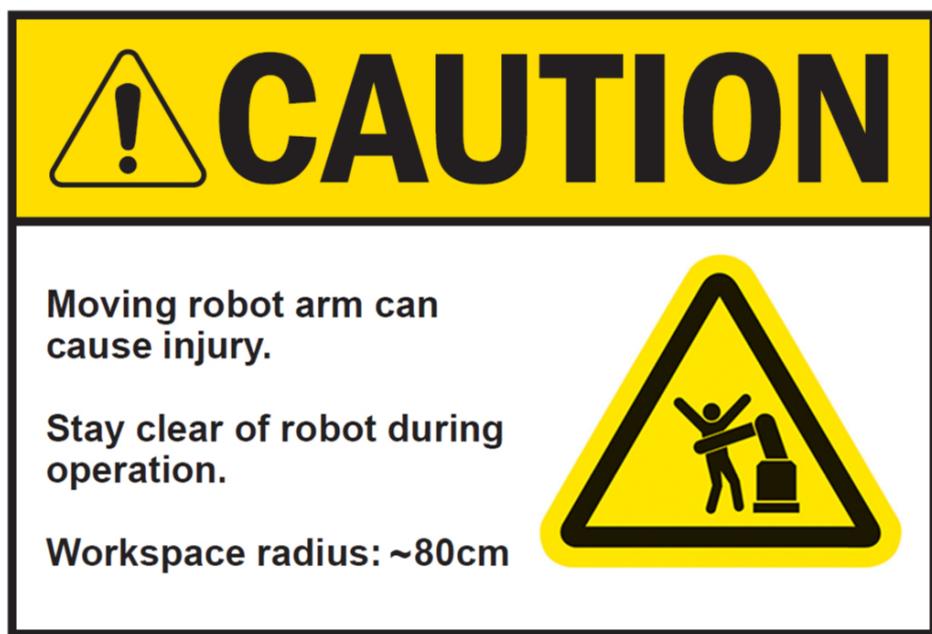


Fig. 31: De waarschuwing op de bodemplaat

### 4.3 As 1

De eerste as heeft eigenlijk nog twee herwerkingen gekend. Dit omdat ik ook twee keer van motor moest veranderen. De eerste herwerking vervangt het tandwielenmechanisme door een belt die het zonnewiel met de motor en de encoder verbond. Dit had ik zo gedaan omdat ik niet echt een manier vond om de motor in het midden te plaatsen en tegelijkertijd de beweging over te brengen naar de encoder. Dit werkte goed, tot het moment dat de motor het begaf wegens te hoge stromen. Na het opnieuw openen van de as merkte ik ook dat de plaats tussen de verschillende componenten te klein was. Dit had als gevolg dat er te veel wrijving ontstond waardoor de geprinte tandwielen begonnen te vervormen als gevolg van warmte.

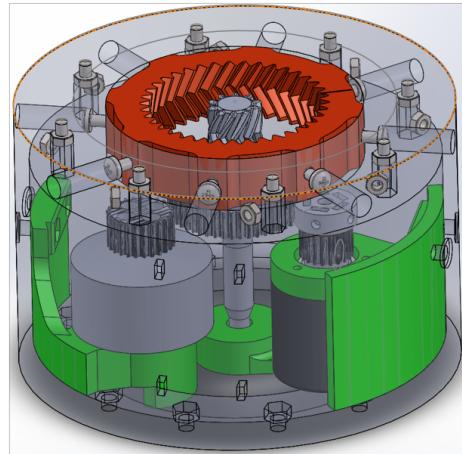


Fig. 32: De eerste herwerking van de eerste as van de derde iteratie

Dat brengt me direct bij de reden van de tweede herwerking. Deze herwerking zorgde ervoor dat ik een grotere motor kon plaatsen. Momenteel is de motor van hetzelfde type als die van as 2 en as 3. Na een aantal experimenten uit te voeren kan ik besluiten dat de temperaturen van de motor meer dan redelijk blijven. Deze herwerking heeft wel als neveneffect dat de robotarm ongeveer 2cm langer is geworden. De uiteindelijke overbrengingsverhouding van de eerste as is 50.4:1.

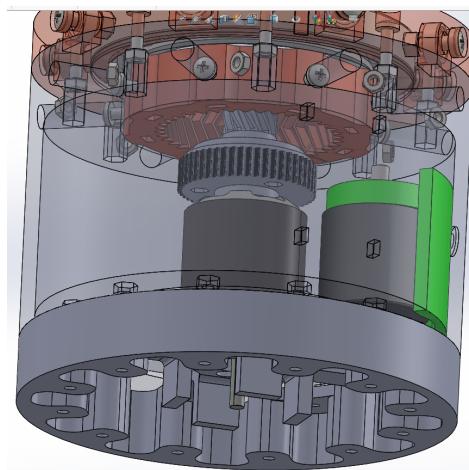


Fig. 33: De tweede herwerking van de eerste as van de derde iteratie

Het grijze stuk staat onder de originele basis van de eerste as. Dat grijs gedeelte is nodig om te compenseren voor die grotere motor en dient ook alshouder waar de motor

op gemonteerd kan worden. Langs de achterkant van de motor (wat ook gebruikt kan worden als uitgang) is er een combinatie bevestigd van het aandrijvende zonnewiel en een riemschijf van het type HTD3mm. Deze riemschijf zal door middel van een riem de beweging overbrengen naar de encoder die rechts tegen de zijkant gemonteerd is. Het stabilisatiesysteem is nog steeds hetzelfde als bij de tweede iteratie.

#### 4.4 As 2

Buiten een kleine aanpassing aan de vormgeving ten voordele van bewegingsvrijheid is er eigenlijk niets verandert aan de tweede as ten opzichte van de tweede iteratie. De overbrengingsverhouding is -49.4:1.

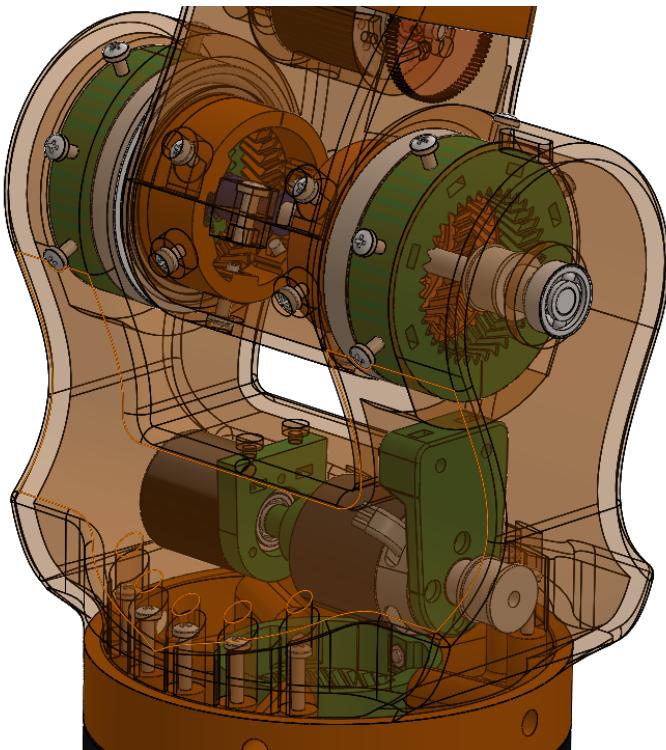


Fig. 34: As 2 van de derde iteratie

#### 4.5 As 3

De derde as heeft wel een drastische herwerking ondergaan. Om de bewegingsvrijheid zo groot mogelijk te maken, moest ik ervoor zorgen dat de derde as maar één steunpunt biedt voor de vierde as. Dit heeft als gevolg dat ik de encoder onder de motor moet plaatsen en de motor met de encoder moet verbinden via een tandwieloverbrenging. Normaal gezien zou je dit moeten vermijden, maar de accuraatheid is nog steeds redelijk hoog omdat de tandwielen een lage modulus hebben. Dit is niet goed voor het maximale moment dat overgebracht kan worden, maar dat is hier niet nodig. Verder heeft de motor ook nog een riemoverbrenging naar een algemene as, dat op zijn beurt de hoofdoverbrenging aandrijft. Aan de hoofdoverbrenging is ook iets speciaals. De gedreven ring (blauw) is namelijk dubbel uitgevoerd. Dit heb ik ontworpen met als doel om zo een groter vermogen te kunnen overbrengen. De overbrengingsverhouding is aan de kleine kant, namelijk -38:1. In de toekomst ga ik dit verhogen naar 50.4:1.

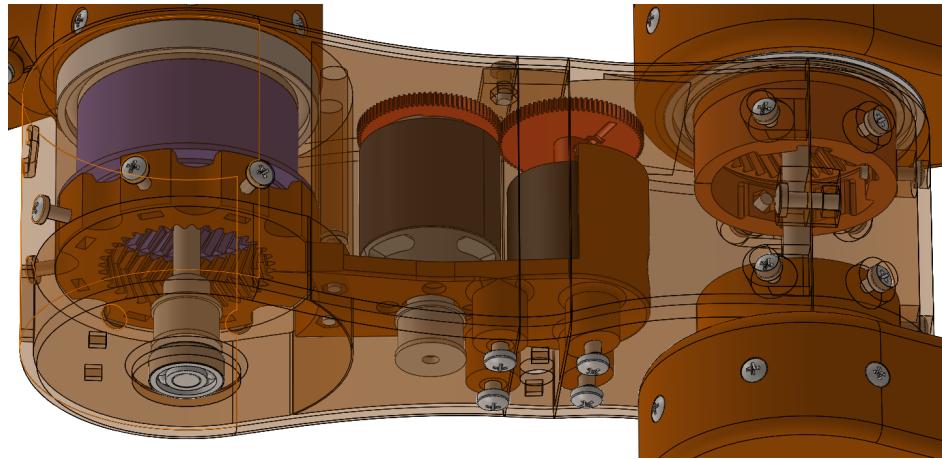


Fig. 35: As 3 van de derde iteratie

In de onderstaande afbeelding zie je een doorsnede van de dubbele uitvoering van de gedreven ring. Je kan ook opnieuw het stabilisatiesysteem herkennen met behulp van een grote kogellager. Eerst had ik het idee om de kogellager zelf ook dubbel uit te voeren, maar het was te moeilijk om deze tweede kogellager gemonteerd te krijgen. Daarom heb ik maar met één kogellager en een soort van placeholder gewerkt om druk uit te oefenen op de kogellager en de vastgezette ring. Dit heeft wel als gevolg dat er redelijk veel spelting op deze as zit.

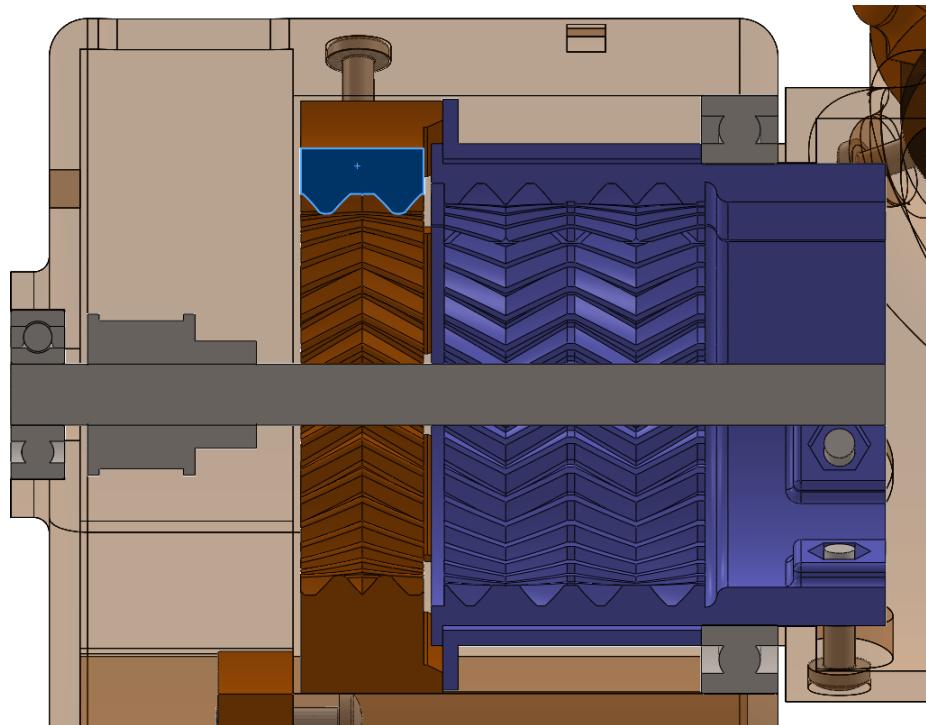


Fig. 36: Doorsnede van as 3 van de derde iteratie

## 4.6 As 4

De vierde as is sterk gebaseerd op de eerste as. De overbrenging wordt net zoals bij de eerste as niet alleen zijdelings ondersteund door een kogellager, maar ook nog eens verticaal door kleinere kogellagers (605-ZZ). Het enige verschil met de eerste as is dat de kogellagers nu bevestigd zijn in de volgende as (as 5) in plaats van in de as zelf (as 4).

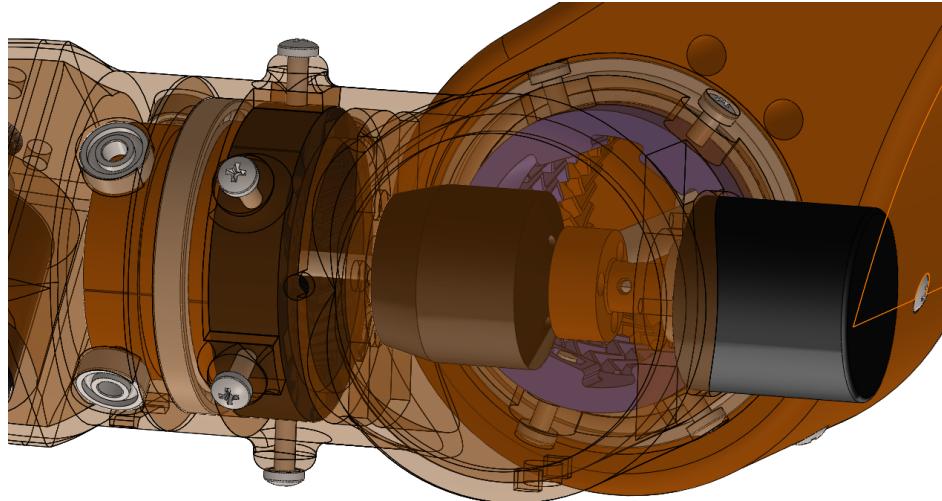


Fig. 37: As 4 van de derde iteratie

De motor is direct in lijn bevestigd met de geschaalde hoofdoverbrenging met een buitendiameter van 50mm. Dit zorgt voor een overbrengingsverhouding van 50.4:1. De encoder is op de achterkant van de motor bevestigd en buiten de robotarm geplaatst voor extra bewegingsvrijheid.

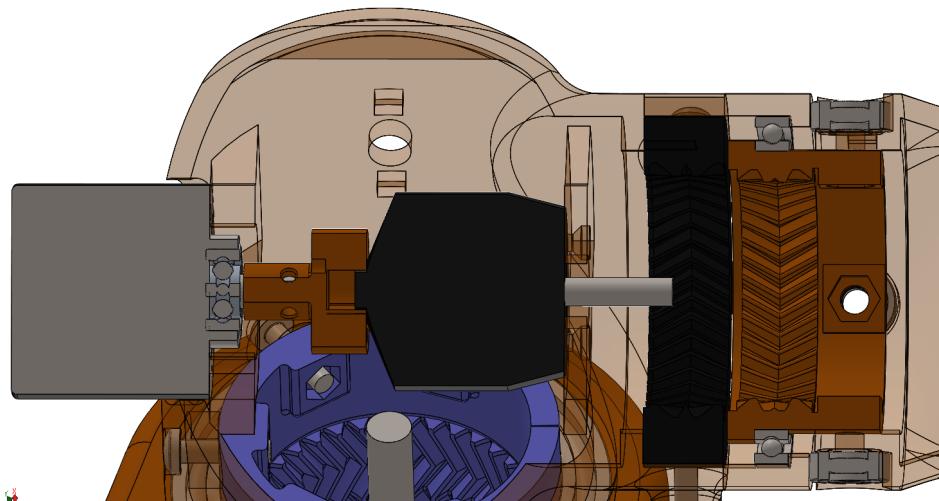


Fig. 38: Een doorsnede van as 4 van de derde iteratie

## 4.7 As 5

De vijfde as was één van de moeilijkste om te ontwerpen. Dit komt door de beperkte plaats dat ik ter beschikking had. Na veel tekeningen heb ik iets ontwikkeld dat tot op vandaag nog steeds goed werkt.

Eerst wou ik hetzelfde principe als de derde as toepassen, maar ook daarvoor was er niet genoeg plaats. Daarom heb ik geopteerd om de motor zo goed als buiten de as te plaatsen en de motoras te verlengen door er een as op te bevestigen dat ook langs de andere kant door een kogellager ondersteund wordt. Die as bestaat uit twee pulleys van verschillende types. Dit zorgt ervoor dat ik een riemoverbrenging van het GT2 type kan gebruiken om de encoder aan te drijven en een riemoverbrenging van het type HTD3mm om de algemene metalen as aan te drijven dat op zijn beurt de hoofdoverbrenging aandrijft. Deze as heeft uiteindelijk een overbrengingsverhouding van 100.8:1.

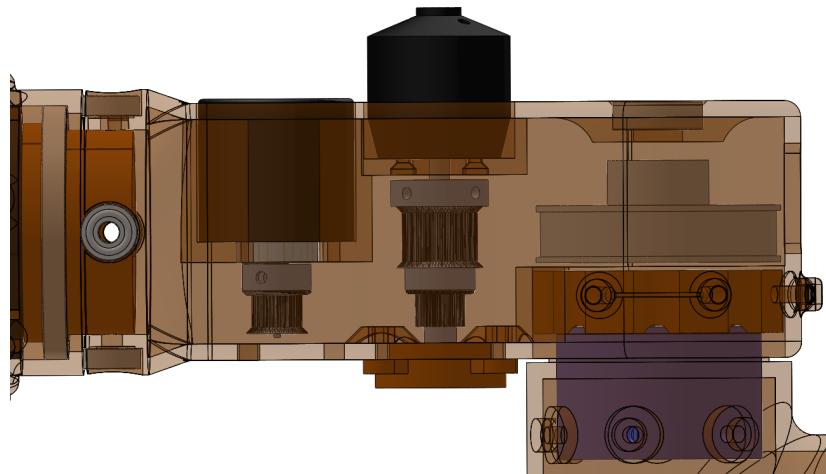


Fig. 39: As 5 van de derde iteratie

## 4.8 As 6

De zesde as is een exacte kopie van de vierde as buiten het feit dat er zich een asverschuiving plaatsvindt. De motor drijft namelijk een geprinte pulley aan, dat via een riemoverbrenging van het type HTD3mm een andere pulley aandrijft, dat op zijn beurt de hoofdoverbrenging aandrijft. De uiteindelijke overbrengingsverhouding is 50.4:1. Ik heb deze asverschuiving voorzien omdat het als gevolg heeft dat de wiskunde eenvoudiger wordt. De handmatige berekeningen zijn eenvoudiger, maar het is ook een vereiste als ik wil werken met bibliotheken zoals IKFast. Zie hoofdstuk 8.10 voor meer informatie over de vereenvoudiging.

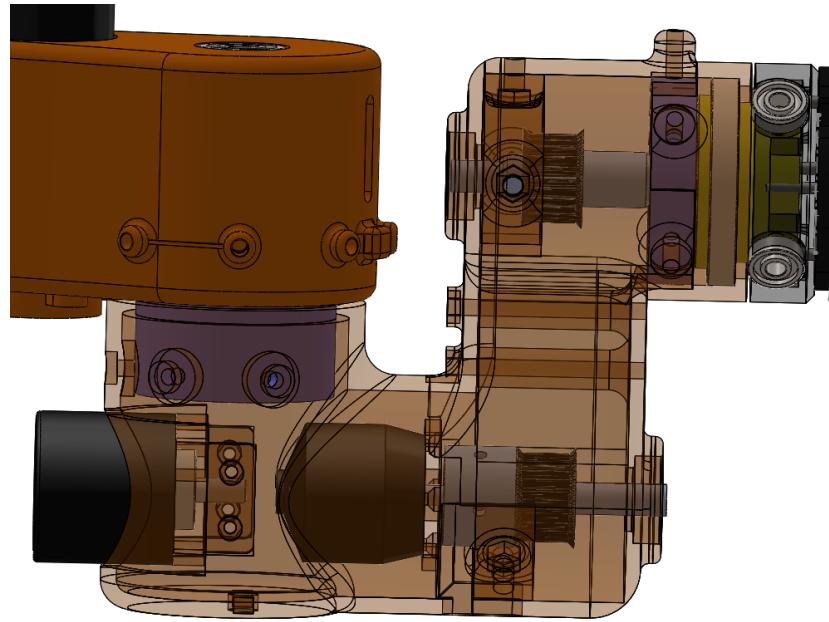


Fig. 40: As 6 van de derde iteratie

#### 4.9 End effector

Voor het plaatsen van een tool heb ik een algemene end-effector interface ontwikkeld. Hier kan in Solidworks verder gebouwd worden om eigen end-effectoren en tools te ontwikkelen.

Het grijze stuk zit direct op de zesde as bevestigd, zoals je kan zien in de onderstaande doorzichtige foto. Het is ook een mooi voorbeeld van hoe het systeem van de kogellagers in elkaar zit. De afmetingen van het vlak waarop gewerkt kan worden voor het ontwikkelen van de tool kan je zien in bijlage H.

Qua bekabeling voorzie ik vier kabels. Twee zijn er voor stroomvoorziening en de andere twee voor data. Op deze manier kan ik zo goed als elke interface aansluiten met communicatiemogelijkheden langs beide richtingen.

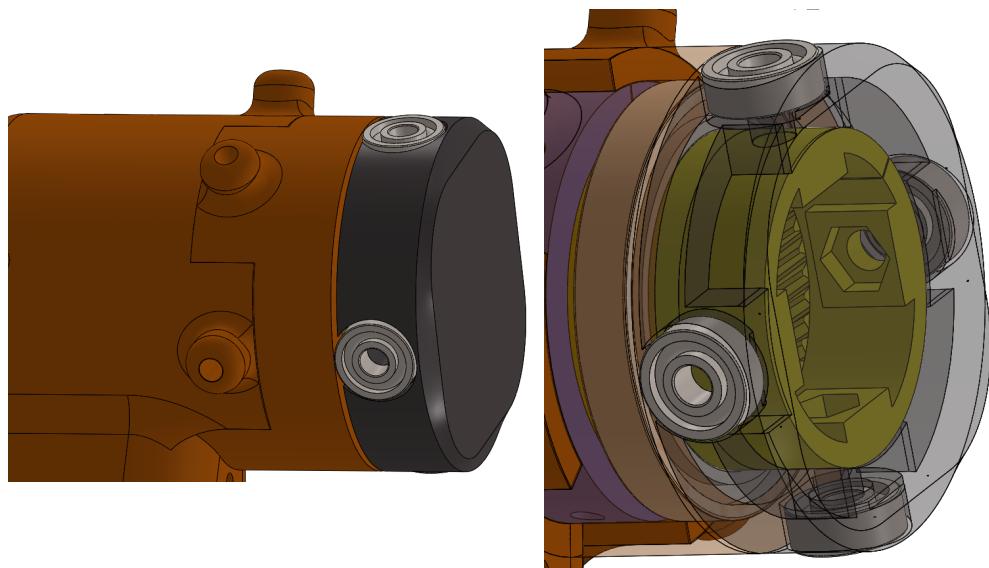


Fig. 41: Het grijze stuk is de interface waarop tools ontwikkelt kunnen worden

## 4.10 De grijper

Hoewel het ontwikkelen van tools eigenlijk niet meer tot deze GIP behoort en ik ook geen tijd meer over had om zelf één te ontwerpen, heb ik geopteerd voor de grijper van een andere robotarm aan te passen, namelijk die van de BCN3D Moveo.

Deze voorziet een eenvoudige grijper, aangestuurd door een servo motor, meerbepaald de HS422 servo. Deze grijper is vrijgegeven onder een open-source licentie. De source code kan je hier vinden: <https://github.com/BCN3D/BCN3D-Moveo> Ik heb wel de tandwielen en de klauwen aangepast om een groter moment te kunnen uitoefenen.

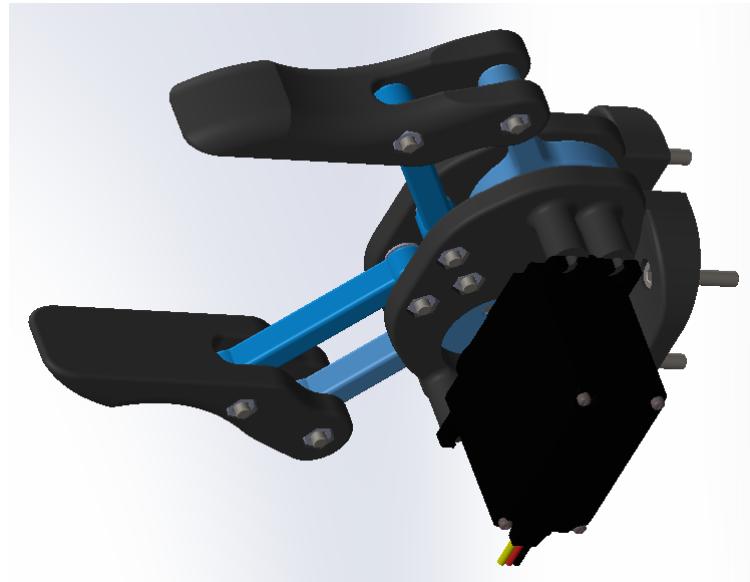


Fig. 42: De BCN3D Moveo grijper

**Denavit-Hartenbergconventie** Deze tool moet uiteraard ook voldoen aan de Denavit-Hartenbergconventie (zie hoofdstuk 7). In onderstaande afbeelding kan je zien waar ik het Denavit-Hartenbergassenstelsel geplaatst heb. De Denavit-Hartenbergparameters zijn  $(\theta_7, d_7, r_7, \alpha_7) = (0, 79, 0, 0)$ .

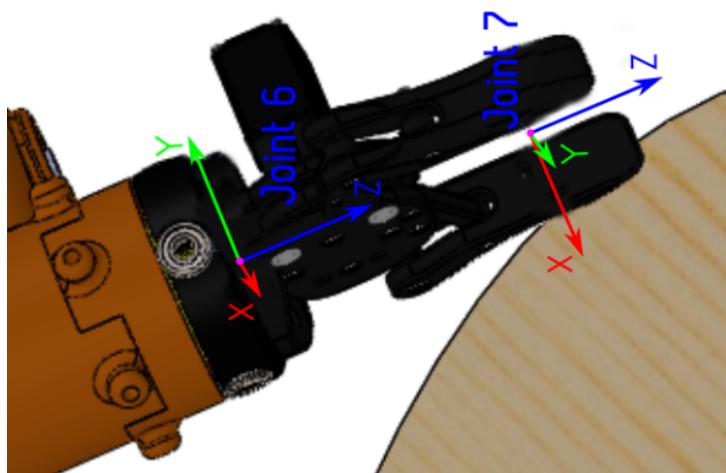


Fig. 43: De DH-assenstelsels van joint 6 en de tool (joint 7)

## 5 Elektronica behuizing

### 5.1 Inleiding

Uiteraard heb ik een behuizing moeten ontwikkelen om alle ODrives en stroomvoorziening in te steken. Deze behuizing is volledig gelaserd uit 4mm populier, opnieuw door de lasersnijder van FabLab Factory. De behuizing is ontworpen met de gedachte om het zo zelfstandig mogelijk te maken. Als de behuizing volledig op zichzelf kan werken, kan ik de bijhorende elektronica bij andere projecten inzetten waar ik gebruik maak van BLDC-motoren.

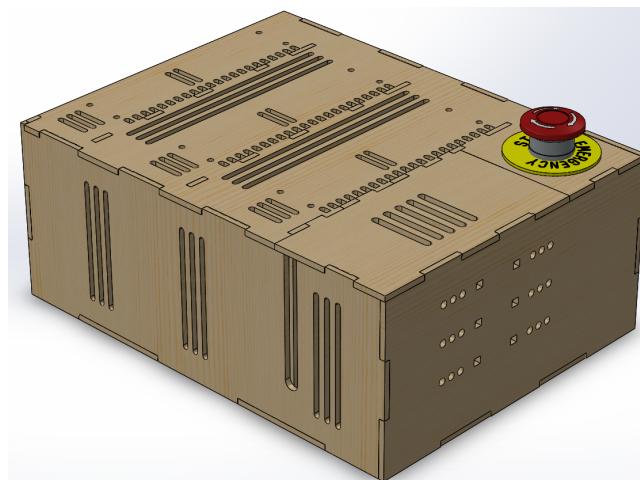


Fig. 44: De elektronica behuizing

### 5.2 Plaatsing elektronica

De ODrives en toebehoren zijn vooraan in de behuizing geplaatst. De drie ODrive's die ik nodig heb voor mijn robotarm (twee motoren per ODrive) zijn op elkaar gestapeld en met behulp van geprinte voetjes op de bodemplaat gezet. Het voorpaneel heeft een aantal gaten waar de connectoren voor de motoren en de encoders geplaatst zijn. Aan het linkerpaneel is er ook een uitsparing voorzien om de USB-kabels naar buiten te laten komen. De ODrives zijn redelijk hoog geplaatst zodanig dat er onderaan plaats vrijkomt om de vermogensweerstanden te plaatsen.

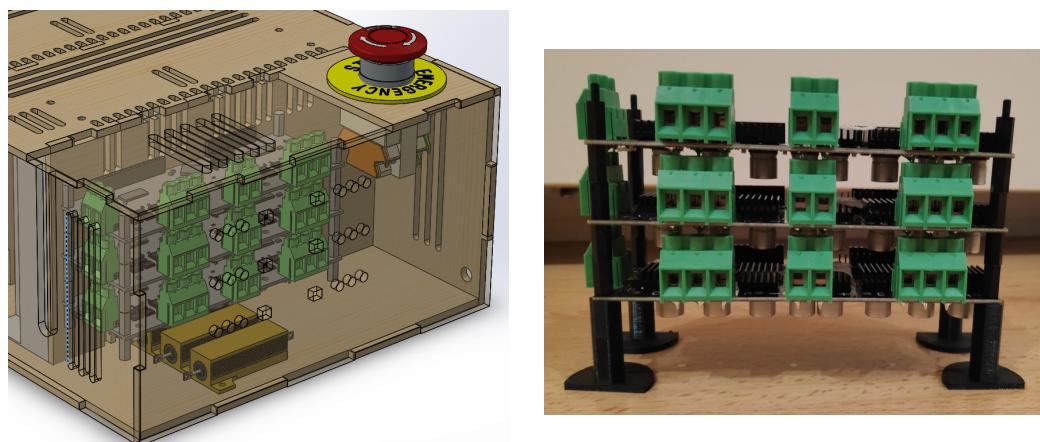


Fig. 45: Links: de elektronica in de behuizing. Rechts: de opeenstapeling van ODrives

### 5.3 Stroomvoorziening

Elke ODrive is voorzien van een aparte power supply van 24V/16A. Op het eerste zicht lijkt dit misschien overkill, maar je moet rekening houden met de piekstromen die kunnen ontstaan. Als alle zes motoren op hetzelfde moment een piekstroom trekken, dan is de kans groot dat de power supply het begeeft in het geval dat er maar één power supply is voor alle ODrives.



Fig. 46: Een overzicht van de power supplies

Omdat ik wil vermijden dat ik de hele behuizing uit elkaar moet halen indien ik een power supply moet vervangen, heb ik er voor gezorgd dat je de power supplies er makkelijk in en uit kan halen. Als je er een power supply in wilt steken, moet je het linkerpaneel verwijderen, de power supply in de sleuf schuiven, de elektrische verbindingen leggen en voordat je het linkerpaneel er terug op mag plaatsen moet je de power supply langs de bovenkant vastmaken met schroeven. Het omgekeerde geldt voor het verwijderen van een power supply.

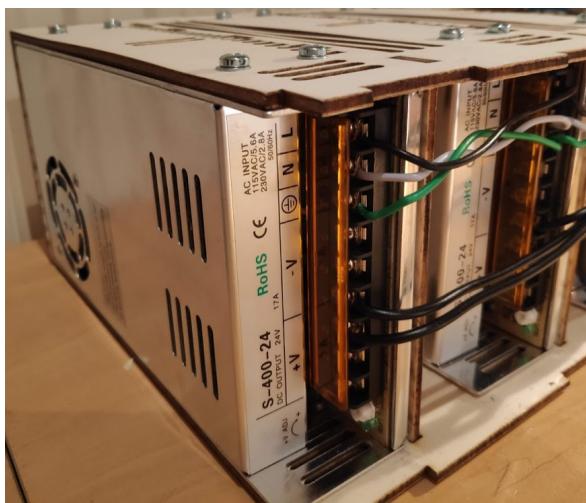


Fig. 47: Een close-up van een geïnstalleerde power supply

Zoals je ook kan zien staan de power supplies redelijk ver uit elkaar. Dit heb ik zo gedaan om er voor te zorgen dat het actief koelsysteem van elke power supply goed zijn werk kan doen. Dat is ook de reden waarom er in elk houten vlak gleuven zitten die overeenkomen met de verluchtingsgleuven van de power supplies.



Fig. 48: De IEC C14 plug van de behuizing

De power supplies krijgen hun ingangsstroom via een IEC C14 connector. De stroom naar de power supplies kan ook onderbroken worden door de noodstop. Dit is echter een 'trage' noodstop aangezien de power supplies na het indruwen van de noodstop nog even in staat zijn om genoeg stroom te geven om te robotarm te besturen. De 'snelle' noodstop zit verwerkt in ODriveControl, de zelfgeschreven software dat verantwoordelijk is voor de communicatie tussen de hardware en de computer. Zie hoofdstuk 12 voor meer informatie.

## 5.4 Voorpaneel

Het voorpaneel bevat per ODrivekanaal een connector om de driefasige aansluiting van de BLDC-motor aan te sluiten en een connector voor de vier aansluitingen (5V, GND, A en B) van de bijhorende encoder.



Fig. 49: Het voorpaneel. De cirkels zijn voor de motoren, de vierkanten zijn voor de encoders

## 6 Toekomstvisie praktische uitwerking

In het algemeen ben ik tevreden over het niveau van deze robotarm. Zeker als je rekening houdt met het feit dat de derde getekende iteratie de eerste iteratie is dat

volledig praktisch uitgewerkt is. De robotarm heeft al op een paar beurzen gestaan en doet het tot nu toe goed qua duurzaamheid.

Wat ik wel wil oplossen in de toekomst, is de speling van de hoofdoverbrenging. Dit heeft een nadelig effect op de accuraatheid van de robotarm. Verder wil ik ook nog een wijziging aanbrengen aan de elektronica behuizing omdat het nu redelijk moeilijk is om onderhoud te doen aan de ODrives.

## 7 Denavit-Hartenbergconventie

### 7.1 Inleiding

De Denavit-Hartenbergparameters zijn vier parameters die gebruikt kunnen worden om transformaties voor te stellen tussen de verschillende gewrichten van een kinematische ketting, of een robotische manipulator. Deze parameters zijn geïntroduceerd door Jabues Denavit en Richard Hartenberg in 1955. Op zijn beurt toonde Richard Paul hoe deze parameters waardevol zijn voor kinematische analyse in 1981. Hoewel er veel conventies ontwikkeld zijn, is deze Denavit-Hartenberg conventie nog steeds de populairste.

Eerst en vooral moet je de robotische manipulator opdelen in 'links' en 'joints'. Een link is een fysiek onderdeel van de robotarm en een joint is de connectie tussen twee opeenvolgende links. Rond een joint kan er zich ook een rotatie en/of een translatie voordoen.

Je hebt dus twee soorten joints: de roterende joint om twee linken te verbinden via een rotatie en de translatende joint die twee linken verbindt via een translatie volgens een rechte.

De conventie houdt in dat je op elke joint een assenstelsel plaatst en vervolgens met vier Denavit-Hartenbergparameters de transformatiematrix bepaalt van een vorige assenstelsel (de vorige joint) naar het huidige assenstelsel (huidige joint). Door dit systematisch te doen kan je op die manier een punt transformeren van een globaal, vrij te kiezen assenstelsel naar het assenstelsel van de end-effector. Je kan zo eenvoudig de positie en de rotatie van de end-effector t.o.v. het globaal assenstelsel bepalen.

### 7.2 Denavit-Hartenbergassenstelsels

Eerst en vooral wordt er een globaal assenstelsel vrij gekozen. Daarna wordt er per joint een assenstelsel ingevoerd. De regels voor het invoeren van een assenstelsel bij de roterende joint  $n$  zijn als volgt:

1. De z-as wordt in de richting van de rotatieas gelegd
2. De x-as wordt parallel met de normaal tussen de vorige- en de huidige z-as gelegd, zodanig dat:

$$x_n = z_n \times z_{n-1}$$

Als er geen unieke normaal is, dan is parameter  $d$  (zie hoofdstuk 7.3) vrij te kiezen. De richting van  $x_n$  is van  $z_{n-1}$  naar  $z_n$ .

3. De y-as wordt gekozen zodanig dat je een rechtshandig assenstelsel krijgt.

Het assenstelsel van de eerste joint ( $n = 1$ ) wordt gekozen ten opzichte van het globaal assenstelsel.

Op onderstaande figuur kan je van elke joint van mijn robotarm het ingevoerde assenstelsel zien.

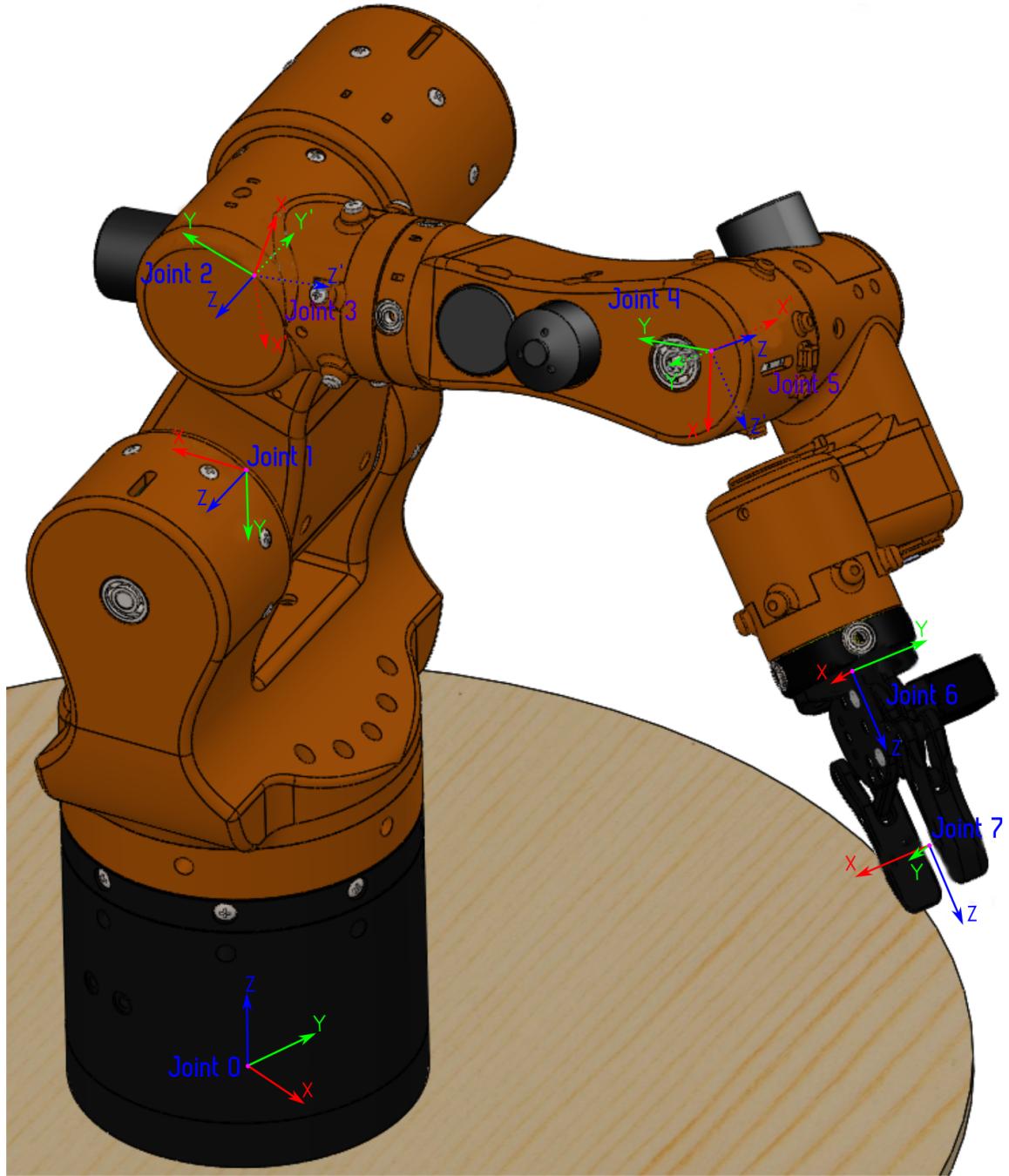


Fig. 50: De assenstelsels van de joints van mijn robotarm, bepaald door de DH-conventie

### 7.3 Denavit-Hartenbergparameters

Om transformaties uit te kunnen voeren tussen assenstelsels, moeten we Denavit-Hartenbergparameters bepalen. De vier Denavit-Hartenbergparameters van joint  $n$  worden bepaald als volgt:

- $d$ : de afstand langs de vorige z-as tot de normaal van  $z_n$  en  $z_{n-1}$ . Indien er geen unieke normaal is, is deze parameter vrij te kiezen. Deze parameter is de onbekende bij lineaire gewrichten.
- $\theta$ : dit is de hoek rond de vorige z-as, van  $x_{n-1}$  naar  $x_n$ . Deze parameter is de onbekende bij roterende gewrichten. Dit is de waarde die de motor aanpast bij een rotatie.
- $r$ : de lengte van de huidige normaal.
- $\alpha$ : de hoek rond de normaal van  $z_n$  en  $z_{n-1}$  van de vorige z-as  $z_{n-1}$  tot de nieuwe z-as  $z_n$ .

**Toegepast** Op basis van de theorie en de CAD-tekening van de robotarm, heb ik alle parameters bepaald. Een overzicht van alle DH-parameters van mijn GIP vindt u hier:

Joint n	$\theta_n$	$d_n$	$r_n$	$\alpha_n$
0	/	/	/	/
1	$\theta_1$	291	0	$-\pi/2$
2	$\theta_2$	94.5	181	0
3	$\theta_3$	0	0	$-\pi/2$
4	$\theta_4$	216.34	0	$-\pi/2$
5	$\theta_5$	0	0	$-\pi/2$
6	$\theta_6$	145.04	0	0
7	$\theta_7$	$d_7$	$r_7$	$\alpha_7$

Tab. 3: De DH-parameters van mijn robotarm

Joint 0 is het vrij gekozen globaal assenstelsel, joint 7 stelt de tool voor en als gevolg zijn  $d_7$ ,  $r_7$  en  $\alpha_7$  afhankelijk van die tool. Deze parameters zijn dus te vinden bij de besprekking van de tool. Aangezien er geen motor voorzien is voor de zevende link (de link die de end-effector interface verbindt met de tool) is ook  $\theta_7$  constant en vormt het dus geen probleem voor de inverse kinematica. (Zie hoofdstuk 8)

## 7.4 Denavit-Hartenbergtransformatiematrix

De transformatiematrix wordt gebruikt om van het ene assenstelsel naar het andere assenstelsel te transformeren. Het legt met andere woorden vast hoe men op basis van de DH-parameters kan transformeren van de ene joint naar de andere.

De transformatiematrix zelf bestaat uit het product van twee schroeftransformaties. Omdat een schroeftransformatie meestal gesplitst wordt in een translatie en een rotatie, kan een transformatiematrix dus voorgesteld worden als een product van vier matrices. Elke DH-parameter is verantwoordelijk voor één matrix, namelijk:

$$Trans_{z_{n-1}}(d_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{z_{n-1}}(\theta_n) = \begin{bmatrix} \cos(\theta_n) & -\sin(\theta_n) & 0 & 0 \\ \sin(\theta_n) & \cos(\theta_n) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trans_{x_{n-1}}(r_n) = \begin{bmatrix} 1 & 0 & 0 & r_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{x_{n-1}}(\theta_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_n) & -\sin(\alpha_n) & 0 \\ 0 & \sin(\alpha_n) & \cos(\alpha_n) & 0 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

De transformatiematrix van joint  $n - 1$  naar  $n$  wordt bepaald door:

$${}^{n-1}T_n = Trans_{z_{n-1}}(d_n) Rot_{z_{n-1}}(\theta_n) Trans_{x_{n-1}}(r_n) Rot_{x_{n-1}}(\alpha_n)$$

Na uitwerking volgt dat:

$${}^{n-1}T_n = \begin{bmatrix} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformatiematrices hebben de eigenschap dat we ze met elkaar kunnen vermenigvuldigen om zo opeenvolgende transformaties door te voeren. We kunnen een transformatie van een punt van joint  $n$  naar joint  $m$  dus als volgt voorstellen:

$${}^nT_m = {}^nT_{n+1} * {}^{n+1}T_{n+2} * \dots * {}^{m-2}T_{m-1} * {}^{m-1}T_m$$

Het maakt niet uit dat joint  $m$  voor of na joint  $n$  ligt in de kinematische ketting, maar de volgorde van vermenigvuldiging speelt wel een zeer belangrijke rol en je moet er ook voor zorgen dat je op één of andere manier toch rekening houdt met elke tussenliggende joint.

## 7.5 De inverse van de DH-transformatiematrix

Voor het uitwerken van de kinematica (zie hoofdstuk 8) moeten we niet alleen de DH-transformatiematrix gebruiken, maar ook de inverse. De inverse DH-transformatiematrix wil geometrisch gezien hetzelfde zeggen van de normale DH-transformatiematrix. Het enige verschil is dat je van een joint naar een vorige joint transformeert in plaats van naar een volgende joint. In bijlage A kan u een bewijs vinden dat de inverse van de DH-matrix,  ${}^{n-1}T_n^{-1} = {}^nT_{n-1}$ , de volgende definitie heeft:

$${}^{n-1}T_n^{-1} = \begin{bmatrix} \cos \theta_n & \sin \theta_n & 0 & -r_n \\ -\sin \theta_n \cos \alpha_n & \cos \theta_n \cos \alpha_n & \sin \alpha_n & -d \sin \alpha_n \\ \sin \theta_n \sin \alpha_n & -\cos \theta_n \sin \alpha_n & \cos \alpha_n & -d \cos \alpha_n \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

**Toegepast** Op basis van voorgaande theorie heb ik alle DH-transformatiematrices en de inverses kunnen opstellen. Een overzicht:

$${}^0T_1 = \left[ \begin{array}{ccc|c} \cos \theta_1 & 0 & -\sin \theta_1 & 0 \\ \sin \theta_1 & 0 & \cos \theta_1 & 0 \\ 0 & -1 & 0 & d_1 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad {}^0T_1^{-1} = \left[ \begin{array}{ccc|c} \cos \theta_1 & \sin \theta_1 & 0 & 0 \\ 0 & 0 & -1 & d_1 \\ -\sin \theta_1 & \cos \theta_1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$${}^1T_2 = \left[ \begin{array}{ccc|c} \cos \theta_2 & -\sin \theta_2 & 0 & r_2 \cos \theta_2 \\ \sin \theta_2 & \cos \theta_2 & 0 & r_2 \sin \theta_2 \\ 0 & 0 & 1 & d_2 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad {}^1T_2^{-1} = \left[ \begin{array}{ccc|c} \cos \theta_2 & \sin \theta_2 & 0 & -r_2 \\ -\sin \theta_2 & \cos \theta_2 & 0 & 0 \\ 0 & 0 & 1 & -d_2 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$${}^2T_3 = \left[ \begin{array}{ccc|c} \cos \theta_3 & 0 & -\sin \theta_3 & 0 \\ \sin \theta_3 & 0 & \cos \theta_3 & 0 \\ 0 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad {}^2T_3^{-1} = \left[ \begin{array}{ccc|c} \cos \theta_3 & \sin \theta_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -\sin \theta_3 & \cos \theta_3 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$${}^3T_4 = \left[ \begin{array}{ccc|c} \cos \theta_4 & 0 & -\sin \theta_4 & 0 \\ \sin \theta_4 & 0 & \cos \theta_4 & 0 \\ 0 & -1 & 0 & d_4 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad {}^3T_4^{-1} = \left[ \begin{array}{ccc|c} \cos \theta_4 & \sin \theta_4 & 0 & 0 \\ 0 & 0 & -1 & d_4 \\ -\sin \theta_4 & \cos \theta_4 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$${}^4T_5 = \left[ \begin{array}{ccc|c} \cos \theta_5 & 0 & -\sin \theta_5 & 0 \\ \sin \theta_5 & 0 & \cos \theta_5 & 0 \\ 0 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad {}^4T_5^{-1} = \left[ \begin{array}{ccc|c} \cos \theta_5 & \sin \theta_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -\sin \theta_5 & \cos \theta_5 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$${}^5T_6 = \left[ \begin{array}{ccc|c} \cos \theta_6 & -\sin \theta_6 & 0 & 0 \\ \sin \theta_6 & \cos \theta_6 & 0 & 0 \\ 0 & -1 & 0 & d_6 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad {}^5T_6^{-1} = \left[ \begin{array}{ccc|c} \cos \theta_6 & \sin \theta_6 & 0 & 0 \\ -\sin \theta_6 & \cos \theta_6 & 0 & 0 \\ 0 & 0 & 1 & -d_6 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

Naar: Wikipedia, Denavit–Hartenberg parameters

# 8 Kinematica

## 8.1 Inleiding

Kinematica of bewegingsleer is het onderdeel van de klassieke mechanica dat zich bezig houdt met beweging. De chemische en fysische effecten op het lichaam worden achterwege gelaten, alsook de krachten die er op inwerken. Het verband tussen krachten en beweging wordt namelijk bestudeerd in de dynamica.

Een robotarm kan je kinematisch beschrijven als een stelsel van opeenvolgende linken, die met elkaar verbonden zijn via joints of actuatoren. Voor een robotarm kan je de kinematica splitsen in twee onderdelen, namelijk voorwaartse- en inverse kinematica.

Voorwaartse kinematica is het probleem waarbij het stelsel joints volledig beschreven is (waar dus elke variabele gekend is) en waar je de eindcoördinaat en de richting van end-effector wil weten.

Inverse kinematica is, zoals de naam al aangeeft, het omgekeerde van voorwaartse kinematica. Bij dit probleem ken je de coördinaat en richting van de end-effector en probeer je te zoeken naar een beschrijving van het stelsel joints die aan de voorwaarden van de end-effector voldoet.

De voorwaartse kinematica is meestal vrij straigth-forward. De inverse kinematica is een ander paar mouwen. Hier zit je met het probleem dat er vrijwel altijd veel mogelijke oplossingen zijn als het punt in de eerste plaats bereikbaar is. Het is -in tegenstelling tot voorwaartse kinematica- niet gegarandeerd dat een oplossing effectief bestaat, ookal ligt het punt in het bereik van de robotarm. Daarom zijn er verschillende methodes ontwikkelt om dit probleem zo goed mogelijk aan te pakken. Ik heb voor mijn robotarm drie verschillende methodes uitgewerkt. Meerbepaald twee iteratieve methodes (Stochastic Gradient Descent en de Jacobimatrix) en een analytische methode, eerst handmatig en daarna met behulp van de IKFast bibliotheek. Per methode zal ik de werkwijze bespreken, alsook de voor- en de nadelen.

Eerst en vooral moeten we afspraken maken rond een aantal zaken. Bijvoorbeeld hoe we de positie en de oriëntatie van de end-effector definiëren.

## 8.2 Positie van de end-effector

De positie van de end-effector is eenvoudig te bepalen met behulp van een positieverctor. We stellen daarom vast dat de positieverctor  $\vec{P}$  van de end-effector als volgt bepaald is:

$$\vec{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

Waarbij het aangrijppingspunt zich altijd in de oorsprong van het globaal assenstelsel bevindt.

## 8.3 Oriëntatie van de end-effector: eulerhoeken

Eulerhoeken zijn een set van hoeken die gebruikt kunnen worden om een 3D-rotatie te bepalen. Een rotatie met eulerhoeken wordt als volgt bepaald: gegeven zijn twee

rechtshandige assenstelsels, waarvan één vast staat en de andere mobiel is. Initieel zijn het vast assenstelsel ( $OXYZ$ ) en het mobiel assenstelsel ( $OX'Y'Z'$ ) hetzelfde. Deze assenstelsels delen ook dezelfde oorsprong. Om de rotatie van een derde assenstelsel ( $OX''Y''Z''$ ) vast te stellen, ten opzichte van het vast assenstelsel, moeten we het mobiele assenstelsel volgens de onderstaande regels naar het derde assenstelsel brengen door drie rotaties:

1. De eerste rotatie: rond  $X$ ,  $Y$  of  $Z$  van het vast assenstelsel of  $X'$ ,  $Y'$  of  $Z'$  van het mobiele assenstelsel, met als hoek de eerste eulerhoek  $\alpha$ .
2. De tweede rotatie: rond de  $X$ ,  $Y$  of  $Z$  van het vast assenstelsel of  $X'$ ,  $Y'$  of  $Z'$  van het mobiele assenstelsel, met als hoek de tweede eulerhoek  $\beta$ .
3. De derde rotatie: rond de  $X$ ,  $Y$  of  $Z$  van het vast assenstelsel of  $X'$ ,  $Y'$  of  $Z'$  van het mobiele assenstelsel, met als hoek de derde eulerhoek  $\gamma$ .

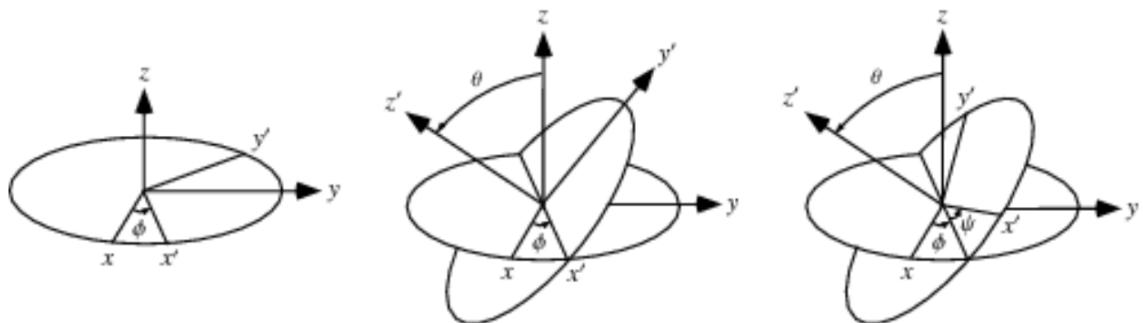


Fig. 51: Een voorstelling van eulerhoeken met als conventie  $Z'X'Z'$

Bron: Weisstein, Eric W., *Euler Angles*

De volgorde van de rotaties ( $XYZ$ ,  $X'Z'Y$ ,  $X'Y'X$ , ...) speelt een grote rol. Omdat we bij elke rotatie kunnen kiezen uit zes assen ( $X$ ,  $Y$ ,  $Z$ ,  $X'$ ,  $Y'$ ,  $Z'$ ), hebben we  $6^3 = 216$  mogelijkheden. Als je een optie gebruikt waarvan twee opeenvolgende rotaties rond dezelfde as zijn ( $YYZ$ ,  $ZZX'X'$ , ...), dan kan je niet elke rotatie beschrijven. Bovendien kan het gebeuren dat de assen  $X$ ,  $Y$  en  $Z$  respectievelijk op de assen  $X'$ ,  $Y'$  en  $Z'$  liggen, waardoor opties zoals  $XYZ$  en  $X'YZ$  hetzelfde betekenen. Dit zorgt ervoor dat er uiteindelijk twaalf nuttige conventies overblijven:  $XYZ$ ,  $XZY$ ,  $YXZ$ ,  $YZX$ ,  $ZXY$ ,  $ZYX$ ,  $XYX$ ,  $XZX$ ,  $YXY$ ,  $YZY$ ,  $ZXZ$ ,  $ZYZ$ .

Merk op dat deze twaalf conventies enkel gebruik maken van de assen van het vast assenstelsel. Daarom worden deze twaalf conventies ook wel de vaste conventies genoemd. Elk vaste conventie heeft echter een 'mobiel' equivalent dat gebruik maakt van de assen van het mobiele assenstelsel, namelijk  $X'$ ,  $Y'$ , en  $Z'$ .

Dit legt onmiddellijk het eerste probleem van eulerhoeken bloot. Er is geen vaste conventie om eulerhoeken te bepalen, wat het soms zeer ingewikkeld maakt voor gebruikers en voor ontwikkelaars als ze bestaande code in andere systemen willen integreren.

De mobiele  $X'Y'Z'$ -conventie is een bepaalde conventie dat gebruikt wordt in de luchtvaart. De eulerhoeken zijn dan gekend als yaw, pitch en roll.

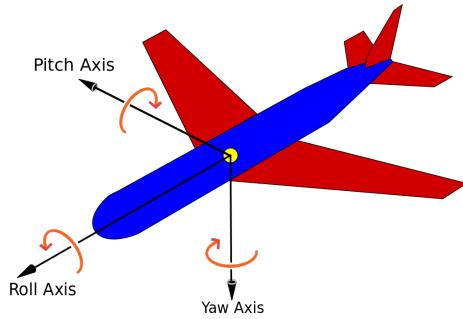


Fig. 52: De yaw, pitch en roll parameters die gebruikt worden in de luchtvaart

Bron: Wikipedia, *Euler angles*

Een ander probleem is de zogenaamde 'gimbal lock'. Dat is het probleem waarbij twee eulerhoeken dezelfde rotatie uitvoeren en het volledige systeem zo een vrijheidsgraad verliest. Dat zorgt ervoor dat je in dat geval niet meer elke rotatie kan beschrijven, wat voor onbepaald en mogelijks gevaarlijk gedrag kan zorgen bij het besturen van een robotarm.

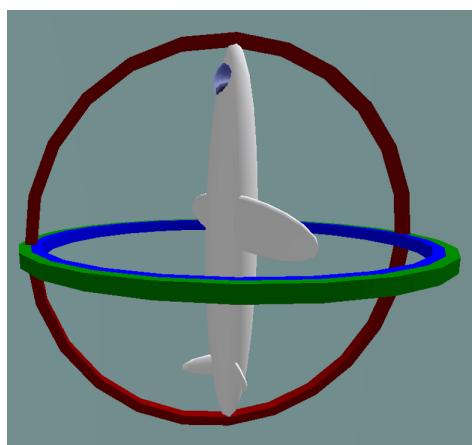


Fig. 53: De eerste eulerhoek heeft ervoor gezorgd dat de twee volgende eulerhoeken hetzelfde rotatieeffect hebben.

Bron: Wikipedia, *Euler angles*

*Naar:*

[Wikipedia, Euler angles](#)

[Mecademic, How is orientation in space represented with Euler angles?](#)

[Wikipedia, Gimbal lock](#)

## 8.4 Oriëntatie van de end-effector: as-hoek methode

Een assenstelsel kan ook naar elke rotatie gebracht worden met behulp van de as-hoek methode. Gegeven is een vast referentieassenstelsel  $OX_A Y_A Z_A$ . Een rotatie van dit assenstelsel naar een nieuw assenstelsel  $OX_B Y_B Z_B$  kan vervolgens in elk geval bepaald worden door een hoek  $\theta$  te zoeken en een eenheidsvector  $r$  waarrond geroteerd wordt.

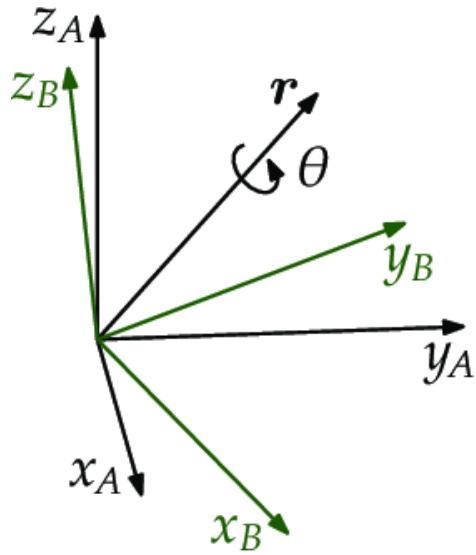


Fig. 54: Een rotatie van assenstelsel  $OX_A Y_A Z_A$  naar assenstelsel  $OX_B Y_B Z_B$

Bron: Brandstötter, “Adaptable Serial Manipulators in Modular Design”

Dit is één van de eenvoudigste manieren om een rotatie voor te stellen, maar een groot nadeel is dat er singulariteiten zijn als  $\theta = 180^\circ$  of  $\theta = 0^\circ$ . Het zou als gevolg kunnen gebeuren dat de end-effector zich volledig zal omdraaien als je met een kleine thetaverandering over deze singulariteiten wilt bewegen. Dit kan leiden tot onvoorspelbaar en gevaarlijk gedrag.

## 8.5 Quaternionen

De quaternionen zijn een uitbreiding van de complexe getallen. De complexe getallen zijn op hun beurt een uitbreiding van de reële getallen. Quaternionen zijn voor het eerst vermeld door de Ierse wiskundige William Rowan Hamilton in 1843. De quaternionen worden onder andere gebruikt voor 3D-transformaties.

Omdat de quaternionen zeer sterk lijken op complexe getallen, is het handig om eerst de complexe getallen even te bespreken en wat hun voordeel is bij het uitvoeren van 2D-rotaties.

### 8.5.1 Complexe getallen

Complexe getallen zijn tweedimensionale getallen. Om complexe getallen te noteren, moeten we een nieuwe eenheid invoeren, namelijk de imaginaire eenheid  $i$  met  $i^2 = -1$ .

Algemeen kunnen we dan stellen dat een complex getal  $c$  geschreven kan worden als volgt:

$$c = a + bi$$

Ook kunnen we een complex getal voorstellen in een assenstelsel waarbij de x-as de reële as genoemd wordt en de y-as met eenheid  $i$  de imaginaire as genoemd wordt. Neem nu het punt P met de coordinaat  $(3, 2)$ . Dit kunnen we dan eenvoudiger voorstellen als  $3 + 2i$ . We gaan namelijk drie eenheden langs de reële as en twee eenheden langs de imaginaire as.

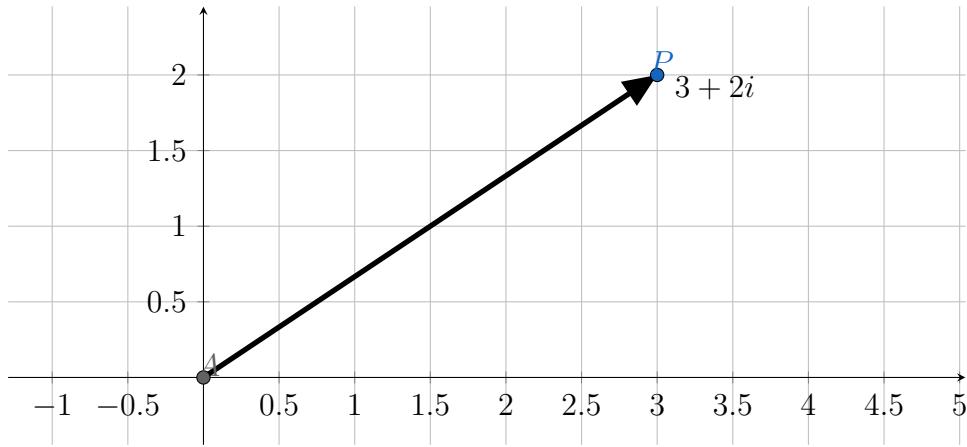


Fig. 55: Een punt in een imaginair assenstelsel

**Het vermenigvuldigen van complexe getallen** Omdat zowel het reële gedeelte, het imaginair gedeelte en  $i$  als afzonderlijke getallen beschouwt worden, kunnen we de distributiviteitsregels en de regel  $i^2 = -1$  toepassen.

De vermenigvuldiging van twee complexe getallen  $c_1 = a_1 + b_1i$  en  $c_2 = a_2 + b_2i$  gaat dus als volgt

$$\begin{aligned} c_1 * c_2 &= (a_1 + b_1i) * (a_2 + b_2i) = a_1 * a_2 + a_1 * b_2i + b_1i * a_2 + b_1i * b_2i \\ &\Rightarrow (a_1 * a_2 - b_1 * b_2) + (a_1 * b_2 + b_1 * a_2)i \end{aligned}$$

**2D-rotaties met complexe getallen** Een interessant gegeven van complexe getallen is dat als je twee 2D-vectoren met hetzelfde aangrijppingspunt in complexe vorm met elkaar vermenigvuldigt, dat de resulterende vector een rotatie van de ene vector met de andere is.

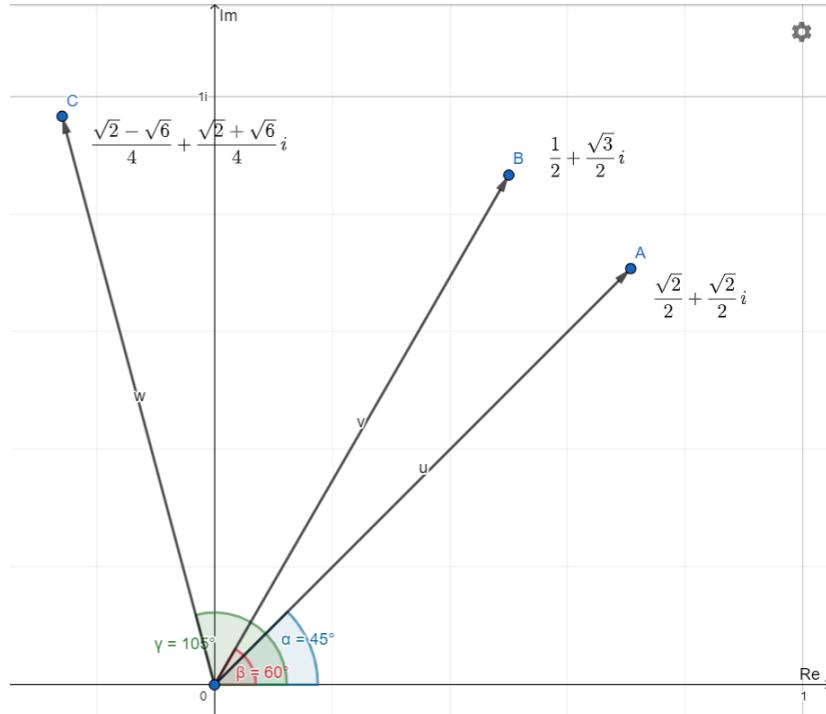


Fig. 56: Een 2D-rotatie met complexe getallen

Als je in bovenstaande figuur de vector  $u$  vermenigvuldigt met de vector  $v$  (uiteraard in complexe vorm), dan zal je als resultaat de vector  $w$  krijgen. En zoals je ziet is  $\gamma = \alpha + \beta$ . Dit wil eigenlijk zeggen dat vector  $w$  het resultaat is van een rotatie van vector  $v$  door vector  $w$ , of vice versa.

Hetzelfde concept wordt toegepast bij quaternionen, maar dan bij 3D-rotaties van 3D-vectoren.

### 8.5.2 Uitbreiding naar de quaternionen

Quaternionen zijn een uitbreiding van de complexe getallen. Om een quaternion te bepalen moeten we bovenop  $i$  nog twee imaginaire eenheden invoeren, namelijk  $j$  en  $k$ . Dit zorgt ervoor dat quaternionen vierdimensionale getallen zijn. Dat heeft als gevolg dat we quaternionen ons niet kunnen voorstellen in hun zuiverste vorm omdat de mens maximaal in drie dimensies kan observeren. De algemene definitie van een quaternion  $q$  is

$$q = a + bi + cj + dk$$

Met  $i^2 = j^2 = k^2 = i * j * k = -1$ .

Daaruit volgt onmiddelijk dat

$$\begin{aligned} i * j &= -j * i = k \\ j * k &= -k * j = i \\ k * i &= -i * k = j \end{aligned}$$

Zoals je ziet speelt de volgorde van vermenigvuldigen ook een rol voor het teken.

**Het vermenigvuldigen van quaternionen (Hamiltonproduct)** Quaternionen vermenigvuldigen kan net zoals bij complexe getallen gebeuren door gebruik te maken van de distributieve eigenschap en de speciale regels voor de imaginaire eenheden. Maar je moet wel goed opletten dat je de volgordes waarin de imaginaire eenheden vermenigvuldigd worden respecteert en de imaginaire eenheden niet zomaar van plaats wisselt, anders zal je tekenfouten introduceren. De vermenigvuldiging van twee quaternionen  $q_1 = a_1 + b_1i + c_1j + d_1k$  en  $q_2 = a_2 + b_2i + c_2j + d_2k$  volgens het Hamiltonproduct gaat als volgt:

$$\begin{aligned} q_1 q_2 &= q_1 * q_2 = (a_1 + b_1i + c_1j + d_1k) * (a_2 + b_2i + c_2j + d_2k) \\ &= a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\ &\quad + (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)i \\ &\quad + (a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)j \\ &\quad + (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k \end{aligned}$$

**De geconjugeerde van een quaternion** De geconjugeerde  $\bar{q}$  van een quaternion  $q = a + bi + cj + dk$  wordt bepaald als volgt:

$$\bar{q} = a - bi - cj - dk$$

**3D-rotaties met quaternionen** Het interessante gegeven van quaternionen is dat het gebruikt kan worden om 3D-rotaties te bepalen. De regel om een puntvector te roteren met behulp van quaternionen gaat als volgt:

$$p' = qp\bar{q}$$

Waarbij  $q$  een quaternion voorstelt,  $\bar{q}$  de geconjugeerde daarvan en  $p$  de puntvector dat geroteerd zal worden. Wel moet de puntvector geschreven worden met behulp van de imaginaire eenheden. Dus:  $p = (p_x, p_y, p_z) = p_x i + p_y j + p_z k$

Je kan de quaternion in zijn zuivere vorm bepalen, maar dat is zeer moelijk aangezien we ons quaternionen niet goed kunnen voorstellen. Je kan ook eerst de gewenste rotatie bepalen met de as-hoek methode en dan omvormen naar een quaternion met de volgende formule:

$$q = \cos \frac{\theta}{2} + (r_x i + r_y j + r_z k) \sin \frac{\theta}{2}$$

Waarbij  $r = (r_x, r_y, r_z)$  de eenheidsvector is waarrond geroteerd wordt en  $\theta$  de hoek rond de eenheidsvector is.

Deze manier om rotaties te bepalen is veruit de beste manier. Je hebt hier geen singulariteiten, er is maar 1 conventie en quaternionen zijn ook ten opzichte van de andere methodes veel vlugger te berekenen door een computer met gespecialiseerde bibliotheken.

Bewijsvoering voor deze formules valt ver buiten het gebied van deze GIP, dus ga ik er van uit dat ze bewezen zijn. Wie toch meer wilt weten over dit concept kan terecht bij de cursus van Bruno Vilhena Adorno (Adorno, *Robot Kinematic Modeling and Control Based on Dual Quaternion Algebra — Part I: Fundamentals.*) dat gratis te downloaden is via de volgende link: <https://hal.archives-ouvertes.fr/hal-01478225/document>

Naar:

Wikipedia, *Quaternion*

Wikipedia, *Quaternions and spatial rotation*

## 8.6 Voorwaartse kinematica

Voorwaartse kinematica is het zoeken van de positie en rotatie van de end-effector bij een bepaalde joint space vector. Met 'joint space vector' bedoel ik de zesdimensionale vector  $\vec{G} = [\theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6]^T$ .

De positie van de end-effector stel ik voor als een puntvector in de ruimte. Zoals je kan zien op onderstaande afbeelding, komt de positie van de end-effector (van de tool) overeen met de oorsprong van het allerlaatste Denavit-Hartenbergassenstelsel.

De rotatie van de end-effector komt overeen met de z-as van het laatste Denavit-Hartenbergassenstelsel. Als rotatiemethode maak ik gebruik van een quaternion dat de z-as van het globaal assenstelsel naar de laatste z-as roteert. Ik maak gebruik van quaternionen omdat dit de meest efficiënte methode is om 3D-rotaties voor te

stellen en omdat de motion planner (MoveIt!, zie hoofdstuk 11.3) ook gebruik maakt van quaternioen.

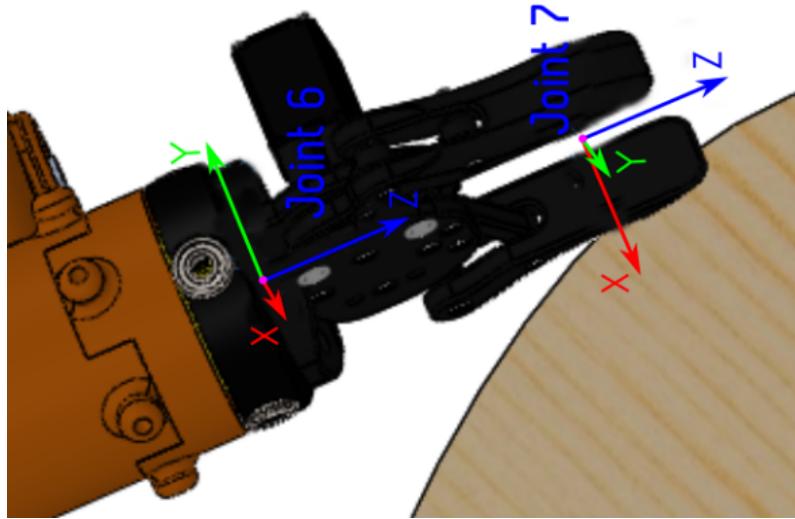


Fig. 57: Het laatste DH-assenstelsel kan gebruikt worden voor de positie en rotatie van de end-effector

Als we  $\vec{P}$ ,  $\vec{Q}$  en  $\vec{G}$  respectievelijk definieren als de positievector, quaternion en de joint state vector, dan kunnen we de kinematische vergelijking  $F(\vec{G})$  schrijven als:

$$F(\vec{G}) = \vec{E} = \begin{bmatrix} \vec{P} \\ \vec{R} \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ q_x \\ q_y \\ q_z \\ q_w \end{bmatrix}$$

Om de positievector en de quaternion te bepalen maken we gebruik van de Denavit-Hartenbergconventie. Uit vergelijking (7) van hoofdstuk 7.3 kunnen we halen dat:

$${}^0T_7 = {}^0T_1 * {}^1T_2 * {}^2T_3 * {}^3T_4 * {}^4T_5 * {}^5T_6 * {}^6T_7$$

Waar  ${}^0T_7$  de transformatiematrix is van het globaal assenstelsel naar het assenstelsel van de end-effector en  ${}^6T_7$  een toolgebonden matrix is zonder onbekende parameter. Deze heeft dus geen grote invloed op de berekeningen.

Aangezien elke transformatiematrix bepaald wordt door een parameter  $\theta_n$ , kunnen we  ${}^0T_7$  bepalen als we de vector  $\vec{G}$  kennen:

$$\vec{G} = [\theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6]^T$$

Aangezien  ${}^0T_7$  nog steeds een transformatiematrix kunnen we stellen dat:

$${}^0T_7 = \left[ \begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{array} \right]$$

Uit deze matrix kan je vervolgens de positieverctor  $\vec{P} = [p_x \ p_y \ p_z]^T$  onmiddelijk aflezen. Ook kunnen we gemakkelijk de rotatiematrix  $\vec{R}$  aflezen. Deze kunnen we dan omvormen naar de quaternion  $\vec{Q}$ .

## 8.7 Inverse kinematica

Inverse kinematica is zoals de naam al aangeeft, de inverse van de voorwaartse kinematica. Hierbij is een eindtoestand van end-effector (de positieverctor  $\vec{P}$  en de quaternion  $\vec{Q}$ ) gekend en zoeken we de bijhorende joint state vector  $\vec{G}$ .

$$\hat{F}(\vec{E}) = \vec{G}$$

Het grote probleem van inverse kinematica van een robotarm is dat een oplossing niet altijd bestaat. En als een oplossing wel bestaat, zijn er meestal meerdere oplossingen. Een goede methode voor de 'beste' oplossing uit te kiezen bestaat niet echt en het kiezen van een slechte oplossing kan gevaarlijk zijn. Als je bijvoorbeeld bijna op de maximale limiet zit van een joint en je kiest een oplossing dat daar over gaat, kan het zijn dat die joint een volledige rotatie moet maken in de andere richting om dan bijna tegen de minimale limiet te zitten. Deze plotse bewegingen kunnen onverwacht komen en gevaarlijk zijn voor de omgeving.

Er zijn verschillende methodes om de inverse kinematica te berekenen. Ik heb een iteratieve methode ontwikkeld en geprobeerd om een analytische methode te ontwikkelen. De handmatige analytische methode werkte deels in de Unreal Engine simulatie, maar er zitten een aantal fouten in. De analytische methode heb ik daarom berekend met IKFast, een bibliotheek dat hier speciaal voor gemaakt is.

## 8.8 Jacobimatrixmethode

### 8.8.1 De Jacobimatrix

Een jacobimatrix is een matrix die alle partiële afgeleiden bevat van een vectorfunctie.

Neem de twee vectoren  $\vec{A} = [a_1 \ a_2 \ \dots \ a_n]$  en  $\vec{B} = [b_1 \ b_2 \ \dots \ b_m]$ . Dan kunnen we de volgende vectorfunctie  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  bepalen:

$$f(\vec{A}) = \vec{B}$$

Volgens de definitie kunnen we de jacobimatrix bepalen:

$$J = \begin{bmatrix} \frac{\partial b_1}{\partial a_1} & \frac{\partial b_1}{\partial a_2} & \dots & \frac{\partial b_1}{\partial a_n} \\ \frac{\partial b_2}{\partial a_1} & \frac{\partial b_2}{\partial a_2} & \dots & \frac{\partial b_2}{\partial a_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial b_m}{\partial a_1} & \frac{\partial b_m}{\partial a_2} & \dots & \frac{\partial b_m}{\partial a_n} \end{bmatrix}$$

De jacobimatrix kan ook gezien worden als de afgeleide van de functie  $f$ . Met behulp van de definitie van een differentiaal kunnen we dan stellen dat

$$\begin{aligned} d\vec{A} &= J \times d\vec{B} \\ \Leftrightarrow d\vec{B} &= J^{-1} \times d\vec{A} \end{aligned}$$

De laatste bewerking zal belangrijk zijn voor het toepassen van deze methode op het probleem van de inverse kinematica. Er is wel een voorwaarde aan verbonden. Namelijk dat de Jacobimatrix inverteerbaar moet zijn. Indien dit niet zo is, moeten er alternatieve methodes gebruikt worden om deze te benaderen. Dit heeft natuurlijk een nadelig effect op de accuraatheid van deze methode.

*Naar: Wikipedia, Jacobi-matrix*

### 8.8.2 Toegepast op inverse kinematica

We kunnen de jacobimatrix bepalen van de voorwaarde kinematicavergelijking  $F(\vec{G})$ . De jacobimatrix van mijn robotarm zou er als volgt uitzien:

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \dots & \frac{\partial p_x}{\partial \theta_6} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \dots & \frac{\partial p_y}{\partial \theta_6} \\ \frac{\partial p_z}{\partial \theta_1} & \frac{\partial p_z}{\partial \theta_2} & \dots & \frac{\partial p_z}{\partial \theta_6} \\ \frac{\partial q_x}{\partial \theta_1} & \frac{\partial q_x}{\partial \theta_2} & \dots & \frac{\partial q_x}{\partial \theta_6} \\ \frac{\partial q_y}{\partial \theta_1} & \frac{\partial q_y}{\partial \theta_2} & \dots & \frac{\partial q_y}{\partial \theta_6} \\ \frac{\partial q_z}{\partial \theta_1} & \frac{\partial q_z}{\partial \theta_2} & \dots & \frac{\partial q_z}{\partial \theta_6} \\ \frac{\partial q_w}{\partial \theta_1} & \frac{\partial q_w}{\partial \theta_2} & \dots & \frac{\partial q_w}{\partial \theta_6} \end{bmatrix}$$

De partiële afgeleiden kunnen analytisch berekend worden, maar dit is zeer veel rekenwerk. Aangezien je toch met een iteratieve methode werkt is het veel beter om de partiële afgeleiden ook te benaderen. Dit doe je door eerst de huidige end-effectortoestand op te slaan ( $F(\vec{G})$ ) en daarna een zeer kleine wijziging toe te brengen aan een element van de vector  $\vec{G}$ . Dit geeft een nieuwe vector  $\vec{G}'$  en als gevolg ook een nieuwe end-effectortoestand  $F(\vec{G}')$ . Door dan te kijken wat de verandering is tussen de elementen van  $F(\vec{G})$  en  $F(\vec{G}')$ , kan je zo de partiële afgeleide benaderen.

Zoals je kan zien is de jacobimatrix in dit geval niet inverteerbaar. Om de inverse te benaderen (de pseudoinverse van een matrix) heb ik gebruik gemaakt van de Eigen bibliotheek. Eigen is een bibliotheek in c++ voor wiskundige operaties zoals deze pseudoinverse.

Als de jacobimatrix berekent is en geïnverteerd is, kan ik de thetawaarden updaten met de definitie van een differentiaal:

$$\begin{aligned} dF(\vec{G}) &= J \times d\vec{G} \\ \Rightarrow d\vec{G} &= \alpha * J^{-1} \times dF(\vec{G}) \end{aligned}$$

De verandering van de end-effectortoestand  $dF(\vec{G})$  is gewoon het verschil tussen de gewenste end-effectortoestand (waar de robotarm naar toe probeert te werken) en de huidige end-effectortoestand  $F(\vec{G})$ . De onderste regel in bovenstaande vergelijking wordt ook de updatestap genoemd en voegt een extra parameter  $\alpha$  in. Die parameter wordt de learning rate genoemd en is noodzakelijk om zo het probleem zo snel en goed mogelijk op te lossen. Deze updatestap wordt keer op keer herhaald om zo uiteindelijk

de gewenste end-effectortoestand te bereiken.

Het voordeel van deze methode is dat je in principe geen motion planner nodig hebt. Aangezien je stap voor stap de thetawaarden optimaliseert zal de robotarm sowieso een pad volgen. Ook zijn er eenvoudige methodes om de updatestap te manipuleren zodanig dat de robotarm niet met zichzelf botst. Het nadeel hiervan is dat je geen invloed hebt op het pad. Je hebt bijvoorbeeld geen zekerheid dat de robotarm voor het snelste pad kiest.

Een tweede nadeel is het feit dat het zeer traag is en computationeel ook intensief is. Daarom heb ik uiteindelijk niet gekozen voor deze methode.

Het is ook redelijk moeilijk om dit proces optimaal te laten verlopen. Als je een te grote learning rate kiest, dan zal de robotarm zeer snel naar de gewenste toestand gaan, maar ook last hebben van overshoot en oscillatie. Maak je de learning rate te klein, dan zal de robotarm zeer traag naar de gewenste toestand gaan, maar niet accuraat zijn. Wel heb ik uit interesse deze methode eens geprogrammeerd in een simulatie in Unreal Engine. Het resultaat hiervan kan je bezichtigen op <https://tinyurl.com/y6kkn5wv>.

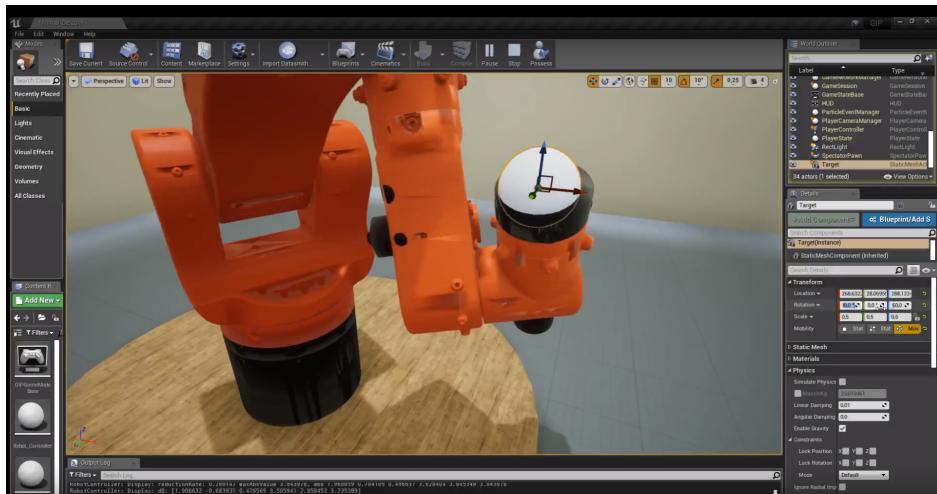


Fig. 58: Een screenshot van de test in Unreal Engine met de jacobimatrixmethode

## 8.9 Gradient descentmethode

Gradient descent is een optimalisatietechniek dat onder andere gebruikt wordt bij machine learning en kunstmatige intelligentie. De optimalisatietechniek probeert om een functie (waarbij het veel te moeilijk is om ze numeriek te beschrijven) te benaderen door stap voor stap een bepaalde kostfunctie te minimaliseren.

De functie waarbij het in mijn geval zeer moeilijk is om die numeriek te beschrijven, is de inverse kinematische functie  $\hat{F}(\vec{E}_g) = \vec{G}$  waarbij enkel  $\vec{E}_g$  gekend is.

De kostfunctie van mijn robotarm maakt gebruik van de gewenste end-effectortoestand  $\vec{E}_g$  en de huidige end-effectortoestand  $\vec{E}$ . De bedoeling van de kostfunctie is om aan te duiden hoe goed de huidige toestand de gewenste toestand benaderd. Je zou dus kunnen zeggen dat de kostfunctie gelijk is aan:

$$K(\vec{E}, \vec{E}_g) = \sum_{i=1}^6 (\vec{E}_{gi} - \vec{E}_i)$$

Maar deze kostfunctie zal overshoot en oscillatie in de hand werken. Veel beter is om gebruik te maken van een kwadratisch verband van die kostfunctie. Met andere woorden:

$$K(\vec{E}, \vec{E}_g) = \sum_{i=1}^6 (\vec{E}_{gi} - \vec{E}_i)^2$$

Als gevolg zal de robotarm bij een grote fout grote correcties doorvoeren en naarmate dat de fout afneemt ook kleinere correcties doorvoeren.

De bedoeling is uiteraard om de kostfunctie te minimaliseren. Minimaliseren van de kostfunctie doen we door de gradient ervan te berekenen. Wiskundig gezien is de gradient van de kostfunctie  $\vec{\omega} = [\frac{\partial K}{\partial \theta_1} \quad \frac{\partial K}{\partial \theta_2} \quad \frac{\partial K}{\partial \theta_3} \quad \frac{\partial K}{\partial \theta_4} \quad \frac{\partial K}{\partial \theta_5} \quad \frac{\partial K}{\partial \theta_6}]$ .

De partiële afgeleiden zijn analytisch gezien te moeilijk om exact te bepalen, maar aangezien we toch met een iteratieve methode werken is het beter om de partiële afgeleiden te benaderen. Je kan deze benaderen door eerst de kost van de huidige end-effectortoestand op te slaan, vervolgens een kleine aanpassing te doen aan een element in de joint state vector  $\vec{G}$  en te kijken welke invloed die verandering heeft op de kost.

De gradientvector kan je bekijken als de vector die de richting aangeeft in de huidige toestand langswaar de kostfunctie het snelst zal stijgen. Met andere woorden dus welke thetawaarden van  $\vec{G}$  je moet aanpassen om de kostfunctie zo snel mogelijk te laten stijgen. Wij willen dat de kostfunctie daalt, dus moeten we de thetawaarden zodanig aanpassen dat we een 'stap' nemen in de tegenovergestelde richting van de gradientvector. Wiskundig gezien ziet de update stap er als volgt uit:

$$d\vec{G} = -\alpha \vec{\omega}$$

Hier heb je opnieuw de extra parameter  $\alpha$  dat geoptimaliseerd moet worden. Het voordeel van deze methode is dezelfde als bij de jacobimethode, namelijk dat je geen motion planner nodig hebt. Een extra voordeel ten opzichte van de jacobimethode is dat deze methode veel sneller is en computationeel gezien ook minder zwaar is.

De nadelen en de gevolgen van een slecht gekozen learning rate blijven spijtig genoeg dezelfde. Er is zelfs een extra nadeel, namelijk het gevaar van lokale minima's. Als je een functie iteratief probeert te minimaliseren, dan kan het gebeuren dat het algoritme vast geraakt in een lokaal minima in plaats van het gewenste globaal minima zoals je in onderstaande afbeelding ziet. Bij deze toepassing bestaat dit probleem ook. Dit heeft als gevolg dat de robotarm kan denken de gewenste eindtoestand bereikt te hebben, maar er eigenlijk nog zeer ver van verwijderd is.

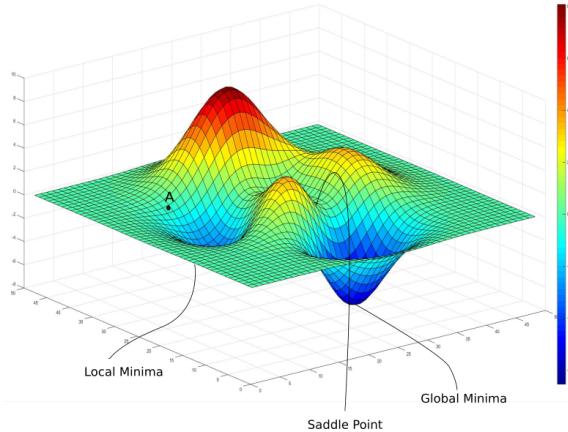


Fig. 59: Een voorstelling van een eenvoudige kostfunctie. Als de huidige toestand zich in punt A bevindt, is de kans groot dat het algoritme in het lokaal minima sukkelt in plaats van naar de gewenste toestand te gaan in het globaal minima.

Toch heb ik deze methode geprobeerd in mijn Unreal Engine simulatie waarvan je de resultaten hier kan zien: <https://tinyurl.com/yxfbck69>

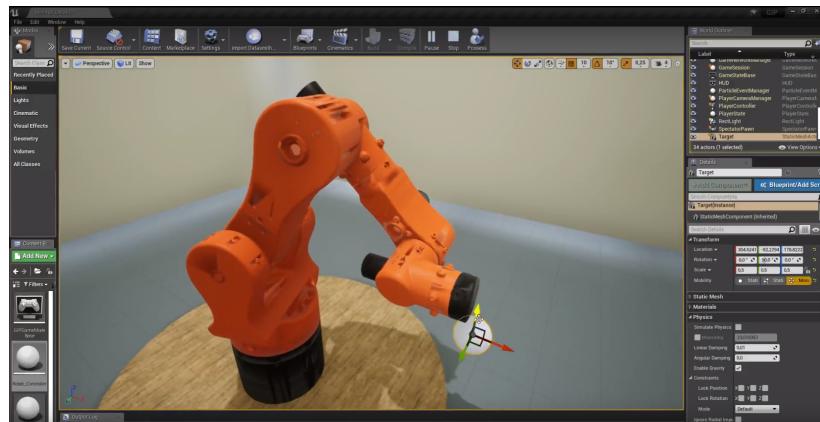


Fig. 60: Een screenshot van de test in Unreal Engine met de gradient descentmethode. De robotarm zit hier vast in een lokaal minima

*Naar: Wikipedia, Gradient descent*

## 8.10 IKFast

IKFast is de inverse kinematicasolver die ik uiteindelijk gebruik heb. Het is een bibliotheek dat automatisch de analytische formules van het inverse kinematica probleem op voorhand berekent. Het enige wat ik moet doen is een kinematisch model van de robotarm voorzien in een colladabestand (.dae). Een colladabestand kan je eenvoudig genereren op basis van een urdf-bestand. Een urdf-bestand is de kinematische beschrijving van de robotarm.

De analytische formules die de solver aanbiedt (in de vorm van c++ headers) zijn zeer snel berekenbaar en efficiënt. Wel heb je een motion planner nodig dat de 'beste' oplossing uitkiest en op basis daarvan een pad tussen de huidige toestand en de gewenste toestand uitstippelt. Wel biedt een motion planner je extra voordelen aan zoals het leggen van padbeperkingen en het veranderen van de uitvoersnelheid. De motion planner die ik gebruik is MoveIt! (zie hoofdstuk 11.3).

Wat wel een vereiste is vooraleer IKFast oplossingen kan vinden, is dat je met een zesassige robotarm werkt die ontkoppelbaar is in een ontkoppelpunt. Dat ontkoppelpunt laat IKFast toe om een aantal sterke vereenvoudigingen door te voeren. Het ontkoppelpunt kan elk punt in de ruimte zijn, zolang het maar aan de volgende voorwaarden voldoet:

- De ruimtelijke positie (rotatie maakt niet uit) van het punt is te berekenen enkel en alleen op basis van de eerste drie thetawaarden.
- De ruimtelijke positie (rotatie maakt niet uit) van het punt is te berekenen enkel en alleen op basis van de gewenste positie (rotatie maakt opnieuw niet uit) van de end-effector.

Concreet betekent dit voor mijn ontwerp dat de rotatieassen van de laatste drie assen concurrent moeten zijn. Het gemeenschappelijk punt is dan het ontkoppelpunt.

In de onderstaande afbeelding zie je een voorstelling van de rotatieassen van as 4 (rood), as 5 (groen), as 6 (blauw) en het ontkoppelpunt (paars).

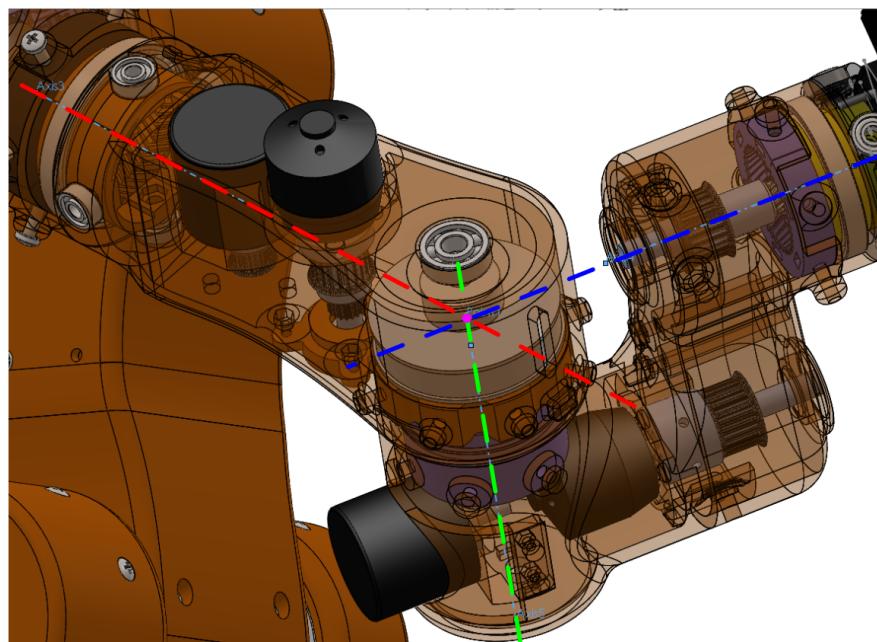


Fig. 61: Een voorstelling van de rotatieassen en het bijhorende ontkoppelpunt van as 4, 5 en 6

## 9 PID-loops

### 9.1 Inleiding

Een PID-controller (Proportioneel Integrerend en Differentiërend) is een closed-loop feedbacksysteem dat vaak toegepast wordt bij het regelen van een proces. De controller zal continu de fout  $e(t)$  berekenen als zijnde het verschil tussen de gewenste waarde en de gemeten waarde en zal een correctie doorvoeren op basis van de proportionele, integrerende en differentiërende term met als doel de error zo klein mogelijk te krijgen.

Een voorbeeld van zo'n systeem is de cruisecontrol van een wagen. De cruisecontrol heeft als taak om de effectieve snelheid van de wagen constant op de gewenste snelheid

te houden. Stel dat de wagen van een heuvel rijdt, dan zal de snelheid automatisch stijgen. De geïntegreerde PID-controller van de cruisecontrol zal hier op reageren en minder 'gas' geven om zo te compenseren voor de natuurlijke stijging in snelheid.

Een ander voorbeeld is het stabiel houden van de temperatuur van een 3D-printer. Afhankelijk van hoe snel er filament door de nozzle geduwd wordt zal er meer of minder stroom nodig zijn om de temperatuur constant te houden. De PID-controller is hier verantwoordelijk voor.

## 9.2 Werking

De onderstaande figuur toont een algemeen schema van een PID-controller. De PID-controller berekent de error  $e(t)$  op basis van de gewenste waarde  $r(t)$  en de gemeten waarde  $y(t)$ . Na het toepassen van een wiskundige berekening met een proportionele, integrerende en differentiërende term zal de PID-controller een controlewaarde  $u(t)$  hebben die gebruikt kan worden om het proces te regelen (bijvoorbeeld de hoeveelheid 'gas' bij de cruisecontrol) met als doel de error over tijd te doen dalen.

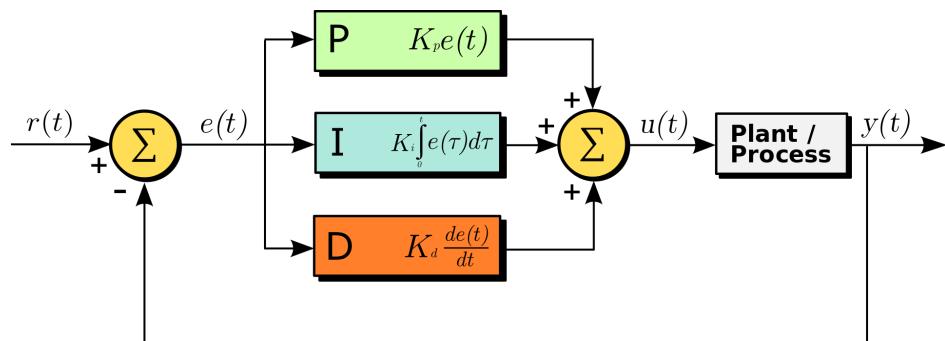


Fig. 62: Het theoretisch schema van een PID-controller

Bron: Wikipedia, *PID controller*

De verschillende termen hebben het volgende effect:

- **Proportioneel:** De proportionele term is, zoals de naam al zegt, proportioneel tot de error  $e(t) = y(t) - r(t)$  rekening houdend met de parameter  $K_p$ . Wiskundig gezien kan je zeggen dat

$$P = K_p * e(t)$$

Deze term is vooral verantwoordelijk voor het snel corrigeren van grote storingen op het proces. Is  $K_p$  te groot, dan zal de storing te agressief gecorrigeerd worden en zal je te maken krijgen met overshoot en als gevolg oscillatie. Als  $K_p$  te klein is, dan zal  $P$  zodanig klein worden naarmate  $e(t)$  kleiner wordt dat deze term praktisch geen effect meer zal hebben op de verbetering van  $e(t)$ . Daarom krijg je te maken met een SSE (Steady State Error) als je enkel deze term gebruikt in de controller. Een P-controller met andere woorden.

- **Integratorend:** De integrerende term is proportioneel met zowel de grootte van de error als de tijdsduur van de error. Het is de som van alle gemeten fouten over tijd. Eigenlijk zegt deze term dus iets over hoe veel 'foutmetingen' nog niet 'gecorrigeerd' zijn. Deze term, samen met de parameter  $K_i$ , kan wiskundig gezien worden als zijnde

$$I = K_i * \int_0^t e(t)dt$$

Deze term is vooral verantwoordelijk voor het wegwerken van de SSE. Deze term zal namelijk groter en groter worden naarmate de SSE aanwezig blijft en dus ook een effect hebben op  $u(t)$  en uiteindelijk op  $e(t)$ . Langs de andere kant kan het ook overshoot en oscillatie in de hand werken als  $K_i$  niet correct getuned is.

- **Differentiërend:** De differentiërende term wordt berekent door de differentiaal te nemen van  $e(t)$  op tijdstip  $t$  en daarna te vermenigvuldigen met de  $K_d$  parameter. Wiskundig gezien kan je  $D$  bepalen als volgt:

$$D = K_d * \frac{de(t)}{dt}$$

Aangezien de differentiaal van een functie die functie probeert te voorspellen, kan deze term er voor zorgen dat het systeem sneller stabiliseert en oscillatie dempt. Deze term is echter redelijk agressief en kan juist voor meer vibraties en (kleinere) oscillaties zorgen als er zeer regelmatig (op een hoge frequentie) kleine storingen voorkomen. Daarom wordt deze term meestal niet gebruikt.

Algemeen kan je stellen dat

$$u(t) = P + I + D = K_p * e(t) + K_i * \int_0^t e(t)dt + K_d * \frac{de(t)}{dt}$$

### 9.3 Tunen van de parameters

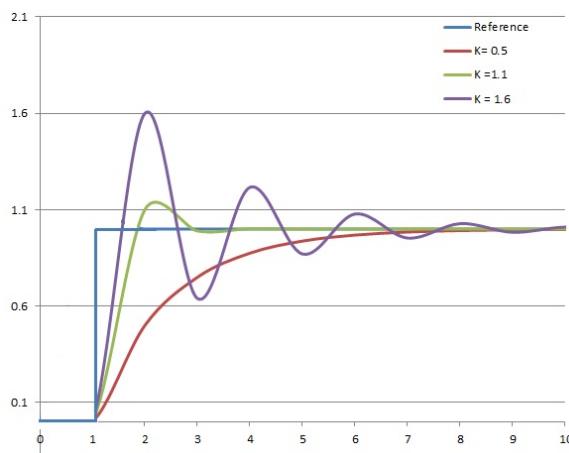


Fig. 63: De reactie van  $y(t)$  door een wijziging van  $r(t)$  met drie verschillende  $K_p$ -waarden

Bron: Wikipedia, *PID controller*

Voor een goede werking van deze controller dienen de parameters goed getuned te zijn. Dit is echter geen eenvoudig proces. Er bestaan een aantal algoritmes om deze parameters te tunen zoals het redelijk bekende Ziegler-Nichols algoritme. Daarmee zal het tuningsproces sneller resultaat leveren, maar in hoeverre dat algoritme helpt is afhankelijk van systeem tot systeem.

*Naar: Wikipedia, PID controller*

# 10 ODrive

## 10.1 Inleiding

ODrive is hardware dat gemaakt is voor het besturen van BLDC-motoren. Een ODrive kan tot twee motoren tegelijkertijd besturen, kan piekstromen aan tot meer dan 100A per motor, ondersteund verschillende feedbacksystemen, kan zowel met een batterij als met een power supply van 24V of 48V werken, kost relatief gezien niet veel én is open-source!

Je kan er mee communiceren over USB, met een simpele step/direction interface, via UART (goed voor Arduino), een PWM-signalen en een aantal andere communicatiemethodes die nu al op hardware-level mogelijk zijn, maar waarvan de firmware nog in volle ontwikkeling is. Het elektrische schema van deze ODrive is bijgevoegd als bijlage I.

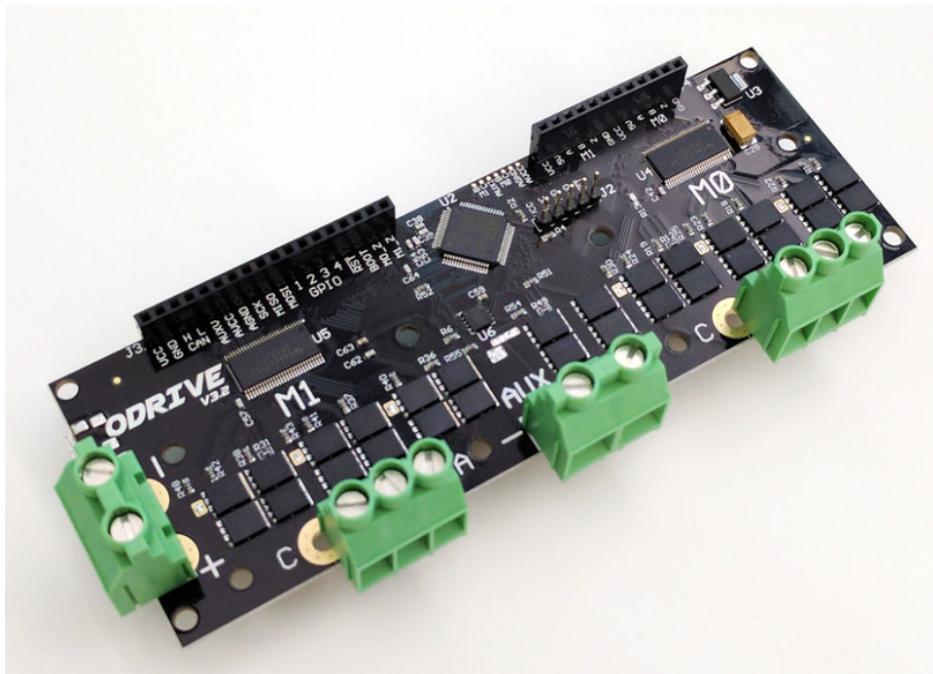


Fig. 64: De ODrive V3.2

Bron: madcowswe, *ODrive Documentation*

Voor feedback van de motoren te krijgen kan je gebruik maken van encoders (zowel type AB als ABI), hall sensoren of gewoon door het meten van de terug-EMK. Om de motor te commanderen kan je gebruik maken van een positiecontroller, snelheidscontroller, momentcontroller of van de ingebouwde trajectory planner die rekening houdt met acceleratie en deacceleratie.

Deze hardware levert goede prestaties, zeker voor wat het kost. Dit onderdeel is onmisbaar in mijn GIP. In dit hoofdstuk zal ik meer uitleg geven over hoe de ODrive werkt.

## 10.2 Hardware

In dit hoofdstukje zal ik meer informatie geven over de belangrijkste hardware van een ODrive. In de verschillende subhoofdstukken zal ik verwijzen naar onderstaande foto met behulp van cijfertjes zoals [3] of [6].

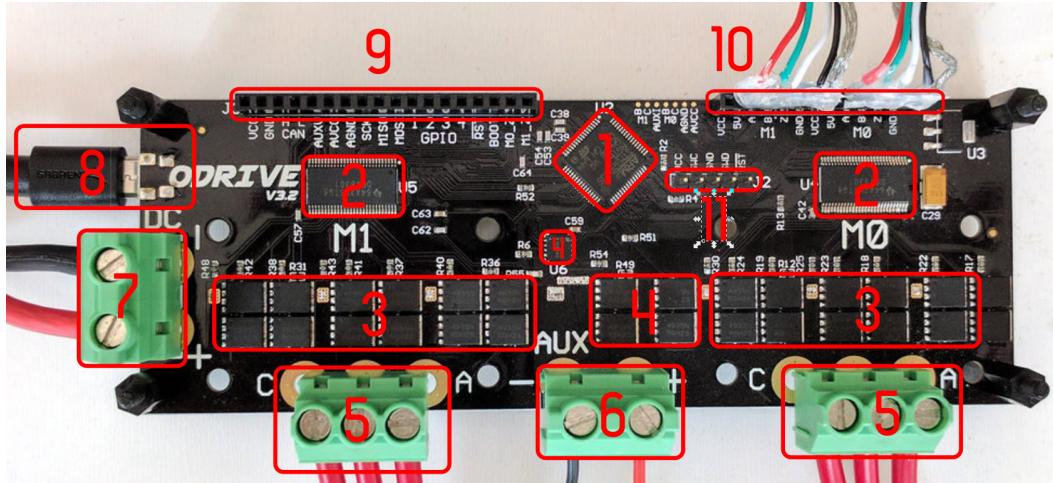


Fig. 65: De bovenkant van een ODrive

### 10.2.1 Microcontroller (1)

De microcontroller op dit moederbordje is een STM32 ontwikkeld door STMicroelectronics. Het is een 32-bit ARM microcontroller. Deze uitvoering van de STM32 (STM32F405RGT6) kan een kloksnelheid aanhouden van maximaal 168MHz. Deze hoge kloksnelheid (vergeleken met andere MCU's) en het feit dat het een 32-bit ARM MCU is maakt deze uiterst geschikt voor de ODrive. De firmware van de ODrive moet namelijk veel zware en preciese berekeningen uitvoeren om op hoge snelheid encoders uit te lezen, BLDC-motoren te commuteren, de PID-controllers up-to-date te houden, communiceren met andere hardware, enzovoort.

De STM32 kan bij de start onmiddelijk een programma uitvoeren vanuit zijn flashgeheugen. Firmware kan geschreven worden in c++. Om gecompileerde firmware naar het flashgeheugen te schrijven moet je gebruik maken van de ST-LinkV2 tool.

Als je meer informatie wil over de specificaties en de mogelijkheden van deze microcontroller, kan je een kijkje nemen in de datasheet.

<https://www.st.com/resource/en/datasheet/dm00037051.pdf>

### 10.2.2 Driefasige gate driver (2)

De twee aangeduide IC's zijn driefasige gate drivers, één per motor. Meerbepaald gaat het over de DRV8301 IC van Texas Instruments. Het apparaat biedt drie half-bridge drivers aan, elk capabel om maximaal twee externe N-channel MOSFETs te sturen. Een DRV8301 kan deze aansturen met een maximale stroom van 1.7A.

De DRV8301 bevat ook twee modules om accuraat stroom te meten. Het meten van de motorstroom is handig voor het besturen van de motor zonder feedback, sensorless control. De IC werkt op een ingangsspanning van 6-60V en heeft ook een geïntegreerde spanningsomvormer om bijvoorbeeld een MCU van stroom te voorzien. In het geval van ODrive is de DRV8301 van M0 verantwoordelijk voor de stroomvoorziening van de STM32.

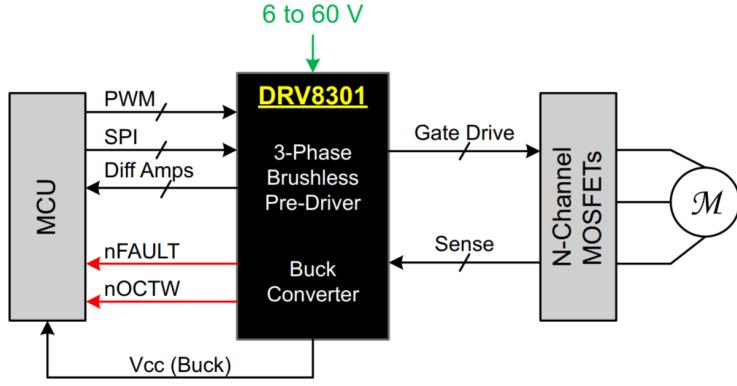


Fig. 66: Het versimpelde schema van een DRV8301

Bron: Texas Instruments

Deze predriver is een vereiste voor de ODrive omdat de STM32 niet in staat is om genoeg stroom te leveren om de echte MOSFETs te besturen. Voor meer informatie over dit onderdeel kan je de datasheet raadplegen.<http://www.ti.com/lit/ds/symlink/drv8301.pdf>

### 10.2.3 N-channel MOSFETs (output stage) (3)

Om de drie fases van de motor onder spanning te zetten, moet je gebruik maken van een driefasige H-brug. De driefasige H-brug dat de ODrive gebruikt kan je in onderstaande afbeelding zien. De GH\_X en GL\_x aansluitingen zijn de uitvoerlijnen van de DRV8301 en de SNX en SPX aansluitingen gaan naar de stroommeetmodules van de DRV8301. Het circuit kan eigenlijk elke fase aan de referentiespanning koppelen (0V, PGND) of aan de geleverde gelijkspanning (DCBUS), in mijn geval +24V. Zie het stukje over sinusvormige commutatie in hoofdstuk 1.3.2 voor een uitleg van hoe deze MOSFETs de motoren aansturen en waarom de referentiespanning 0V is.

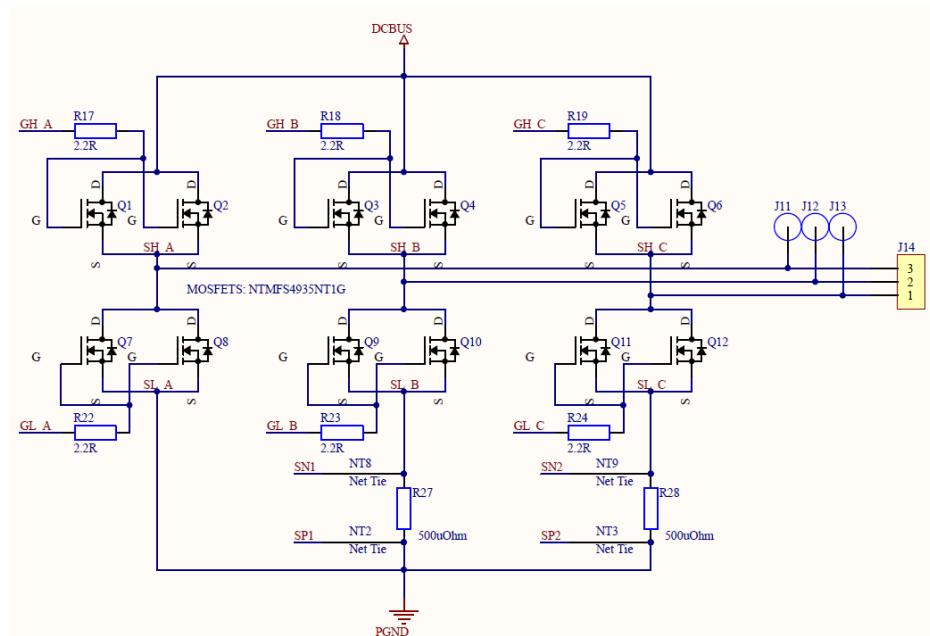


Fig. 67: De output stage van een ODrive

Bron: madcowswe, *ODrive Documentation*

De MOSFETs die gebruikt worden zijn er van het type NTMFS4935NT1G. Zoals je ziet zijn ze telkens dubbel uitgevoerd. Als de DRV8301 een MOSFET wil aansturen, stuurt het dus eigenlijk twee identieke MOSFETs aan die in parallel geschakeld zijn.

De datasheet van de MOSFETs kan je op de onderstaande link terugvinden:  
<https://www.onsemi.com/pub/Collateral/NTMFS4935N-D.PDF>

#### 10.2.4 N-channel MOSFETs (vermogensweerstand) (4)

Deze vier MOSFETs van hetzelfde type als de rest zijn verantwoordelijk voor het onder spanning zetten van de vermogensweerstand. Deze MOSFETs zijn ook dubbel uitgevoerd door er telkens twee in parallel te schakelen. Ze zijn niet aangedreven door een DRV8301, maar wel door een LM5109BSD/NOPB van Texas Instruments. Deze chip heeft hetzelfde doel als een DRV8301, maar is compacter, heeft minder functies en lagere specificaties. Voor het aandrijven van de vermogensweerstand zijn die extra functies en hogere specificaties helemaal niet nodig. Meer informatie over de LM5109BSD/NOPB kan je vinden in de datasheet <http://www.ti.com/lit/ds/symlink/lm5109b.pdf>

Als je de motor laat remmen of laat uitlopen, dan zal deze motor zich als een generator gaan gedragen. De kinetische energie van de motor wordt met andere woorden terug omgevormd naar elektrische energie. Aangezien de sturing zelf zeer weinig stroom verbruikt, zal die extra elektrische energie proberen om terug naar de bron te gaan. Als gevolg zal de ingangsspanning van de ODrive stijgen. Als je met batterijen werkt kan dit gebruikt worden om de batterijen terug op te laden en zo het systeem efficiënter te maken. Maar als je werkt met een power supply (wat in mijn geval zo is) dan is deze stijging van de ingangsspanning zeer slecht voor de power supply.

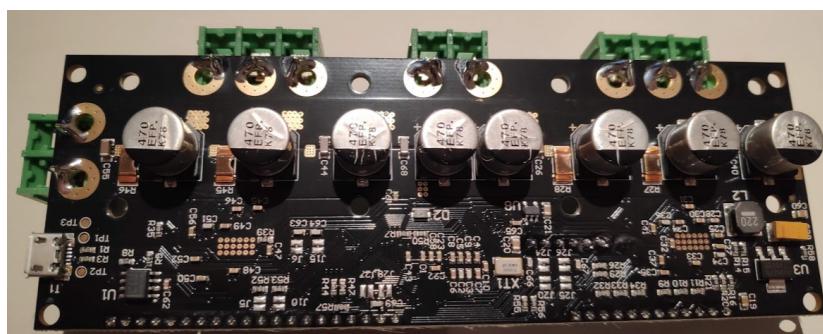


Fig. 68: De onderkant van de ODrive met de rij condensatoren

Bovendien zijn er op de onderkant van de ODrive een hoop condensatoren parallel op de ingangsspanning geplaatst om rimpelspanningen weg te werken en de spanningsvallen als gevolg van stroompieken te beperken. Deze condensatoren hebben op de 24V versie van de ODrive een maximale spanningswaarde van 25V. Dat is dus nog een reden waarom het geen goed idee is om de ingangsspanning te laten stijgen.

ODrive lost dit probleem op door deze opgewekte elektrische energie dankzij de wet van Joule in warmte om te zetten met behulp van een vermogensweerstand. Een vermogensweerstand is een passieve weerstand met een zeer lage ohmse waarde, maar wel met een groot dissipatievermogen. De vermogensweerstanden die ik gebruik hebben een weerstandswaarde van  $0.47\Omega$  en een dissipatievermogen van 50W.



Fig. 69: Een vermogensweerstand

### 10.2.5 Connectoren, IO en communicatie (5-11)

De terminals met nummer [5] worden gebruikt om de twee motoren aan te sluiten, op terminal [6] kan je de vermogensweerstand aansluiten en terminal [7] wordt gebruikt om de ingangsspanning (van de power supply) op aan te sluiten. Nummer [8] is een USB-poort voor te communiceren met een computer, header [9] wordt gebruikt voor de andere vormen van communicatie (step/dir, UART, ...), header [10] kan gebruikt worden om de encoders aan te sluiten en header [11] is de header waar je de ST-LinkV2 tool op moet aansluiten voor nieuwe firmware te flashen.

## 10.3 Firmware

### 10.3.1 Controlelus

Met controleloop bedoel ik het volledige systeem dat een positie- of snelheidscommando als input neemt en op basis van feedback de motoren bestuurt. Indien je met een encoder werkt, is deze controlelus volledig closed-loop.

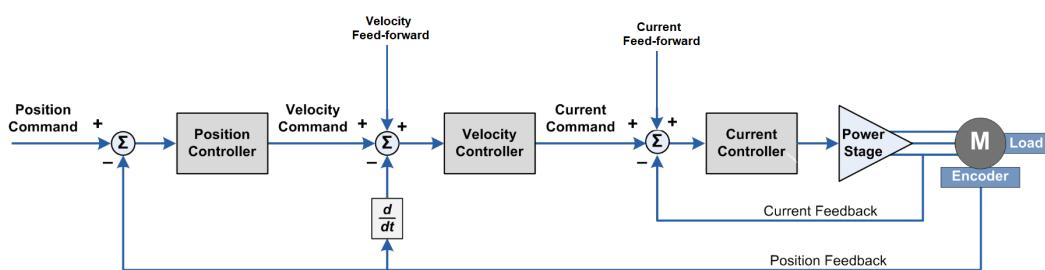


Fig. 70: De controlelus van ODrive

Bron: madcowswe, *ODrive Documentation*

De controleloop bestaat uit drie controllers: een positiecontroller, een snelheidscontroller en een stroomcontroller. Deze controllers zorgen ervoor dat je zowel positie-, snelheids- als stroomcommando's kan geven. De snelheidscontroller controleert ook dat de

maximale snelheid niet overschreden wordt en de stroomcontroller past indien nodig stroombegrenzing toe.

**Positiecontroller** Als je de ODrive ingesteld hebt op closed-loop positiecontrole, zal deze controller eerst zijn werk doen. Het is een simpele P-controller (zie hoofdstuk 9) dat de error berekent op basis van feedback van de encoder en het nieuwe positiecommando, uitgedrukt in counts. Recht evenredig met de pos\_gain parameter zal die een snelheidscommando als output geven.

**Snelheidscontroller** De snelheidscontroller krijgt als input de som van het snelheidscommando en een optionele feed-forwardsnelheidsparameter. Als je aan closed-loop positiecontrole doet komt het snelheidscommando van de positiecontroller, als je aan closed-loop snelheidscontrole doet dan zal de positiecontroller niet actief zijn en zal het snelheidscommando rechtstreeks afkomstig zijn van de commander (een gebruiker, andere software, ...).

De snelheidscontroller zelf is een PI-controller (zie hoofdstuk 9). De snelheidsfout wordt berekend op basis van het snelheidscommando en de afgeleide van de positiefeedback van de encoder, namelijk de snelheid van de encoder. Het proportioneel gedeelte werkt met de vel\_gain parameter en het integrerend gedeelte werkt met de vel\_integrator\_gain parameter. Deze controller zal na de PI-controller een stroomcommando naar de stroomcontroller sturen.

**Stroomcontroller** De stroomcontroller krijgt als input de som van het snelheidscommando en een optionele feed-forwardstroomparameter. Er is een mogelijkheid om aan closed-loop stroomcontrole te doen, wat ook veel gedaan wordt in de industrie, maar het is zeer moeilijk om de bijhorende parameters correct te tunen en de simulatiesoftware moet zo'n interface natuurlijk ook ondersteunen.

Deze stroomcontroller zelf is opnieuw een PI-controller. De stroomfout wordt berekend op basis van het stroomcommando en de door de DRV8301 gemeten fasestroom. De bijhorende parameters van deze PI-controller zijn current\_gain voor het proportioneel gedeelte en current\_integrator\_gain voor het integrerend gedeelte. Deze controller zal daarna een spanningscommando sturen naar de power stage dat verantwoordelijk is voor de sinusvormige commutatie van de fasespanningen.

## 10.4 Trapeziumvormige trajectory planner

Je kan de motor direct besturen door positie- of snelheidscommando's te sturen, maar deze commando's houden geen rekening met fysieke acceleratielimieten van de toepassing. Daarom biedt de firmware ook een trapeziumvormige trajectory planner aan. Deze trapeziumvormige trajectory planner zal op basis van een positiecommando volautomatisch getimed positiecommando's naar de positiecontroller sturen om zo geleidelijk aan volgens een acceleratieparameter de snelheid op te drijven tot de maximaal ingestelde snelheid en daarna geleidelijk aan volgens een deacceleratieparameter de snelheid te laten afnemen.

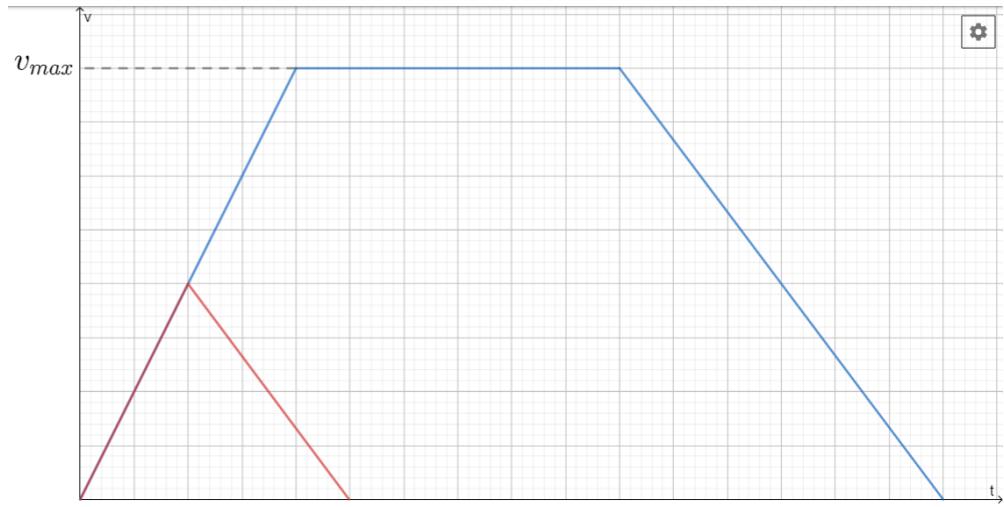


Fig. 71: De trapeziumvormige trajectory planner van ODrive. De blauwe lijn is een lange beweging, de rode lijn is een korte beweging

Bovendien biedt deze planner ook een aantal extra veiligheden aan. De planner houdt rekening met de feedback van de encoder en van de controlelus. Als hij een abnormale wijziging merkt (snelheid dat plots sterk wijzigt, positie dat te veel afwijkt van het geplande pad, ...) dan zal hij dat kanaal van de ODrive in veiligheid brengen.

Aangezien ik origineel van plan was om de simulatiesoftware te schrijven in Unreal Engine, moest ik een eenvoudigere versie van dit mechanisme zelf implementeren. Uiteindelijk heb ik het niet gebruikt maar de geïnteresseerden kunnen mijn berekeningen bekijken in bijlage C.

## 11 ROS

### 11.1 Inleiding

Software schrijven voor een roboticaproject is moeilijk en vergt veel tijd. Daarom dat ik de simulatiesoftware geschreven heb met een framework dat een reeks van voorgebouwde pakketten, tools en conventies aanbied, namelijk ROS.

ROS, of Robot Operating System is een open-source flexibel framework om software te schrijven voor roboticatoepassingen. Het is origineel ontwikkeld in 2000 door Stanford University als prototype voor hun roboticaprogramma's. In 2007 heeft Willow Garage samen met andere onderzoekers de ideeën voor ROS verder uitgebreid en de fundamentele ROS-pakketten ontwikkeld. Dit allemaal onder de open-source BSD-licentie. Momenteel wordt ROS onderhouden door de Open Source Robotics Foundation en is het het meest gebruikte framework in de roboticacommmunity. Ook wordt het dankzij de flexibiliteit zeer breed toegepast: van een klein robotje op de bureau van een hobbyist tot de automatisatie van een industrieel proces.

Zoals de naam al doet vermoeden is ROS gebouwd op de concepten van een besturingssysteem. Het biedt low-levelcommunicatie met hardware aan, alsook een process manager, een pakketmanager, een complexe manier om verschillende processen met elkaar te laten communiceren, etc... Maar toch is ROS (nog) niet standalone uitvoerbaar als besturingssysteem. Het is zoals eerder al aangehaald gebaseerd op de *concepten* van een besturingssysteem. ROS werkt afhankelijk van de versie op een bepaalde

linuxdistributie. Ik gebruik ROS Kinetic omdat dit de laatste stabiele release is. ROS Kinetic draait het beste op Ubuntu 16.04 LTS. Die linuxdistributie staat dan ook als tweede besturingssysteem op mijn laptop, naast Windows 10.

## 11.2 Fundamentele concepten

### 11.2.1 ROS Packages

Software in ROS is ingedeeld in pakketten. Een pakket kan nodes bevatten, een onafhankelijke bibliotheek dat niets te maken heeft met ROS, een dataset, configuratiebestanden, ... Eigenlijk alles dat logisch is om in een aparte 'module' te steken. Het doel van alles in een pakket te steken is om er voor te zorgen dat de software gemakkelijk gedeeld kan worden met andere mensen. Die kunnen op hun beurt het stukje software opnieuw implementeren in hun eigen ROS omgeving, zonder al te veel aanpassingen te moeten doorvoeren.

Pakketten zijn makkelijk te maken met de hand of met de catkin tool (catkin is de compiler, zie verder in dit hoofdstuk) *catkin\_create\_pkg*.

Alle pakketten volgen conventies die opgelegd zijn door ROS. De bestandenstructuur van een pakket kan er als volgt uitzien:

```
voorbeeld_pakket/
  config/
    controllers.yaml
    joint_limits.yaml
    ...
  launch/
    move_group.launch
    rviz.launch
    ...
  urdf/
    robot.urdf
    ...
  scripts/
    runTrajectory.py
  CMakeLists.txt
  package.xml
```

**config/** In deze map zitten de .yaml bestanden die bepaalde configuraties aanbieden. Zo bepaald controllers.yaml welk type controller (positiecontroller, snelheidscontroller, etc.) gebruikt kan worden voor welke as. Of joint\_limits.yaml is verantwoordelijk voor het vastleggen van snelheids- en acceleratielimieten van elke as.

**launch/** Deze map bevatten de .launch bestanden van het pakket. Deze bestanden zijn verantwoordelijk om nodes (processen) op te starten (zie hoofdstuk 11.2.2), de juiste parameters in te laden in de parameter server (zie hoofdstuk 11.2.7) en om Publishers/Subscribers op te starten (zie hoofdstuk 11.2.3 en 11.2.4). rviz.launch is bijvoorbeeld verantwoordelijk voor het opstarten van een visualisatietool en move\_group.launch voor de motion planner.

**urdf/** Hier worden alle urdf-bestanden opgeslagen. URDF staat voor Unified Robot Description File. Een urdf-bestand is dus eigenlijk een genormaliseerde beschrijving van

een robotarm, van de relaties tussen alle joints en linken. De Denavit-Hartenbergconventie wordt o.a. hier in toegepast.

**scripts/** Scripts zijn uitvoerbare bestanden. Scripts worden voornamelijk gebruikt voor programma's die voor interactie met de gebruiker zorgen. Zo kan runTrajectory.py bijvoorbeeld een interface lanceren dat losstaat van de simulatie met als doel de robotarm een vooraf bepaald pad te laten uitvoeren.

**CMakeLists.txt en package.xml** Deze bestanden worden gebruikt door de compiler van ROS, namelijk catkin. Hierin wordt beschreven van welke bibliotheken het pakket afhankelijk is, op welke andere rospakketten het pakket steunt, waar de header files staan voor c++ scripts, enzovoort.

*Naar:* Open Source Robotics Foundation, *Packages - ROS Wiki*

### 11.2.2 Nodes

Een node is een proces dat bewerkingen uitvoert. Nodes zijn samengevoegd in een netwerk en communiceren met andere nodes via topics, services, en de globale parameterserver. Zie hoofdstukken 11.2.3, 11.2.5 en 11.2.7 voor meer informatie over de communicatie tussen nodes.

Het gebeurt vaak dat er veel nodes actief moeten zijn om een robot te besturen. Het gebruik van nodes in ROS voorziet daarom veel voordelen in het systeem. Het systeem zal bijvoorbeeld beter fouten kunnen verwerken, omdat de fouten geïsoleerd zijn in een node. Complexiteit van de code is ook gereduceerd. En omdat een node beperkte functionaliteit aanbiedt is het maar afhankelijk van een deel van de ROS API. Daarom is het gemakkelijk implementeerbaar in andere ROS omgevingen.

Alle actieve nodes hebben een unieke naam dat hen identificeerbaar maakt voor de rest van het systeem. Zo is bijvoorbeeld de node '/motion\_planning\_node' verantwoordelijk voor de motion planning. Nodes hebben ook een node type, dat het lanceren van nodes vereenvoudigd. Deze node types zijn meestal de naam van het pakket waartoe de node behoort.

Een ROS node wordt geschreven met behulp van de ROS client bibliotheken, zoals roscpp of rospy.

*Naar:* Open Source Robotics Foundation, *Nodes - ROS Wiki*

### 11.2.3 Topics

ROS biedt uiteraard een manier aan om pakketten (eigenlijk nodes) met elkaar te laten communiceren. Dat doet het voornamelijk via het rostopic pakket. Dit pakket biedt een publisher/subscriber communicatiemodel aan.

Een topic kan je zien als een onderwerp waarover gecommuniceerd wordt. Zo heb je bijvoorbeeld de topic '/planned\_path' dat gebruikt kan worden om het geplande pad op te slaan of '/gripperstate', dat gebruikt zou kunnen worden om aan de simulatie te laten weten dat de grijper al dan niet gesloten is.

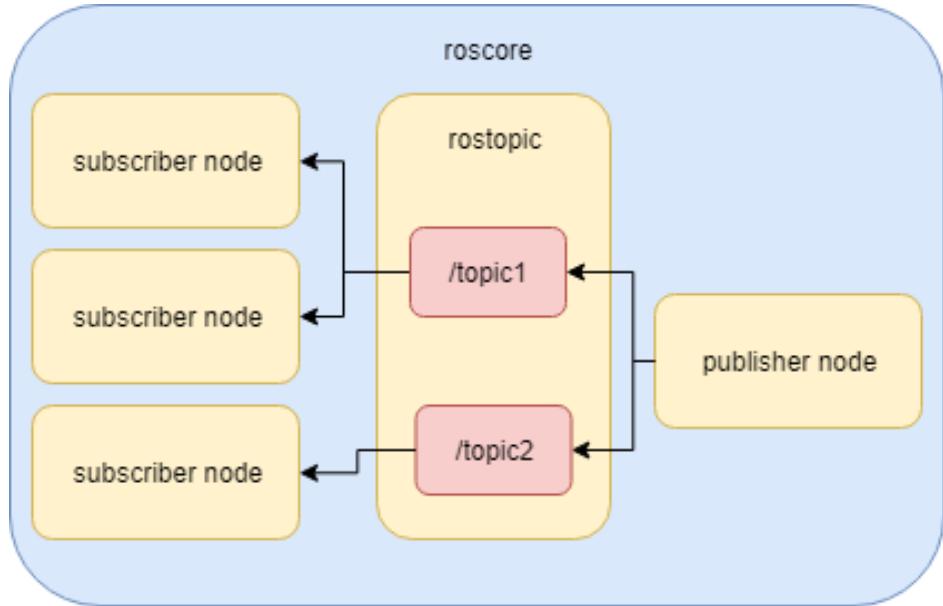


Fig. 72: Theoretische structuur van communicatie met topics

Een node kan via een publisher een nieuwe entry naar een topic publishen. Deze nieuwe entry wordt dan beschikbaar gesteld aan elke subscriber van die topic.

Nodes kunnen zowel publiceren als abonneren op hetzelfde moment, er kunnen meerdere nodes tegelijk publiceren naar eenzelfde topic of er kunnen meerdere nodes abonneren op eenzelfde topic. Een publisher weet niet welke nodes de nieuwe entry te zien zal krijgen. Subscribers weten ook niet van welke publisher ze de nieuwe entry ontvangen hebben. Dit zorgt ervoor dat rostopic heel abstract is en ook heel flexibel. Als er bijvoorbeeld topics aangemaakt worden voor de communicatie tussen de motion planner en de inverse kinematicasolver, dan kan je tijdens de runtime de inverse kinematicasolver veranderen zonder dat de motion planner daar iets van merkt. Op voorwaarde dat die nieuwe solver dezelfde topicconventies gebruikt.

De data dat gebruikt wordt door topics moet volgens een vaste structuur bepaald zijn. Zo'n vaste structuur wordt ook wel een message genoemd (zie hoofdstuk 11.2.4)

Uiteraard moet er ook wel een systeem zijn dat alle topics registreert en de nodes van een centrale databank voorziet. Hiervoor is de Master node verantwoordelijk, afkomstig van het roscore pakket. (Zie hoofdstuk 11.2.6)

**Transport van topics** ROS stuurt momenteel berichten via topics op basis van het TCP/IP- en het UDP-protocol. Het aangepaste TCP/IP-protocol wordt TCPROS genoemd en zendt berichten over TCP/IP verbindingen. Het is de standaardtransportmethode in ROS en is ook de enige methode dat bibliotheken moeten ondersteunen. Het andere protocol (gebaseerd op het UDP-protocol, ook wel UDPROS genoemd) is momenteel enkel ondersteund in de rosccp bibliothek.

De nodes onderhandelen over de transportmethode tijdens runtime. Als een node een verzoek maakt naar een andere node om te communiceren met UDPROS, maar tegelijkertijd merkt dat de andere node enkel TCPROS ondersteund, dan kan de eerste node altijd terugvallen op TCPROS. Dit zorgt ervoor dat je gemakkelijk nieuwe

transportmethodes kan toevoegen zonder te eisen dat andere pakketten die nieuwe methode ook moeten ondersteunen.

*Naar:* Open Source Robotics Foundation, *Topics - ROS Wiki*

**Voorbeeld** Om deze toch wel belangrijke principes duidelijk te maken ga ik in deze paragraaf een simpel voorbeeld geven. Ik heb twee eenvoudige scripts geschreven en gecompileerd onder het pakket "communicatie\_voorbeeld". Het ene script zal op de topic '/voorbeeldtopic' 10 keer per seconde "Hallo wereld" publishen, terwijl het andere script (de Subscriber) zal luisteren naar dezelfde topic en elke nieuwe entry zal uitprinten naar de console.

Als msg maak ik gebruik van het datatype 'std\_msgs/String'. Deze laat je toe om een String te sturen, namelijk een verzameling van tekens.

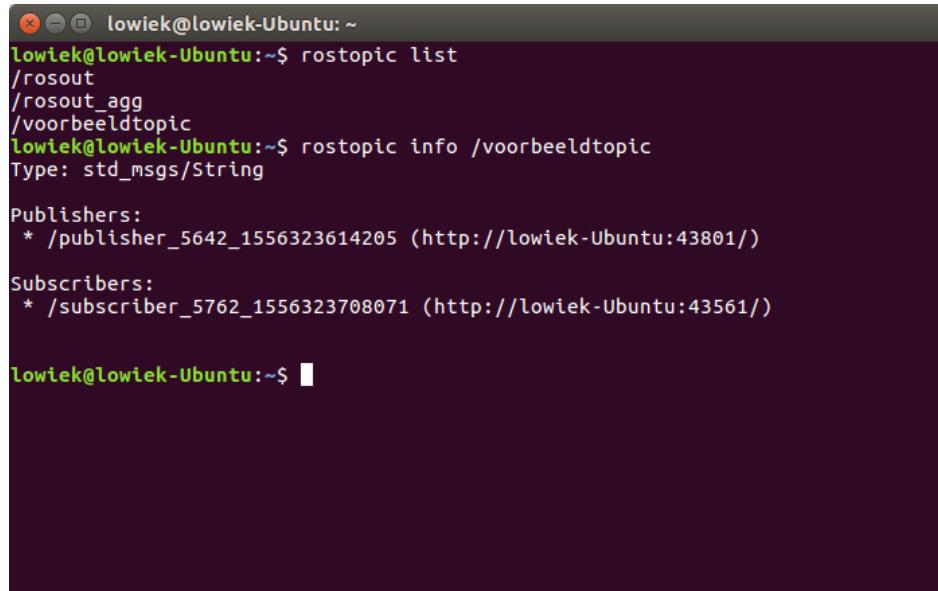
Eerst en vooral moet ik een terminal openen en `$ roscore` uitvoeren. Dit zal de master lanceren. Daarna lanceer ik in nieuwe terminals de publisher- en de subscribernode door in hun respectievelijke terminal een van de volgende commando's in te geven:

```
$ rosrun voorbeeld_pakket publisher.py
$ rosrun voorbeeld_pakket subscriber.py
```

The screenshot shows two terminal windows side-by-side. The left terminal window, titled 'lowiek@lowiek-Ubuntu: ~/ws\_moveit/src/voorbeeld\_pakket/scripts', displays the output of the publisher.py script. It shows multiple INFO messages at different timestamps, each containing the string 'Hallo wereld!'. The right terminal window, also titled 'lowiek@lowiek-Ubuntu: ~/ws\_moveit/src/voorbeeld\_pakket/scripts', displays the output of the subscriber.py script. It shows the same INFO messages from the publisher, but they are preceded by the prefix '[INFO] [1556323815.446778]:', indicating they were received by the subscriber node.

Fig. 73: Van links naar rechts: subscriber.py, publisher.py

Nu kan je in bovenstaande figuur zien dat de Publisher telkens een nieuwe entry published en dat de Subscriber deze entry binnenhaald en terug uitprint in de terminal. Als ik nu een vierde terminal open en daarin `$ rostopic list` uitvoer, dan zal je in onderstaande figuur zien dat de topic '/voorbeeldtopic' geregistreerd is. Als ik daarna `$ rostopic info /voorbeeldtopic` uitvoer, dan zie je dat er zoals verwacht 1 publisher en 1 subscriber van deze topic is.



```

lowiek@lowiek-Ubuntu: ~
lowiek@lowiek-Ubuntu:~$ rostopic list
/rosout
/rosout_agg
/voorbeeldtopic
lowiek@lowiek-Ubuntu:~$ rostopic info /voorbeeldtopic
Type: std_msgs/String

Publishers:
* /publisher_5642_1556323614205 (http://lowiek-Ubuntu:43801/)

Subscribers:
* /subscriber_5762_1556323708071 (http://lowiek-Ubuntu:43561/)

lowiek@lowiek-Ubuntu:~$ ■

```

Fig. 74: Lijst met actieve topics en informatie over '/voorbeeldtopic'

#### 11.2.4 Messages

De data die je naar een topic kan sturen wordt een message genoemd. Een message is een speciaal datatype dat je kan maken met behulp van het rosmsg pakket. In het algemeen ziet de definitie van een msg (een .msg bestand) er als volgt uit:

```

Header header

constanttype1 CONSTANTNAME1 = constantvalue1
constanttype2 CONSTANTNAME2 = constantvalue2

fieldtype1 fieldname1
fieldtype2 fieldname2
fieldtype3 fieldname3

```

**Header** De header is zelf een message datatype. Het is een niet-verplicht onderdeel van het bestand dat je toelaat om id's in te stellen of om timestamps toe te voegen.

**Fields** Daarna kan je 'fields' toevoegen. Dat zijn placeholders die je kan gebruiken om het echte msg object op te vullen met data. FieldtypeX stelt het datatype van een field voor. Dit kan een standaarddatatype zijn van c++ zoals int of float, maar ook even goed een ander message datatype van een andere rospackage. Zoals je ziet kan je dus heel ver met dit datatype gaan.

**Constanten** Constanten kunnen simpelweg bepaald worden door een field een waarde te geven. Deze speciale fields kunnen door een publisher dus ook niet ingevuld worden.

**Compilatie** Als je deze definitie onder de 'msg' folder van een pakket plaatst en de juiste verwijzingen maakt in package.xml en CMakeList.txt, dan zal er bij het builden van het pakket headers en source code gegenereerd worden door het rosmsg pakket. Die headers en source code kunnen dan geïmporteerd worden door subscribers en publishers om ze te gebruiken tijdens hun communicatie via topics of services.

**Voorbeeld** Stel dat je een topic maakt dat een driedimensionaal punt moet vasthouden. Dan kan je bijvoorbeeld punt.msg aanmaken in het pakket "meetkunde\_msgs". Deze kan er dan als volgt uitzien:

```
Header header
```

```
float32 x
float32 y
float32 z
```

Zoals je ziet zijn er drie fields om een punt te bepalen. Na het compileren van het "meetkunde\_msg" pakket is dit .msg bestand geregistreerd als "meetkunde\_msgs/punt" en importeerbaar in c++ of python scripts.

*Naar: Open Source Robotics Foundation, msg*

### 11.2.5 Services

Het publish-subscribemodel met Topics is een zeer flexibele manier van communicatie, maar omdat de data gecentraliseerd en geanonimiseerd wordt is het niet de optimale communicatiemethode voor RPC-verzoeken (Remote Procedure Call). Daarom biedt ROS services aan. Een service is essentieel een koppel van messages. Een message voor het verzoek en een message voor het antwoord. Een node dat verantwoordelijk is voor de service biedt deze service aan en een client (in een andere node) kan naar deze service een verzoek sturen en wachten op het antwoord.

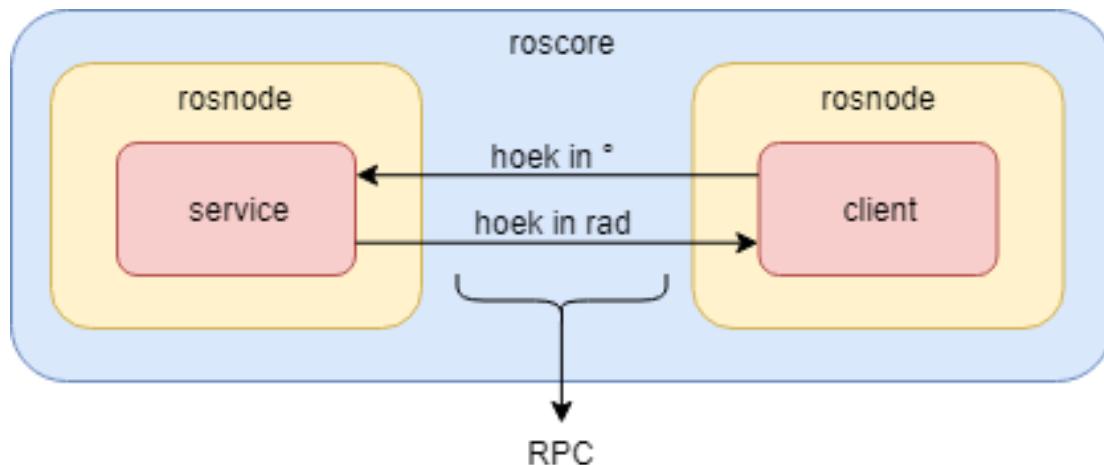


Fig. 75: Voorbeeld van een service dat graden omzet naar radialen

Dit wordt bijvoorbeeld toegepast bij de communicatie tussen de motion planner en de hardwarecontrollers. De motion planner kan een traject (type trajectory\_msgs/JointTrajectory) naar de FollowJointTrajectory action service sturen, die dan zal antwoorden met feedback over de status van het uit te voeren traject.

*Naar: Open Source Robotics Foundation, Services - ROS Wiki*

### 11.2.6 Master

De ROS Master voorziet een aantal naam- en registratieservices voor de andere nodes in de ROS-omgeving. Het houdt publishers en subscribers van topics en services bij. Het

is eigenlijk een centraal punt waar nodes kunnen zoeken. Vanaf dat een node een andere node gevonden heeft, zullen de nodes direct met elkaar communiceren, zonder de master als tussenpersoon.

De Master node beheert ook de Parameter Server.

**Voorbeeld van Master registratieservices** Stel dat we twee nodes hebben: een cameranode en een image\_viewernode. Eerst zou de cameranode de Master node verwittigen dat het wilt publishen op de topic '/images'.

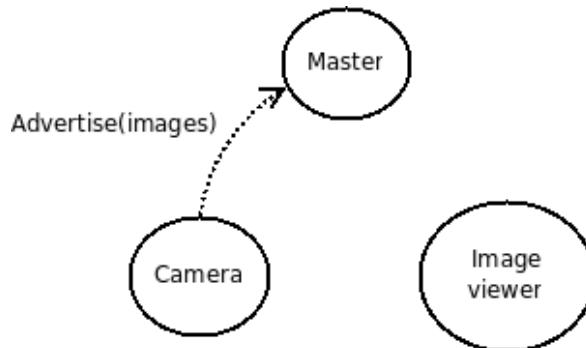


Fig. 76: Verzoek van publisher tot het registreren van de '/images' topic

Bron: Open Source Robotics Foundation, *Master - ROS Wiki*

De cameranode zal afbeeldingen publishen naar de '/images' topic, uiteraard zonder te weten wie er op geabonneerd is of wie er nog naar '/images' published. De master ziet dat er niemand geabonneerd is op de '/images' topic, dus er wordt geen data verstuurd. In de onderstaande figuur zie je dat de subscriber node een verzoek maakt aan de Master node om subscriber te worden van de '/images' topic.

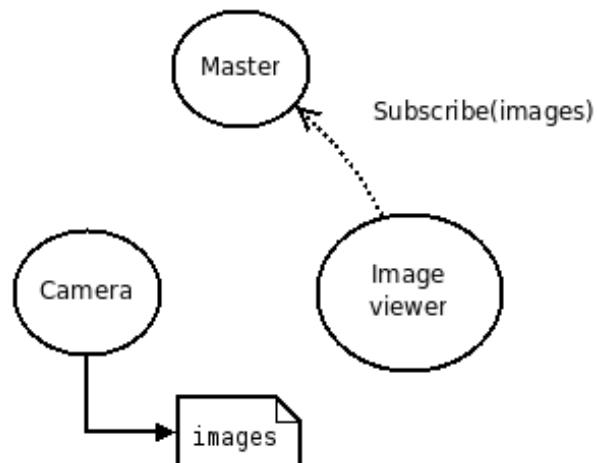


Fig. 77: Verzoek van node om te subscriben op de '/images' topic

Bron: Open Source Robotics Foundation, *Master - ROS Wiki*

Nu de Master node ziet dat er zowel een publisher als een subscriber aanwezig is van de '/images' topic, zal hij beide hiervan op de hoogte brengen en de communicatie tussen de twee nodes op gang trekken.

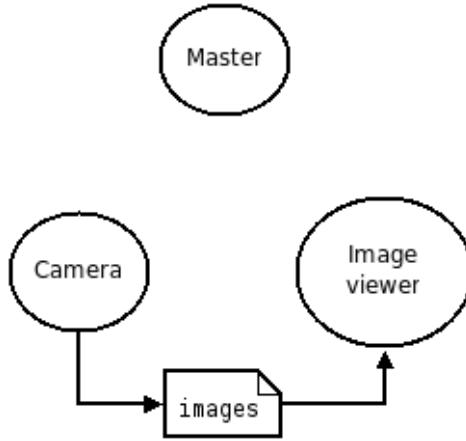


Fig. 78: Overgang naar een node-to-node communicatievorm

Bron: Open Source Robotics Foundation, *Master - ROS Wiki*

Naar: Open Source Robotics Foundation, *Master - ROS Wiki*

### 11.2.7 Parameter Server

Een parameter server kan gezien worden als een gedeeld woordenboek dat toegankelijk is via normale internet API's. Nodes gebruiken deze server om parameters op te slaan en terug te halen tijdens runtime. Het is niet geoptimaliseerd voor snelheid, dus parameters zijn het best constant. Parameters zijn ook altijd globaal zichtbaar voor elke door de Master geregistreerde node.

In de onderstaande figuur zie je een voorbeeld van een ROS omgeving dat gebruik maakt van de parameter server. Het voorbeeldje leest de temperatuur af van een API, zet deze om naar fahrenheit (omdat de user dit zo wenst) en toont het in een GUI.

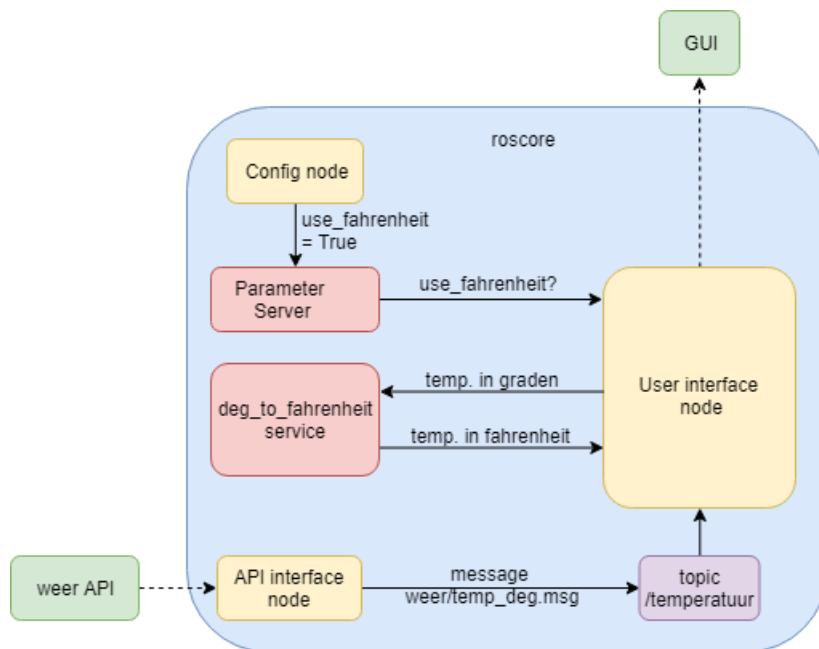


Fig. 79: Overzicht van een ROS omgeving met de parameter server

De config node leest configuratiewaarden uit een .yaml-bestand en plaatst deze in de parameter server. In dit geval gaat het om de use\_fahrenheit parameter dat dankzij het

.yaml-bestand op 'True' ingesteld wordt.

De API interface node krijgt de temperatuur (in graden) van een externe API. Hij published de temperatuur naar een '/temperatuur' topic. De user interface node is subscriber van die topic en zal dus de nieuwe entry ontvangen. De user interface node weet vanuit de parameter server dat de gebruiker wenst de temperatuur in fahrenheit te lezen in plaats van in graden. Daarom doet deze node eerst nog een RPC naar de deg\_to\_fahrenheit service dat een temperatuur in graden (via de weer/temp\_deg message) ontvangt en de temperatuur in fahrenheit (weer/temp\_fahrenheit message) terugstuurt. Daarna stuurt de user interface node de omgevormde temperatuur naar een externe GUI.

*Naar: Open Source Robotics Foundation, Master - ROS Wiki*

### 11.3 MoveIt

MoveIt! is een redelijk geavanceerd pakket in ROS dat een aantal motion planning modules (inverse kinematicsolver, motion planner zelf, etc.) met elkaar laat samenwerken. Het hoofddoel van MoveIt! is het aanbieden van een motion planner waarmee je paden kan uitplannen op een gesimuleerde robotarm, een aantal veiligheidschecks kan uitvoeren (fysieke joint limits, snelheid, acceleratie, collisie met zichzelf, enzovoort...) en dat plan vervolgens kan uitvoeren op de echte robotarm.

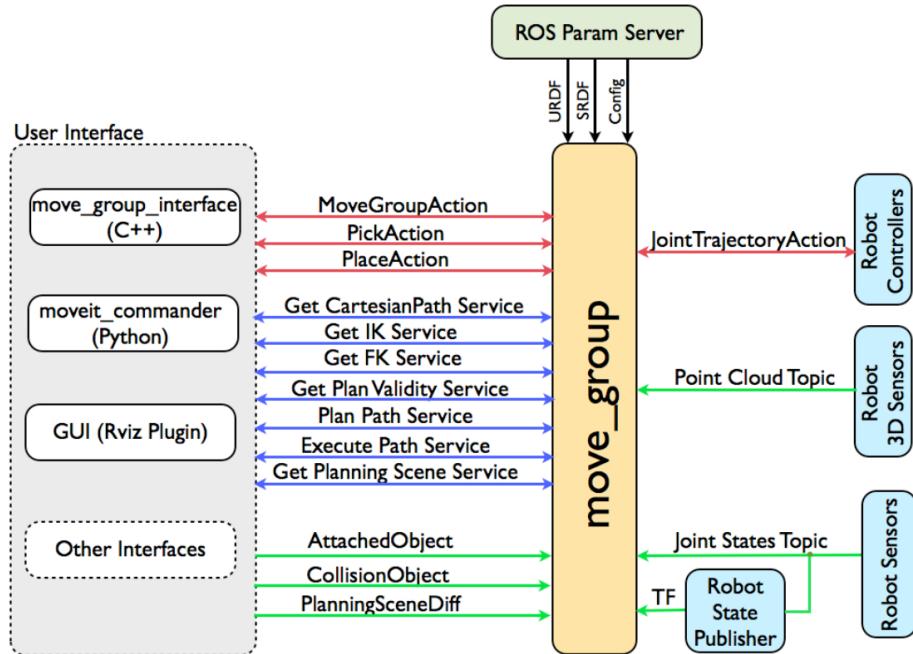


Fig. 80: Theoretische structuur van MoveIt!.

Bron: PickNik Consulting, *Concepts*

In de volgende subhoofdstukken zal ik de concepten van MoveIt! kort bespreken.

#### 11.3.1 move\_group

De hoofdnode van MoveIt! is de move\_group node. Deze gedraagt zich als een manager en verbindt de verschillende modules met elkaar.

### 11.3.2 User interfaces

MoveIt! biedt een aantal user interfaces aan. De eerste is de move\_group\_interface. Deze is enkel bruikbaar door andere nodes die in c++ geschreven zijn (roscpp) en is een speciale interface voor het controleren van end-effectoren. Zo zijn er specifieke messages waar je dingen zoals 'pre grasp pose' of 'post grasp pose' kan instellen. Momenteel gebruik ik deze interface niet om de tool (een grijper bijvoorbeeld) te besturen, maar dit is wel iets wat ik in de toekomst moet doen.

Verder heb je twee interfaces waarmee je de robotarm zelf kan besturen. Eén ervan is Rviz. Rviz is een visualisatietool waarmee je niet alleen de gesimuleerde robotarm kan bekijken, maar waarmee je ook dankzij MoveIt! geplande paden kan inspecteren en met behulp van speciale markers een 'goal pose' kan instellen. De andere (moveit\_commander) kan qua functionaliteit hetzelfde als Rviz, maar dan zonder het visualisatiegedeelte.

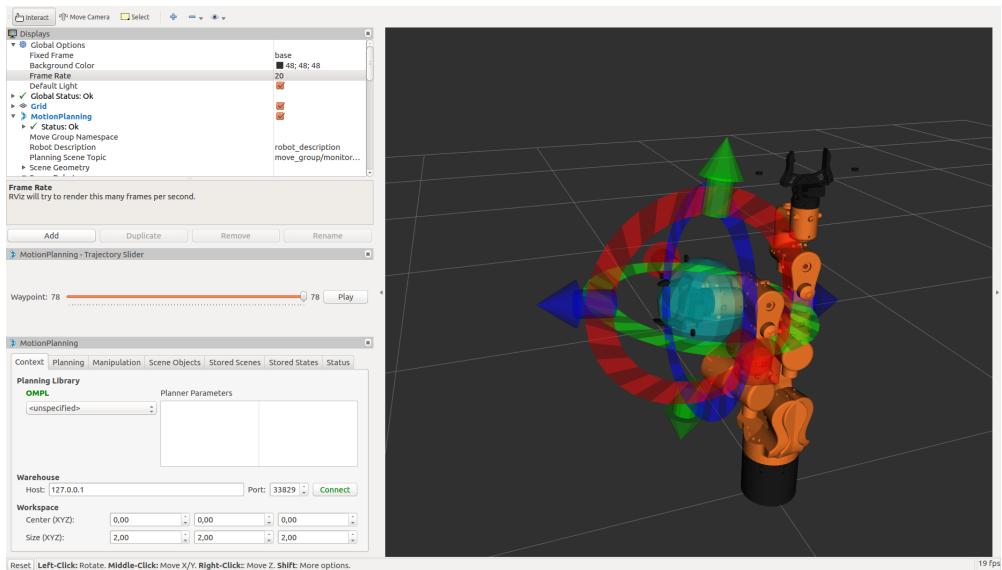


Fig. 81: De interface die Rviz aanbiedt.

Dan zijn er nog een paar andere interfaces waarmee je objecten in de gesimuleerde wereld kan plaatsen om toepassingen zo goed mogelijk te simuleren. Niets of niemand houdt je ook tegen om zelf interfaces te ontwikkelen.

### 11.3.3 Robotic interfaces

Met robotic interfaces bedoel ik de interfaces die normaal gezien niet zichtbaar zijn voor de eindgebruiker. De belangrijkste daarvan zijn de robot controllers. Deze stukjes software moeten een JointTrajectoryAction service aanbieden, wat eigenlijk gewoon wil zeggen dat die service overweg moet kunnen met een pad dat de echte robotarm zou moeten volgen. In mijn geval is het een node dat gebaseerd is op het ros\_control pakket.

Nog een belangrijke interface is de 'robot sensors' interface. Deze interface beheert de 'sensoren' van de robotarm. Meestal zijn die 'sensoren' gewoon de feedback van de encoders. Eigenlijk dus de huidige rotatie van elke as van de robotarm. Hiermee kan MoveIt! enerzijds de simulatie updaten, anderzijds kunnen de robot controllers controleren dat de robotarm het pad correct en binnen de opgelegde maximale afwijkingen aan het uitvoeren is.

#### 11.3.4 MoveIt! Setup Assistant

De MoveIt! Setup Assistant is een wizard dat je toelaat om op een eenvoudige manier een robotbeschrijving om te vormen naar een beschrijving waar MoveIt! een simulatie op kan doen. Zo berekent deze wizard bijvoorbeeld welke robotonderdelen nooit in botsing kunnen komen en waar het collision detection algoritme dus geen berekeningen moet doen. Ook kan je hier verschillende assen en linken groeperen, wat handig kan zijn als je een platform met twee robotarmen wilt besturen. Je kan hier ook bijvoorbeeld de inverse kinematicasolver selecteren. De resulterende beschrijving zal dan in een nieuw pakket geplaatst worden.

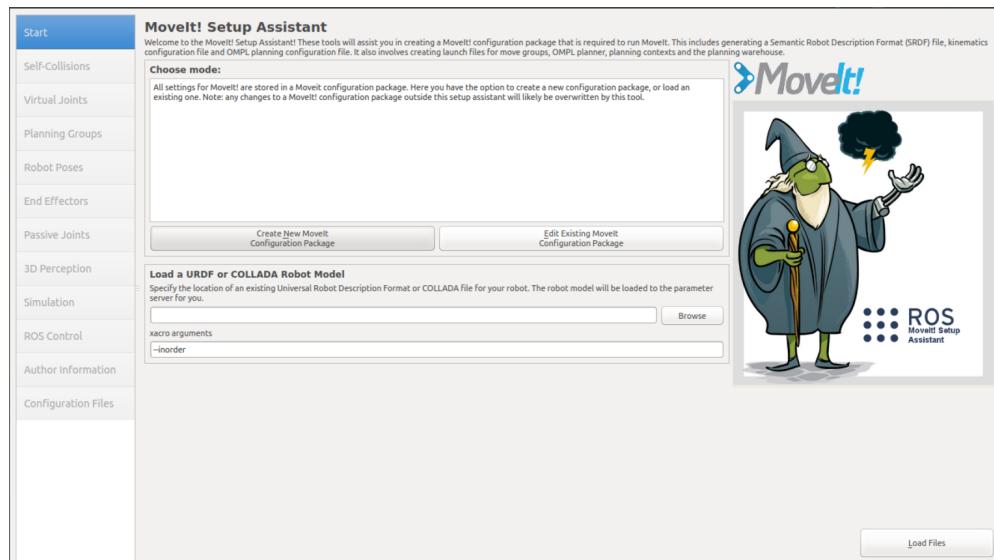


Fig. 82: De MoveIt! Setup Assistant

Naar: PickNik Consulting, *Concepts*

#### 11.4 Trajectory generator

Ik heb ook zelf een toepassing geschreven op de MoveIt software met de move\_group interface. Met deze 'trajectory generator' kan ik met behulp van Rviz een pad plannen en vervolgens laten opslaan in een csv-bestand. Daarna kan ik dat csv-bestand uitvoeren.

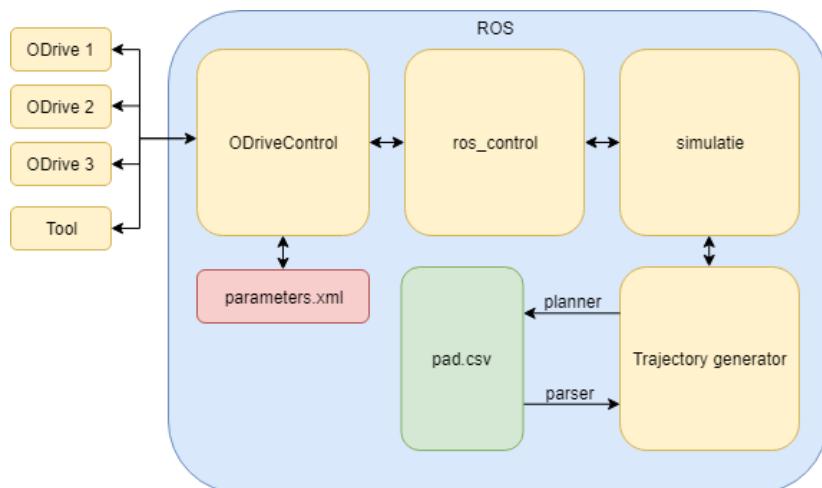


Fig. 83: De plaats van de trajectory generator in de ROS-omgeving.

#### 11.4.1 Het plannen van een pad

In de terminal van de trajectory generator kan je het commando 'plan file.csv' ingeven om een nieuw pad te plannen. Met file.csv bedoel ik het csv-bestand waar het pad in opgeslagen moet worden. De naam van het bestand maakt niet uit, zolang het een csv-bestand is.

Daarna moet je in Rviz (of via andere interfaces, maakt niet veel uit) de 'goal pose' van de gesimuleerde robotarm instellen en het pad plannen. In onderstaande afbeelding wordt de interface Rviz getoond tijdens het plannen. Je ziet drie robotarmen. De robotarm die naar links wijst stelt de gewenste positie voor (goal pose), de robotarm die volledig verticaal staat stelt de huidige positie voor (positie van de echte robotarm) en de doorzichtige robotarm wordt gebruikt om het pad tussen de twee andere 'robotarmen' te visualiseren.

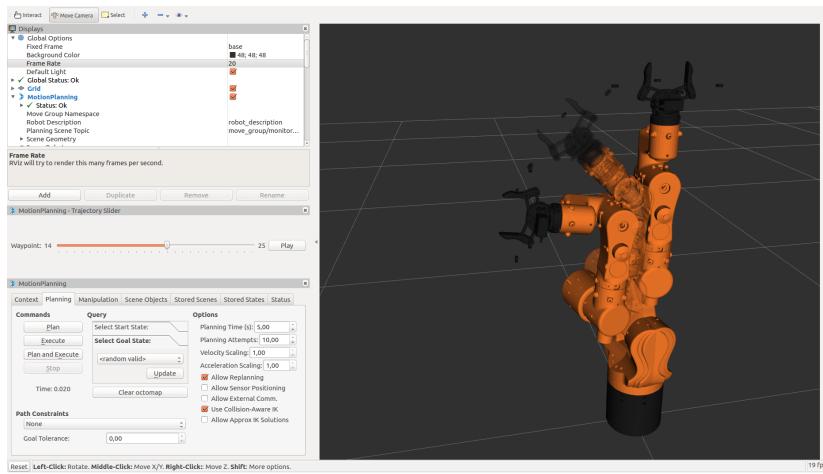


Fig. 84: De Rviz interface waar je een pad kan plannen.

Vervolgens kan je in de terminal van de trajectory generator het commando 'add' en een aantal parameters ingeven om de huidige geplande end-effectortoestand op te slaan. De eerste parameter is het type beweging. Er zijn drie mogelijke bewegingstypes:

- **Bewegingstype p:** dit bewegingstype zorgt er voor dat de robotarm zo snel mogelijk naar de nieuwe positie gaat.
- **Bewegingstype c:** dit bewegingstype zorgt ervoor dat de robotarm in een rechte lijn naar het nieuwe punt gaat. Dat is bijvoorbeeld belangrijk als je een voorwerp wil opnemen.
- **Bewegingstype t:** dit bewegingstype wordt gebruikt als je enkel de tool wilt laten bewegen. De joint state vector dat bij de entry hoort wordt dus tijdens het parsen genegeerd.

De tweede parameter is het activeringspercentage van de tool. Bij een grijper is een parameterwaarde van 100 dus het commando om de grijper volledig te sluiten en een parameterwaarde van 0 het commando om de grijper volledig te openen. De derde parameter is een schalingsfactor van 0 tot 1 voor de uitvoeringssnelheid. Een schalingsfactor van 1 zorgt ervoor dat de beweging op maximale snelheid zal uitgevoerd worden terwijl een schalingsfactor van 0.5 dezelfde beweging zal uitvoeren, maar dan half zo snel. De vierde en laatste parameter is de tijd in seconden dat de parser na het

```

lowiek@lowiek-Ubuntu: ~/Desktop
-----
gip_robot trajectory parser/saver
-----
Choose an action

Action> plan test.csv
('Plan to file ', 'test.csv')
Succes! Possible actions:
    [add a b c d]: add a waypoint where a is the type, b is the toolpercentage, c is the speed scale and d is the time to wait after execution
    [quit ]: save the file and quit the planner
planningAction> add p 0 1 0
['p', '-0.35649198541', '-1.62308459499', '-0.276975732361', '-1.51468819494',
', '-2.09631403743', '0.0856734075848', '0', '1', '0']
planningAction> add p 0 1 0
['p', '-0.726473463911', '-3.06413721827', '-0.429431730071', '-3.14014949721
', '-2.78899407439', '2.31408704414', '0', '1', '0']
planningAction> add p 0 1 0
['p', '-7.08917551674e-05', '-1.57007139062', '-1.57083077148', '-4.291768141
57e-05', '-3.1414357514', '-1.5526869274', '0', '1', '0']
planningAction> quit
Quitting...
Action>

```

Fig. 85: De trajectory planner na het plannen van een traject met test.csv

	A	B	C	D	E	F	G	H	I	J	K
1	type	T1	T2	T3	T4	T5	T6	Toolpercentage	speed	waittime	info
2	p	-0.3564919854	-1.623084595	-0.2769757324	-1.5146881949	-2.0963140374	0.0856734076	0	1	0	
3	p	-0.7264734639	-3.0641372183	-0.4294317301	-3.1401494972	-2.7889940744	2.3140870441	0	1	0	
4	p	-7.08917551674E-05	-1.5700713906	-1.5708307715	-4.29176814157E-05	-3.1414357514	-1.5526869274	0	1	0	

Fig. 86: Het gegenereerde csv-bestand

uitvoeren van een beweging moet wachten om het volgende punt uit te voeren.

Als je het commando 'quit' ingeefdt zal het csv-bestand afgesloten worden. Een csv-bestand kan er uitzien zoals de foto hierboven. Elke lijn stelt een nieuwe 'goal pose' voor. Zoals je ziet slaat de trajectory generator niet de 'goal pose' zelf op (positieverctor en quaternion), maar wel de joint state vector (de thetawaarden) die overeenkomt met de 'goal pose'. Eerst werkte ik wel met de positieverctor en de quaternion van de 'goal pose', maar daar ben ik vlug van afgestapt. Daar zat je namelijk met het probleem dat je tijdens het parsen de inverse kinematicasolver nodig hebt. De resulterende joint state vector was daarom niet altijd dezelfde. De robotarm gedroeg zich dus niet op een voorspelbare manier, wat niet goed is op vlak van veiligheid en al zeker niet de bedoeling is van deze trajectory generator. Door onmiddellijk de joint state vector op te slaan heb ik tijdens het parsen de inverse kinematicasolver niet meer nodig en wordt de robotarm wel voorspelbaar.

#### 11.4.2 Het parsen van een pad

De trajectory generator heeft nog een functie, namelijk het commando 'parse file.csv'. Dit commando zal het bestand file.csv (of eender welk bestand je daar zet, zolang het maar een csv-bestand is) zoeken en vervolgens stap na stap uitvoeren.

Momenteel kan de trajectory generator ook bestanden parsen door een commando naar de '/gip\_robot/trajectory\_generator/command' topic te sturen. Deze topic verwacht een message van het type 'std\_msgs/String' en is tijdelijk ingevoerd zodat het voor een toepassing mogelijk is om voorafbepaalde paden automatisch uit te voeren. In de toekomst zal dit systeem vervangen worden door een service.

```

lowiek@lowiek-Ubuntu: ~/Desktop
['p', '-0.726473463911', '-3.06413721827', '-0.429431730071', '-3.14014949721
', '-2.78899407439', '2.31408704414', '0', '1', '0']
planningAction> add p 0 1 0
['p', '-7.08917551674e-05', '-1.57007139062', '-1.57083077148', '-4.291768141
57e-05', '-3.1414357514', '-1.5526869274', '0', '1', '0']
planningAction> quit
Quitting...
Action> parse test.csv
('Parsing file', 'test.csv')
Sucses! Press enter to execute

('New point to execute:', ['p', '-0.35649198541', '-1.62308459499', '-0.27697
5732361', '-1.51468819494', '-2.09631403743', '0.0856734075848', '0', '1', '0
'])
Robot finished executing... Waiting on point timeout to finish...
('New point to execute:', ['p', '-0.726473463911', '-3.06413721827', '-0.4294
31730071', '-3.14014949721', '-2.78899407439', '2.31408704414', '0', '1', '0
'])
Robot finished executing... Waiting on point timeout to finish...
('New point to execute:', ['p', '-7.08917551674e-05', '-1.57007139062', '-1.5
7083077148', '-4.29176814157e-05', '-3.1414357514', '-1.5526869274', '0', '1'
], '0')
Robot finished executing... Waiting on point timeout to finish...
Action>

```

Fig. 87: De trajectory generator na het parsen van een csv-bestand

## 12 ODriveControl

### 12.1 Inleiding

ODriveControl is een zelfgeschreven programma dat als functie heeft om alle ODrives in te stellen en met elkaar te verbinden, een aantal veiligheidssystemen te implementeren, een GUI te voorzien voor het monitoren van belangrijke waarden zoals motorstroom en een interface te voorzien voor de samenwerking met ROS.

ODriveControl heb ik geschreven in python en daarna in een ROS pakket gestoken als een uitvoerbaar script. Op deze manier hou ik me aan de conventies van ROS en is het makkelijk overzetbaar naar andere ROS-omgevingen.

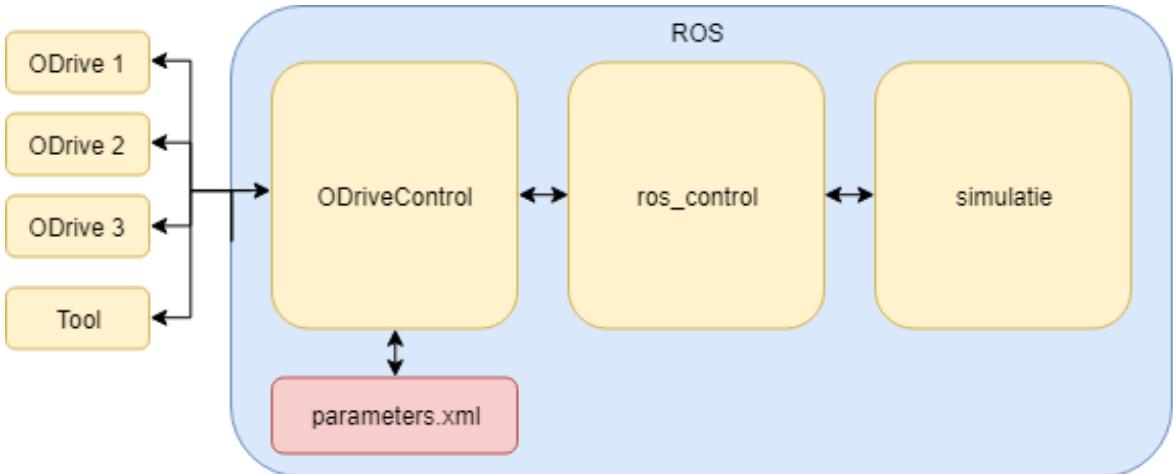


Fig. 88: Overzicht van de locatie van ODriveControl

Zoals je ziet op de bovenstaande afbeelding is ODriveControl verantwoordelijk voor de communicatie met de ODrives en de Arduino (of andere hardware) dat de end-effector (gripper, pen, camera, enzovoort...) bestuurt. Het is onderdeel van de ROS-omgeving en maakt gebruik van de rospy bibliotheek om te communiceren met andere nodes via topics. ODriveControl communiceert voornamelijk met ROS via een aparte ros\_control node. De ros\_control node is verantwoordelijk voor de communicatie tussen de simulatiesoftware (MoveIt!) en ODriveControl. Het zorgt voor de

JointTrajectoryAction service dat noodzakelijk is bij het gebruiken van MoveIt! en zal de juiste interfaces van ODriveControl aanspreken om de robotarm te besturen. De ros\_control node is geschreven bovenop het ros\_control\_boilerplate pakket van PickNik Consulting en de source code is beschikbaar in GitHub via de volgende link: [https://github.com/PickNikRobotics/ros\\_control\\_boilerplate](https://github.com/PickNikRobotics/ros_control_boilerplate)

Ook zal ODriveControl een extern .xml bestand aanspreken dat alle parameters van de ODrives vasthoudt. Daar worden bijvoorbeeld de stroomlimieten bijgehouden, de parameters van de P- en PI-loop, de huidige stand van de as voorgesteld als de theta-waarde van de Denavit-Hartenbergconventie, ... Dit heeft het grote voordeel dat ik elke ODrive kan aansluiten en dat alle instellingen automatisch goedgezet worden. Zo hoef je niet zelf de ODrives te configureren.

Strikt genomen zouden de constante parameters in een .yaml bestand moeten zitten en bij de start ingeladen worden in de parameter server, maar omdat ik met parameters werk die kunnen veranderen (theta-waarde) en omdat python goede ondersteuning biedt voor xml-bestanden, leek het me een beter idee om met een extern xml-bestand te werken.

In de volgende hoofdstukken zal ik meer uitleg geven over de werking van ODriveControl.

## 12.2 Interfaces

In dit onderdeel zal ik het hebben over de interfaces die deze software aanbiedt. Meerbepaald over de communicatievormen die ondersteund worden en het te verwachten gedrag.

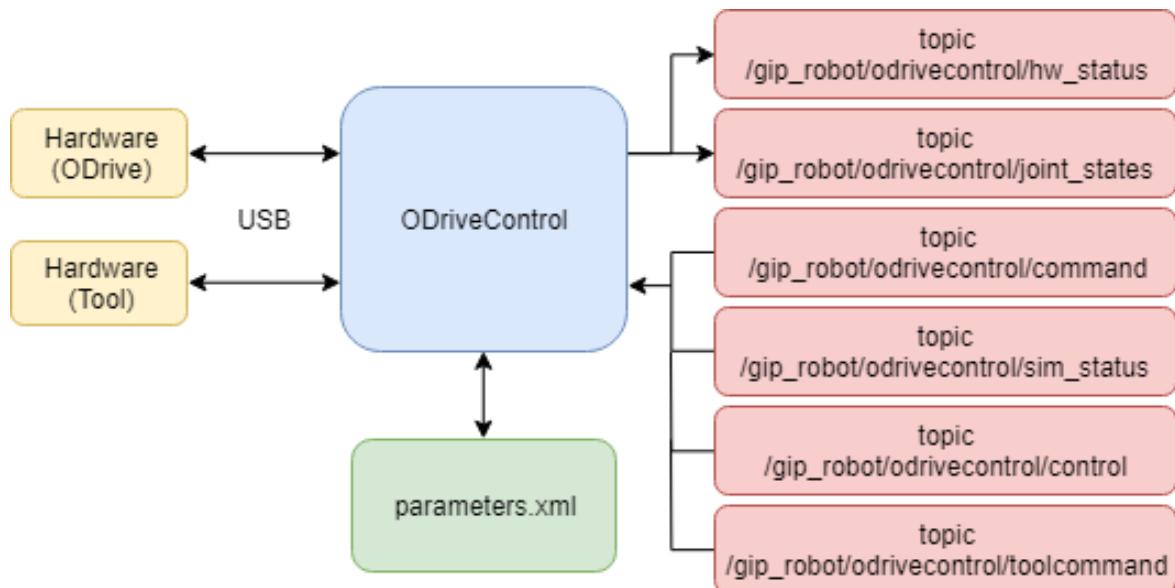


Fig. 89: Overzicht van de interfaces van ODriveControl

### 12.2.1 ODriveControl - hardware

De communicatie tussen de hardware en ODriveControl verloopt via het USB-protocol. ODriveControl maakt de communicatie met de ODrives met behulp van de bibliotheek 'odrive'. Het leggen van de verbinding gebeurt volautomatisch bij het opstarten op

basis van de serienummers van de ODrives. Elke ODrive heeft een uniek serienummer. Door met serienummers te werken kan ik verzekeren dat alle assen van de ODrives juist geïnterpreteerd worden door ODriveControl. Anders zou het kunnen gebeuren dat ODriveControl as 2 een commando geeft, maar dat eigenlijk de vierde as begint te draaien, met alle gevolgen van dien.

Er worden twee commando's naar een ODrive gestuurd als ODriveControl die as wil laten bewegen. Het eerste commando zorgt ervoor dat het systeem op scherp staat en dat de closed-loop position control geactiveerd wordt. Het tweede commando stuurt dan een positie naar de trajectory planner. Zie hoofdstuk 10.4 voor meer informatie over die trajectory planner.

ODriveControl zal na het instellen van de ODrives zoeken naar een ander USB-apparaat dat geschikt is voor het besturen van een tool. Als er geen apparaat gevonden is, dan zal ODriveControl er van uit gaan dat de tool niet aangesloten is. Als er wel een geschikt apparaat gevonden wordt, dan zal ODriveControl daar mee verbinden en dat gebruiken om de tool te besturen.

Momenteel moet de toolhardware (een Arduino bijvoorbeeld) qua interface in staat zijn om de volgende dingen te doen:

- Bij het ontvangen van het commando **p X;** moet de tool bestuurd worden. Met X bedoel ik een percentage dat voorstelt hoeveel de tool geactiveerd is. Voor een grijper zou een waarde van 100% (commando **p 100;**) betekenen dat de grijper volledig gesloten moet worden. Een waarde van 0% betekent dan dat de grijper volledig open moet gaan.
- Bij het ontvangen van het commando **s;** moet de toolhardware de huidige status van de tool terugsturen in de vorm van een percentage. Het antwoord **70;** zou betekenen dat de tool voor 70% geactiveerd is. Een grijper zou dus voor 70% gesloten zijn.

In de toekomst ben ik van plan om deze interface veel algemener te maken en dan ook beter te integreren met de move\_group\_interface van MoveIt. Maar dat was momenteel geen prioriteit.

### 12.2.2 Communicatie ODriveControl - ROS

Er zijn momenteel zes topics waar ODriveControl mee werkt.

**hw\_status** De topic '/gip\_robot/odrivecontrol/hw\_status' is een topic dat de huidige status laat weten van ODriveControl. Het verwacht een message van het type 'std\_msgs/String'. Als er 'RUNNING' of 'IDLE' gepublished wordt, dan weten de andere nodes dat ODriveControl succesvol gestart is en momenteel goed aan het werken is. Vooral de robot controllers maken gebruik van deze topic voor het onderhouden van de JointTrajectoryAction service.

Als er zich een fout voordoet in ODriveControl, dan zal deze status overschakelen naar 'ERROR'. De andere nodes kunnen dan hiermee rekening houden.

**sim\_status** Soortgelijk is de topic '/gip\_robot/odrivecontrol/sim\_status' een topic dat de status van de simulatie laat weten aan ODriveControl. Het verwacht een message van het type 'std\_msgs/String'. Als er 'RUNNING' of 'IDLE' gepublished wordt, dan weet ODriveControl dat de simulatie correct werkt. Als de simulatie vastloopt of een fatale fout tegenkomt, dan is er een mogelijkheid om 'ERROR' te publishen naar deze topic en dan zal ODriveControl de ODrives in veiligheid brengen. Momenteel is dit geïmplementeerd in ODriveControl, maar nog niet in mijn andere ROS-software.

**joint\_states** De topic '/gip\_robot/odrivecontrol/joint\_states' is een topic dat gebruikt wordt om de huidige 'joint states' naar te publishen. Het verwacht een message van het type 'sensor\_msgs/JointState'. Met een joint state wordt de hoek van de bijhorende as bedoeld, beschreven volgens de Denavit-Hartenbergconventie. ODriveControl zal bij het populeren van deze topic die thetawaarden (de joint states) uit parameters.xml halen. De waarden in dat bestand worden op hun beurt geüpdatet op basis van de uitlezing van de encoders.

De simulatie zal deze joint states overnemen in de simulatie en zo ook controleren dat de robotarm wel effectief het geplande pad aan het volgen is.

**command** De topic '/gip\_robot/odrivecontrol/command' kan door andere nodes gebruikt worden om commando's te sturen naar ODriveControl, met als bedoeling om de robotarm fysiek te laten bewegen. De topic verwacht ook een message van het type 'sensor\_msgs/JointState'. Alleen zijn de meegestuurde thetawaarden nu niet de huidige thetawaarden van de robot, maar wel de gewenste thetawaarden. Nadat de veiligheidssystemen van ODriveControl deze gewenste rotatie goedkeuren, zal ODriveControl een aantal conversies toepassen en dan via het USB-protocol de correcte ODrive commanderen om een as naar de gewenste rotatie te laten bewegen.

**toolcommand** De topic '/gip\_robot/odrivecontrol/toolcommand' kan door andere nodes gebruikt worden om commando's te sturen naar ODriveControl, met als bedoeling om de tool te laten bewegen. De topic verwacht een message van het type 'std\_msgs/String'. Voorbeelden van commando's zijn **p 100;** of **p 60;**

Momenteel is er nog geen topic dat de simulatie de huidige tooltoestand laat weten, zoals de '/joint\_states' topic doet voor de assen van de robotarm. In de toekomst zou dit wel moeten veranderen.

**control** De /gip\_robot/odrivecontrol/control topic is een debug interface voor de gebruiker. Hier kan je met een std\_msgs/String message commando's sturen naar ODriveControl. Momenteel zijn de volgende commando's geïmplementeerd:

- **calMotor <as>:** Deze functie zal een motorcalibratie uitvoeren op as <as>.
- **calEncoder <as>:** Deze functie zal een encodercalibratie uitvoeren op as <as>.
- **move <as> <newtheta>:** Hiermee kan je as <as> naar een nieuwe theta <newtheta> laten bewegen. Het gedraagt zich juist hetzelfde als het publishen van een nieuwe JointState message naar de command topic, met het enige verschil dat je een as moet kiezen en dat <newtheta> in graden verwacht wordt in plaats van in radialen.
- **resetAxis <as>:** Hiermee kan je de foutmeldingen van as <as> resetten.

- **resetAll:** Hiermee reset je de foutmeldingen van alle assen.
- **checkErrors:** Deze functie zal alle assen controleren op foutmeldingen en ze uitprinten indien ze aanwezig zijn.
- **stop:** Door dit uit te voeren zal ODriveController de afsluitprocedure starten.

### 12.2.3 Communicatie ODriveControl - parameters.xml

De communicatie tussen ODriveControl en parameters.xml gebeurt via de ingebouwde python bibliotheek 'xml.dom.minidom'. Het is een minimalistische implementatie van een parser die xml-bestanden kan parsen naar een DOM (Document Object Model interface). Deze DOM laat je toe om met behulp van tags en elementen de juiste data uit het xml-bestand te halen. Hieronder vind je de algemene richtlijnen waaraan parameters.xml moet voldoen.

```
<?xml version="1.0" ?><xml>
<root>
  <A1>
    <encoderParameters>
      <cpr>2400</cpr>
    </encoderParameters>
    <motorParameters>
      <accel_limit>500000</accel_limit>
      <decel_limit>500000</decel_limit>
      <vel_limit>200000</vel_limit>
      <vel_gain>0.0005</vel_gain>
      <pos_gain>100</pos_gain>
      <vel_integrator_gain>0.0</vel_integrator_gain>
      <current_lim>25</current_lim>
      <calib_current>20</calib_current>
      <pole_pairs>7</pole_pairs>
    </motorParameters>
    <simulationParamaters>
      <theta>0.08995002167565483</theta>
      <dir>1</dir>
      <mintheta>-180</mintheta>
      <maxtheta>180</maxtheta>
    </simulationParamaters>
  </A1>
  <A2>
    ...
  </A2>
  ...
</root>
</xml>
```

**<cpr>** De Counts Per Revolution waarde is de hoeveelheid 'counts' de encoder zou gestuurd hebben als we de bijhorende motor (niet de as, effectief de motor zelf) 1 rotatie laten uitvoeren. Zie hoofdstuk 1.2 voor meer uitleg over de encoder.

**<accel\_limit>** De acceleratielimiet van de desbetreffende motor. Uitgedrukt in counts/ $s^2$ .

**<decel\_limit>** De deacceleratielimiet van de desbetreffende motor. Uitgedrukt in counts/ $s^2$ .

**<vel\_limit>** De snelheidslimiet van de desbetreffende motor. Uitgedrukt in counts/s.

**<vel\_gain>** De proportionele parameter van de PI-controller voor snelheidscontrole van die as. Zie hoofdstuk 10.3.1 voor meer informatie.

**<pos\_gain>** De proportionele parameter van de P-controller voor positiecontrole van die as. Zie hoofdstuk 10.3.1 voor meer informatie.

**<vel.integrator\_gain>** De integrerende parameter van de PI-controller voor snelheidscontrole van die as. Zie hoofdstuk 10.3.1 voor meer informatie.

**<current\_limit>** De maximale stroom dat de motor van de desbetreffende as mag opnemen.

**<calib\_current>** De stroomsterkte die gebruikt wordt bij het uitvoeren van een motor- en/of encodercalibratie.

**<pole\_pairs>** Het aantal poolparen van de desbetreffende motor. Zie hoofdstuk 1.3 voor meer informatie over poolparen.

**<theta>** De huidige rotatie van de desbetreffende as volgens de Denavit-Hartenbergconventie. Uitgedrukt in graden.

**<mintheta>** De minimaal toegelaten thetawaarde.

**<maxtheta>** De maximaal toegelaten thetawaarde.

## 12.3 Calibratiemenu

Nadat alle ODrives en tools verbonden zijn en dat parameters.xml uitgelezen is, zal ODriveControl de user een calibratiemenu tonen. ODriveControl zal eerst nagaan welke assen al gecalibreerd zijn en vervolgens de user drie opties geven. Optie '0' is de automatische mode. Hier zal ODriveControl de assen calibreren die nog niet gecalibreerd zijn. Optie '1' forceert een calibratie bij elke as, ook bij de assen die al gecalibreerd zijn en '2' slaat het calibratieproces gewoon over. De laatste optie is enkel voor het debuggen of als je handmatig een as wil bewegen.

Fig. 90: Aan de linkerkant zie je het calibratiemenu, rechts zie je calibratieproces in werking

## 12.4 Plotters

Na het volledig opstarten van ODriveControl zal de software een aantal plotters lanceren. Deze plotters houden de positie en de snelheid bij van elke as, alsook de stroom die de motoren trekken en de vermogens van elke ODrive. ODriveControl voert ook een aantal veiligheidscontroles uit op deze waarden. Stel bijvoorbeeld dat een ODrive bijna 400W trekt (de limiet van mijn power supplies), dan zal ODriveControl die ODrive in veiligheid brengen.

Verder zal ODriveControl ook continu controleren dat de ODrive een beweging aan het uitvoeren is of niet. Als een ODrive meer dan een halve seconde geen beweging aan het uitvoeren is, dan zal ODriveControl die ODrive in stand-by zetten. Een nadeel van de closed-loop positiecontrole is namelijk dat het kan gebeuren dat er na een beweging toch nog een redelijk grote stroom door de motoren blijft gaan. Dit zorgt voor zeer veel warmteontwikkeling. De veiligheid dat de ODrives na een tijdje in stand-by zet zorgt er dus voor dat die stroom wegvalt en dat de warmteontwikkeling redelijk blijft.

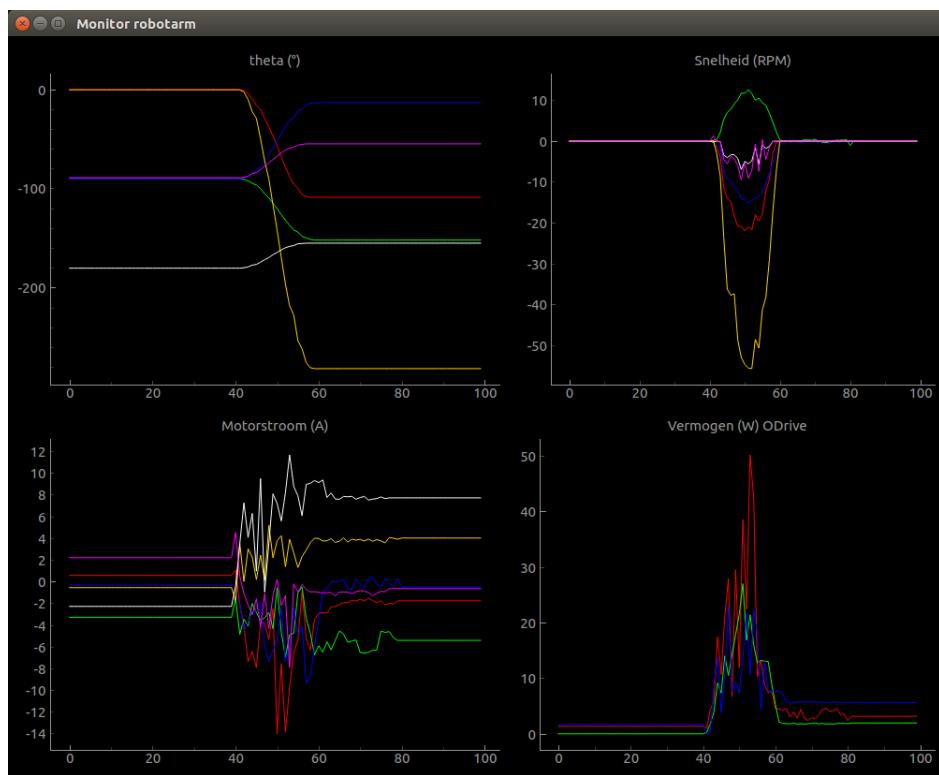


Fig. 91: De plotters van ODriveControl.

Bij de motorstromen (linksonder) zie je eerst veel variatie (tijdens de beweging), daarna zeer weinig variatie maar toch wel hoge stromen (juist na de beweging) en tenslotte een constante, rechte lijn als gevolg van de veiligheid in ODriveControl (0.5s geen beweging). Je zal misschien denken dat er in het laatste gedeelte nog steeds een stroom door de motoren gaat, maar dit is niet waar. Als een ODrive in stand-by gezet wordt, zal die namelijk de laatst gemeten stroom naar ODriveControl sturen in plaats van 0A. Daarom is een stroom van 0A in de plotter te herkennen als een rechte horizontale lijn.

# 13 Praktische uitwerking van de software

## 13.1 Inleiding

In dit hoofdstukje zal ik uitleggen hoe ik de verschillende softwaregedeeltes en ROS-paketten samengebracht heb om zo de eerste iteratie van het softwaregeheel samen te stellen. Om de robotarm te doen werken heb ik een aantal paketten samengesteld:

- **gip\_robot:** het gip\_robot pakket bevat de beschrijving van de robotarm. Onder de map 'urdf' heb ik het urdf-bestand geplaatst van de robotarm (de kinematische beschrijving met andere woorden) en onder de map 'meshes' staan de 3D-modellen die enerzijds gebruikt worden om een visueel beeld te genereren van de robotarm tijdens de simulatie en anderzijds gebruikt worden voor het collision detection algoritme van MoveIt!.
- **gip\_robot\_moveit\_config:** dit pakket is het resulterende pakket na het voltooien van de MoveIt! Setup Assistant. Het bevat ook bestanden zoals joint\_limits.yaml dat de fysieke limieten zoals maximale rotatie, snelheid en acceleratie van elke as bijhoudt.
- **ros\_control\_boilerplate:** dit is een pakket dat geschreven is door PicNick Consulting en waarvan de source code te vinden is op GitHub. Dit pakket laat je toe om een subpakket te schrijven dat op een eenvoudige manier een ros\_control node start om de JointTrajectoryAction service van de robotarm te onderhouden. Het subpakket dat ik voor mijn robotarm geschreven heb heet gip\_robot\_control.
- **odrivecontrol:** dit pakket bevat alle scripts en launchfiles voor ODriveControl te starten.
- **gip\_robot\_master:** dit pakket beheert de andere paketten door launchfiles te voorzien dat de nodes van de andere paketten op een juiste manier laat starten. Ook is dit het pakket dat je toelaat om de trajectory generator te starten.

## 13.2 Het opstarten van de software

The screenshot shows two terminal windows. The left terminal displays the output of the command `$ roscore http://lowiek:11311/`. It includes system warnings about disk usage and ROS master start information. The right terminal shows the command `$ rostopic pub /gip_robot/odrivecontrol/control std_msgs/String "stop" --once` being entered.

```

roscore http://lowiek:11311/
[WARNING] disk usage in log directory [/home/lowiek/.ros/log] is over 1GB.
It's recommended that you use the 'rosclean' command.

started roslaunch server http://lowiek:41411/
ros_comm version 1.12.14

SUMMARY
========
PARAMETERS
  * /roslistro: kinetic
  * /rosverston: 1.12.14

NODES
auto-starting new master
process[master]: started with pid [26085]
ROS_MASTER_URI=http://lowiek:11311/
setting /run_id to afdb268-7e55-11e9-a7a9-e4a7a09dbdcc
process[rosout-1]: started with pid [26098]
started core service [/rosout]

/home/lowiek/ws_moveit/src/gip_robot_master/launch/gip_robot_hw.launch http://lowiek:11311
[INFO] [1558724574.921619448]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.925017435]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.928343218]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.931661664]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.934953993]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.938288518]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.941641898]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.944967139]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.948335959]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.951673145]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5
[INFO] [1558724574.955151463]: -0.000355-1.570326-1.5698920.000073-3.142084-1.5

```

Fig. 92: De actieve terminals na het volledig opstarten van het systeem. De terminal rechtsonder is de 'noodstop' en geeft het stopcommando aan ODriveControl.

Eerst en vooral moet ODriveControl gestart worden. Vooraleer we ODriveControl kunnen starten, moeten we de master node van ROS opstarten door het commando `$ roscore` in een terminal in te geven. Ik zou perfect de master node in dezelfde terminal kunnen opstarten als die van ODriveControl, maar als ODriveControl in dat geval crasht dan zal de master node ook stoppen. Dit heeft dan als gevolg dat de simulatie ook zal crashen. Door de master node in een andere terminal te laten lopen kan ik gewoon ODriveControl opnieuw lanceren na een crash, zonder dat de simulatie er iets van zal merken.

Het ODriveControl.py scripts en alle bijhorende scripts zijn samengevoegd in het pakket 'odrivecontrol'. Voor ODriveControl te lanceren moet ik dus het volgende typen in een nieuwe terminal: `$ rosrun odrivecontrol ODriveControl`.

Na het calibratieproces zal ODriveControl wachten op de JointTrajectoryAction service voorzien door de ros\_control node. Daarom moeten we op dat moment de ros\_control node opstarten in een nieuwe terminal. Het commando om dit te doen is

`$ rosrun gip_robot_master gip_robot_hw.launch` .

De JointTrajectoryAction service zal geen commando's sturen naar ODriveControl vooraleer ODriveControl zelf een eerste call gemaakt heeft. Die eerste call bevat dan de initiële joint states (thetawaarden) die van parameters.xml uitgelezen zijn.

Vervolgens zal ODriveControl de plotter opstarten en zal het gip\_robot\_master pakket automatisch de MoveIt! node en een node voor de visualisatietool Rviz opstarten. Daarna is het systeem actief en kan je elke toepassing dat je wilt starten. De commando's voor de trajectory generator zijn `$ rosrun gip_robot_master trajectory_generator_hw.launch` of `$ rosrun gip_robot_master trajectory_generator_sw.launch` .

Het eerste commando is voor als je met de echte robotarm werkt, het tweede commando is voor als je enkel met de simulatie werkt.

Stel dat ik enkel de simulatie wil hebben om bijvoorbeeld een toepassing te schrijven, volstaat het om enkel `$ roslaunch gip_robot_master gip_robot_simulation.launch` in een terminal uit te voeren. Dit lanceert de node voor MoveIt!, de node voor Rviz en een ros\_control node dat doet alsof het een echte robotarm is. ODriveControl is hier niet nodig.

## 14 Toekomstvisie van de software

Het is perfect mogelijk om met de huidige software toepassingen te ontwikkelen. Maar dat neemt het feit niet weg dat de software nog veel ruimte voor verbetering heeft. Niet alles is even goed doordacht, niet alles volgt de conventies van ROS of MoveIt! en het is ook vaak omslachtig om ze te gebruiken. Daarom zal ik nog steeds verderwerken aan de software.

Eerst en vooral zal ik de bestaande scripts stabieler en abstracter maken. Concreet betekent dit dat de scripts beter moeten kunnen omgaan met interne fouten en dat ze gebruiksvriendelijker moeten worden. Daarna zal ik wijzigingen doorvoeren zodanig dat de conventies van ROS en MoveIt! beter gevuld worden.

Dit zal zeker een werk van lange adem zijn, maar na deze wijzigingen zal ik een stevig platform ter beschikking hebben voor het ontwikkelen van toepassingen. Waar ik me zeker verder in wil verdiepen, is het toepassen van kunstmatige intelligentie en computer vision op deze robotarm.

## Besluit

Het was een leuke uitdaging om deze robotarm te realiseren. Het fysieke gedeelte is zeer goed verlopen. Wel had ik wat moeite met de ontwikkeling van de eerste as. De software was uitdagender dan verwacht. Dit is voornamelijk te danken aan de steile leercurve van ROS. Uiteindelijk is het wel goed gelukt om al een stabiel softwarepakket te ontwikkelen waar ook andere mensen mee aan de slag kunnen gaan.

In de toekomst zou ik graag zowel de snelheid als de precisie van mijn robotarm verbeteren. Ook heb ik wegens tijdsgebrek geen echte toepassingen kunnen ontwikkelen. Maar vooraleer ik daar aan begin, ga ik alles documenteren en onder een open-sourcelicentie vrijgeven aan de roboticacommunity. Op basis van hun feedback zal ik een beter idee krijgen hoe ik verder moet gaan met dit project.

Dankzij deze proef heb ik ook veel bijgeleerd. Dingen die van pas kunnen komen bij mijn latere universitaire studies. Je kan het beschouwen als de kers op de taart van mijn zesjarige technische opleiding.

# Figurenlijst

1	De incrementele encoders die ik gebruik . . . . .	8
2	De voorstelling van de pulsen op A en B . . . . .	8
3	Een gedeassembleerde BLDC-motor van het type 'outrunner' . . . . .	9
4	Schematische voorstelling van een stator. (type: inrunner) . . . . .	10
5	De simpelste vorm van commutatie . . . . .	10
6	Trapeziumvormige- of 120°-commutatie . . . . .	11
7	Sinusvormige- of 180°-commutatie . . . . .	11
8	De referentie shift bij sinusvormige commutatie . . . . .	12
9	Overzicht van de karakteristieken, afkomstig van sinusvormige commutatie	12
10	Turnigy Aerodrive SK3 4250 - 350Kv . . . . .	14
11	Propdrive v2 3536 910Kv . . . . .	15
12	De assembly van de eerste iteratie . . . . .	15
13	De eerste as van de eerste iteratie . . . . .	16
14	Doorsnede van de tweede as van de eerste iteratie . . . . .	17
15	Doorsnede van de derde as van de eerste iteratie . . . . .	17
16	Doorsnede van de vierde as van de eerste iteratie . . . . .	18
17	Differentieel bij rechuit rijden . . . . .	18
18	Differentieel wanneer één wiel van de auto geblokkeerd is . . . . .	19
19	As 5 en 6 van de eerste iteratie . . . . .	19
20	De assembly van de tweede iteratie . . . . .	20
21	Een simpel planeetwielmechanisme . . . . .	21
22	Het ontwerp van Gear_Down_For_What . . . . .	22
23	As 1 van de tweede iteratie . . . . .	23
24	Doorsnede van as 1 van de tweede iteratie . . . . .	24
25	Doorsnede van as 2 van de tweede iteratie . . . . .	24
26	Doorsnede van as 3 van de tweede iteratie . . . . .	25
27	Doorsnede van as 4 van de tweede iteratie . . . . .	25
28	Overzicht van as 5 en 6 van de tweede iteratie . . . . .	26
29	De assembly van de derde iteratie . . . . .	27
30	De bodemplaat: links heb je de bovenplaat, rechts heb je de onderplaat	28
31	De waarschuwing op de bodemplaat . . . . .	28
32	De eerste herwerking van de eerste as van de derde iteratie . . . . .	29
33	De tweede herwerking van de eerste as van de derde iteratie . . . . .	29
34	As 2 van de derde iteratie . . . . .	30
35	As 3 van de derde iteratie . . . . .	31
36	Doorsnede van as 3 van de derde iteratie . . . . .	31
37	As 4 van de derde iteratie . . . . .	32
38	Een doorsnede van as 4 van de derde iteratie . . . . .	32
39	As 5 van de derde iteratie . . . . .	33
40	As 6 van de derde iteratie . . . . .	34
41	Het grijze stuk is de interface waarop tools ontwikkeld kunnen worden .	34
42	De BCN3D Moveo grijper . . . . .	35
43	De DH-assenstelsels van joint 6 en de tool (joint 7) . . . . .	35
44	De elektronica behuizing . . . . .	36
45	Links: de elektronica in de behuizing. Rechts: de opeenstapeling van ODrives . . . . .	36
46	Een overzicht van de power supplies . . . . .	37
47	Een close-up van een geïnstalleerde power supply . . . . .	37
48	De IEC C14 plug van de behuizing . . . . .	38

49	Het voorpaneel. De cirkels zijn voor de motoren, de vierkanten zijn voor de encoders . . . . .	38
50	De assenstelsels van de joints van mijn robotarm, bepaald door de DH-conventie . . . . .	40
51	Een voorstelling van eulerhoeken met als conventie $Z'X'Z'$ . . . . .	45
52	De yaw, pitch en roll parameters die gebruikt worden in de luchtvaart .	46
53	De eerste eulerhoek heeft ervoor gezorgd dat de twee volgende eulerhoeken hetzelfde rotatieeffect hebben. . . . .	46
54	Een rotatie van assenstelsel $OX_AYZ_A$ naar assenstelsel $OX_BY_BZ_B$ . .	47
55	Een punt in een imaginair assenstelsel . . . . .	48
56	Een 2D-rotatie met complexe getallen . . . . .	48
57	Het laatste DH-assenstelsel kan gebruikt worden voor de positie en rotatie van de end-effector . . . . .	51
58	Een screenshot van de test in Unreal Engine met de jacobimatrixmethode	54
59	Een voorstelling van een eenvoudige kostfunctie. Als de huidige toestand zich in punt A bevindt, is de kans groot dat het algoritme in het lokaal minima sukkelt in plaats van naar de gewenste toestand te gaan in het globaal minima. . . . .	56
60	Een screenshot van de test in Unreal Engine met de gradient descentmethode.	
	De robotarm zit hier vast in een lokaal minima . . . . .	56
61	Een voorstelling van de rotatieassen en het bijhorende ontkoppelpunt van as 4, 5 en 6 . . . . .	57
62	Het theoretisch schema van een PID-controller . . . . .	58
63	De reactie van $y(t)$ door een wijziging van $r(t)$ met drie verschillende $K_p$ -waarden . . . . .	59
64	De ODrive V3.2 . . . . .	60
65	De bovenkant van een ODrive . . . . .	61
66	Het versimpelde schema van een DRV8301 . . . . .	62
67	De output stage van een ODrive . . . . .	62
68	De onderkant van de ODrive met de rij condensatoren . . . . .	63
69	Een vermogensweerstand . . . . .	64
70	De controlelus van ODrive . . . . .	64
71	De trapeziumvormige trajectory planner van ODrive. De blauwe lijn is een lange beweging, de rode lijn is een korte beweging . . . . .	66
72	Theoretische structuur van communicatie met topics . . . . .	69
73	Van links naar rechts: subscriber.py, publisher.py . . . . .	70
74	Lijst met actieve topics en informatie over '/voorbeeldtopic'	71
75	Voorbeeld van een service dat graden omzet naar radianen . . . . .	72
76	Verzoek van publisher tot het registreren van de '/images' topic . . . .	73
77	Verzoek van node om te subscriben op de '/images' topic . . . . .	73
78	Overgang naar een node-to-node communicatievorm . . . . .	74
79	Overzicht van een ROS omgeving met de parameter server . . . . .	74
80	Theoretische structuur van MoveIt! . . . . .	75
81	De interface die Rviz aanbiedt. . . . .	76
82	De MoveIt! Setup Assistant . . . . .	77
83	De plaats van de trajectory generator in de ROS-omgeving. . . . .	77
84	De Rviz interface waar je een pad kan plannen. . . . .	78
85	De trajectory planner na het plannen van een traject met test.csv . .	79
86	Het gegenereerde csv-bestand . . . . .	79
87	De trajectory generator na het parsen van een csv-bestand . . . . .	80
88	Overzicht van de locatie van ODriveControl . . . . .	80

89	Overzicht van de interfaces van ODriveControl . . . . .	81
90	Aan de linkerkant zie je het calibratiemenu, rechts zie je calibratieproces in werking . . . . .	85
91	De plotters van ODriveControl. . . . .	86
92	De actieve terminals na het volledig opstarten van het systeem. De terminal rechtsonder is de 'noodstop' en geeft het stopcommando aan ODriveControl. . . . .	88
93	De meest algemene vorm van de trapeziumvormige grafiek . . . . .	102
94	De twee gevallen van de trapeziumvormige grafiek. . . . .	103

## Tabellenlijst

1	Specificaties van Turnigy Aerodrive SK3 4250 - 350Kv . . . . .	13
2	Specificaties van Propdrive v2 3536 910Kv . . . . .	14
3	De DH-parameters van mijn robotarm . . . . .	41
4	De parameters van de tweede mogelijkheid . . . . .	100
5	De parameters van de eerste mogelijkheid . . . . .	101

# Bibliografie

- [1] Bruno Vilhena Adorno. *Robot Kinematic Modeling and Control Based on Dual Quaternion Algebra — Part I: Fundamentals*. hal-01478225, 2017.
- [2] Mathias Brandstötter. “Adaptable Serial Manipulators in Modular Design”. PhD thesis. Nov. 2016. URL: [https://www.researchgate.net/figure/Orientation-part-within-axis-angle-representation\\_fig9\\_310196894](https://www.researchgate.net/figure/Orientation-part-within-axis-angle-representation_fig9_310196894). Geraadpleegd op 30 mei 2019.
- [3] Eliott Wertheimer. *How do electric motors in ebikes work?* Feb. 2018. URL: <https://www.furosystems.com/news/ebike-electric-motors-work/>. Geraadpleegd op 27 mei 2019.
- [4] HobbyKing. *PROPDIVE v2 2830 800KV Brushless Outrunner Motor*. URL: [https://hobbyking.com/en\\_us/propdrive-v2-2830-800kv-brushless-outrunner-motor.html?\\_\\_store=en\\_us](https://hobbyking.com/en_us/propdrive-v2-2830-800kv-brushless-outrunner-motor.html?__store=en_us). Geraadpleegd op 27 mei 2019.
- [5] HobbyKing. *Turnigy Aerodrive SK3 - 4250-350KV Brushless Outrunner Motor*. URL: [https://hobbyking.com/en\\_us/turnigy-aerodrive-sk3-4250-350kv-brushless-outrunner-motor.html](https://hobbyking.com/en_us/turnigy-aerodrive-sk3-4250-350kv-brushless-outrunner-motor.html). Geraadpleegd op 27 mei 2019.
- [6] Kaspars Dambis. *Maytech 5055 75kV BLDC outrunner motor rotor and stator*. URL: <https://www.flickr.com/photos/kasparsdambis/28137980911/in/photostream/>. Geraadpleegd op 27 mei 2019.
- [7] madcowswe. *ODrive Documentation*. May 2019. URL: <https://docs.odriverobotics.com/>. Geraadpleegd op 30 mei 2019.
- [8] Mecademic. *How is orientation in space represented with Euler angles?* URL: <https://mecademic.com/resources/Euler-angles/Euler-angles.html>. Geraadpleegd op 20 mei 2019.
- [9] Open Source Robotics Foundation. *Master - ROS Wiki*. Jan. 2018. URL: <http://wiki.ros.org/Master>. Geraadpleegd op 12 mei 2019.
- [10] Open Source Robotics Foundation. *Master - ROS Wiki*. Nov. 2018. URL: <http://wiki.ros.org/Parameter%20Server>. Geraadpleegd op 12 mei 2019.
- [11] Open Source Robotics Foundation. *msg*. Jan. 2019. URL: <http://wiki.ros.org/msg>. Geraadpleegd op 31 mei 2019.
- [12] Open Source Robotics Foundation. *Nodes - ROS Wiki*. Apr. 2018. URL: <http://wiki.ros.org/Nodes>. Geraadpleegd op 12 mei 2019.
- [13] Open Source Robotics Foundation. *Packages - ROS Wiki*. 14 2019. URL: <http://wiki.ros.org/Packages>. Geraadpleegd op 12 mei 2019.
- [14] Open Source Robotics Foundation. *Services - ROS Wiki*. July 2018. URL: <http://wiki.ros.org/Services>. Geraadpleegd op 12 mei 2019.
- [15] Open Source Robotics Foundation. *Topics - ROS Wiki*. Feb. 2019. URL: <http://wiki.ros.org/Topics>. Geraadpleegd op 12 mei 2019.
- [16] PickNik Consulting. *Concepts*. URL: <https://moveit.ros.org/documentation/concepts/>. Geraadpleegd op 30 mei 2019.
- [17] Texas Instruments. *Commutation techniques for three phase brushless DC motors*. Apr. 2017. URL: <https://training.ti.com/commutation-techniques-three-phase-brushless-dc-motors#>. Geraadpleegd op 23 mei 2019.
- [18] Weisstein, Eric W. *Euler Angles*. May 2019. URL: <http://mathworld.wolfram.com/EulerAngles.html>. Geraadpleegd op 30 mei 2019.

- [19] Wikipedia. *Denavit–Hartenberg parameters*. May 2019. URL: [https://en.wikipedia.org/wiki/Denavit%20%93Hartenberg\\_parameters](https://en.wikipedia.org/wiki/Denavit%20%93Hartenberg_parameters). Geraadpleegd op 30 mei 2019.
- [20] Wikipedia. *Differentieel (werktuigbouwkunde)*. Aug. 2018. URL: [https://nl.wikipedia.org/wiki/Differentieel\\_\(werktuigbouwkunde\)](https://nl.wikipedia.org/wiki/Differentieel_(werktuigbouwkunde)). Geraadpleegd op 14 mei 2019.
- [21] Wikipedia. *Euler angles*. Apr. 2019. URL: [https://en.wikipedia.org/wiki/Euler\\_angles](https://en.wikipedia.org/wiki/Euler_angles). Geraadpleegd op 20 mei 2019.
- [22] Wikipedia. *Gimbal lock*. Apr. 2019. URL: [https://en.wikipedia.org/wiki/Gimbal\\_lock](https://en.wikipedia.org/wiki/Gimbal_lock). Geraadpleegd op 20 mei 2019.
- [23] Wikipedia. *Gradient descent*. May 2019. URL: [https://en.wikipedia.org/wiki/Gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent). Geraadpleegd op 30 mei 2019.
- [24] Wikipedia. *Incremental encoder*. May 2019. URL: [https://en.wikipedia.org/wiki/Incremental\\_encoder](https://en.wikipedia.org/wiki/Incremental_encoder). Geraadpleegd op 23 mei 2019.
- [25] Wikipedia. *Jacobi-matrix*. Nov. 2018. URL: <https://nl.wikipedia.org/wiki/Jacobi-matrix>. Geraadpleegd op 30 mei 2019.
- [26] Wikipedia. *PID controller*. May 2019. URL: [https://en.wikipedia.org/wiki/PID\\_controller](https://en.wikipedia.org/wiki/PID_controller). Geraadpleegd op 20 mei 2019.
- [27] Wikipedia. *Planeetwielmechanisme*. Dec. 2018. URL: <https://nl.wikipedia.org/wiki/Planeetwielmechanisme>. Geraadpleegd op 16 mei 2019.
- [28] Wikipedia. *Quaternion*. May 2019. URL: <https://en.wikipedia.org/wiki/Quaternion>. Geraadpleegd op 31 mei 2019.
- [29] Wikipedia. *Quaternions and spatial rotation*. May 2019. URL: [https://en.wikipedia.org/wiki/Quaternions\\_and\\_spatial\\_rotation](https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation). Geraadpleegd op 31 mei 2019.

# Bijlagen

## A. De inverse van de DH-transformatiematrix

We proberen de inverse te berekenen van de volgende DH-transformatiematrix:

$${}^{n-1}T_n = \left[ \begin{array}{ccc|c} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

De inverse van een matrix kunnen we algemeen definieren als:

$$A^{-1} = \frac{1}{|A|} * adj(A) \quad (1)$$

Waar de  $adj(A)$  de geadjugeerde matrix voorstelt van A. Als  $|A| = 0$ , dan krijgen we een deling door nul, met het gevolg dat de inverse onberekenbaar is en dus niet bestaat. Het berekenen van de determinant van  ${}^{n-1}T_n$  door ontwikkeling langs de onderste rij geeft:

$$\begin{aligned} |{}^{n-1}T_n| &= \left| \begin{array}{cccc} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{array} \right| \\ &= \left| \begin{array}{ccc} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n \\ 0 & \sin \alpha_n & \cos \alpha_n \end{array} \right| \\ &= -\sin \alpha_n (-\cos^2 \theta_n \sin \alpha_n - \sin^2 \theta_n \sin \alpha_n) + \cos \alpha_n (\cos^2 \theta_n \cos \alpha_n + \sin^2 \theta_n \cos \alpha_n) \\ &= \sin^2 \alpha_n + \cos^2 \alpha_n = 1 \end{aligned}$$

We zien dus dat de determinant van elke DH-transformatiematrixmatrix = 1. Dit wil zeggen dat de inverse ten allen tijde bestaat. Nu rest ons alleen nog om de  $adj(A)$  te bepalen.

De geadjugeerde van een vierkante matrix kan je berekenen door elk element te vervangen door zijn cofactor en die matrix vervolgens te transponeren. De elementen  $c_{ij}$  van de cofactorenmatrix zijn gelijk aan hun minor, op het teken na. De minor  $M_{ij}$  van een element  $c_{ij}$  is de determinant van de matrix die overblijft als je de i-de rij en de j-de kolom van matrix A schrappt. Met andere woorden:

$$c_{ij} = (-1)^{i+j} * M_{ij} \quad (2)$$

$$adj(A) = C^T \quad (3)$$

Laten we voor elk element  $c_{ij}$  de cofactor berekenen aan de hand van vergelijking (2). We nemen ter vereenvoudiging:

$$c = \cos \theta_n$$

$$s = \sin \theta_n$$

$$\bar{c} = \cos \alpha_n$$

$$\bar{s} = \sin \alpha_n$$

$$c_{11} = \begin{vmatrix} c\bar{c} & -c\bar{s} & rs \\ \bar{s} & \bar{c} & d \\ 0 & 0 & 1 \end{vmatrix} = c(\bar{c}^2 + \bar{s}^2) = c$$

$$c_{12} = - \begin{vmatrix} s & c\bar{c} & rs \\ 0 & \bar{c} & d \\ 0 & 0 & 1 \end{vmatrix} = -s\bar{c}$$

$$c_{13} = \begin{vmatrix} s & c\bar{c} & rs \\ 0 & \bar{s} & d \\ 0 & 0 & 1 \end{vmatrix} = s\bar{s}$$

$$c_{14} = - \begin{vmatrix} s & c\bar{c} & -c\bar{s} \\ 0 & \bar{s} & \bar{c} \\ 0 & 0 & 0 \end{vmatrix} = 0$$

$$c_{21} = - \begin{vmatrix} -s\bar{c} & s\bar{s} & rc \\ \bar{s} & \bar{c} & d \\ 0 & 0 & 1 \end{vmatrix} = s(\bar{c}^2 + \bar{s}^2) = s$$

$$c_{22} = \begin{vmatrix} c & s\bar{s} & rc \\ 0 & \bar{c} & d \\ 0 & 0 & 1 \end{vmatrix} = c\bar{c}$$

$$c_{23} = - \begin{vmatrix} c & -s\bar{c} & rc \\ 0 & \bar{s} & d \\ 0 & 0 & 1 \end{vmatrix} = -c\bar{s}$$

$$c_{24} = \begin{vmatrix} c & -s\bar{c} & s\bar{s} \\ 0 & \bar{s} & \bar{c} \\ 0 & 0 & 0 \end{vmatrix} = 0$$

$$c_{31} = \begin{vmatrix} -s\bar{c} & s\bar{s} & rc \\ c\bar{c} & -c\bar{s} & rs \\ 0 & 0 & 1 \end{vmatrix} = sc\bar{s}\bar{c} - sc\bar{s}\bar{c} = 0$$

$$c_{32} = - \begin{vmatrix} c & s\bar{s} & rc \\ s & -c\bar{s} & rs \\ 0 & 0 & 1 \end{vmatrix} = c^2\bar{s} + s^2\bar{s} = \bar{s}$$

$$c_{33} = \begin{vmatrix} c & -s\bar{c} & rc \\ s & c\bar{c} & rs \\ 0 & 0 & 1 \end{vmatrix} = c^2\bar{c} + s^2\bar{c} = \bar{c}$$

$$c_{34} = - \begin{vmatrix} c & -s\bar{c} & s\bar{s} \\ s & c\bar{c} & -c\bar{s} \\ 0 & 0 & 0 \end{vmatrix} = 0$$

$$c_{41} = - \begin{vmatrix} -s\bar{c} & s\bar{s} & rc \\ c\bar{c} & -c\bar{s} & rs \\ \bar{s} & \bar{c} & d \end{vmatrix} = -\bar{s}(s^2r\bar{s} + c^2\bar{s}r) + \bar{c}(-s^2r\bar{c} - rc^2\bar{c}) - d(sc\bar{s}\bar{c} - sc\bar{s}\bar{c}) = r\bar{s}^2 + r\bar{c}^2 = -r$$

$$\begin{aligned}
c_{42} &= \begin{vmatrix} c & s\bar{s} & rc \\ s & -c\bar{s} & rs \\ 0 & \bar{c} & d \end{vmatrix} = c(-dc\bar{s} - rs\bar{c}) - s(s\bar{s}d - rc\bar{c}) = -dc^2\bar{s} - rsc\bar{c} - s^2\bar{s}d + rsc\bar{c} = -d\bar{s} \\
c_{43} &= - \begin{vmatrix} c & -s\bar{c} & rc \\ s & c\bar{c} & rs \\ 0 & \bar{s} & d \end{vmatrix} = -c(c\bar{c}d - rs\bar{s}) + s(-s\bar{c}d - rc\bar{s}) = -c^2\bar{c}d + rsc\bar{s} - s^2\bar{c}d - rsc\bar{s} = -d\bar{c} \\
c_{44} &= - \begin{vmatrix} c & -s\bar{c} & s\bar{s} \\ s & c\bar{c} & -c\bar{s} \\ 0 & \bar{s} & \bar{c} \end{vmatrix} = c(c\bar{c}^2 + c\bar{s}^2) - s(-s\bar{c}^2 - s\bar{s}^2) = c^2 + s^2 = 1
\end{aligned}$$

Uit vergelijking (3) halen we dat de

$$\text{adj}(^{n-1}T_n) = C^T$$

In de vorige paragraaf hebben we alle elementen  $c_{ij}$  van C berekend, dus de

$$\text{adj}(^{n-1}T_n) = \begin{bmatrix} \cos \theta_n & \sin \theta_n & 0 & -r_n \\ -\sin \theta_n \cos \alpha_n & \cos \theta_n \cos \alpha_n & \sin \alpha_n & -d_n \sin \alpha_n \\ \sin \theta_n \sin \alpha_n & -\cos \theta_n \sin \alpha_n & \cos \alpha_n & -d_n \cos \alpha_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Uit vergelijking (1) kunnen we nu de inverse van de DH-transformatiematrix berekenen:

$${}^{n-1}T_n^{-1} = \frac{1}{|{}^{n-1}T_n|} * \text{adj}(^{n-1}T_n) = \left[ \begin{array}{ccc|c} \cos \theta_n & \sin \theta_n & 0 & -r_n \\ -\sin \theta_n \cos \alpha_n & \cos \theta_n \cos \alpha_n & \sin \alpha_n & -d_n \sin \alpha_n \\ \sin \theta_n \sin \alpha_n & -\cos \theta_n \sin \alpha_n & \cos \alpha_n & -d_n \cos \alpha_n \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

## B. Bewijs van de overbrengingsverhouding van de hoofdoverbrenging

Bij een normaal planeetwielmechanisme zijn de volgende grondformules geldig:

$$N_s \omega_s + N_p \omega_p - (N_s + N_p) \omega_c = 0 \quad (4)$$

$$N_r \omega_r - N_p \omega_p - (N_r - N_p) \omega_c = 0 \quad (5)$$

Waar na uitwerking volgt dat:

$$N_s \omega_s + N_r \omega_r = (N_s + N_r) \omega_c \quad (6)$$

Waarbij  $N_p$ ,  $N_s$  en  $N_r$  respectievelijk het aantal tanden zijn van het planeetwiel, het zonnewiel en de ring.  $\omega_p$ ,  $\omega_s$  en  $\omega_r$  zijn respectievelijk de hoeksnelheden van het planeetwiel, het zonnewiel en de ring.

Omdat wij met een samengesteld planeetwielmechanisme zitten, moeten we het onderscheid maken tussen bijvoorbeeld  $N_{s1}$  en  $N_{s2}$ . We spreken af dat het planeetwielmechanisme waarvan het zonnewiel als aandrijver gebruikt wordt het eerste planeetwielmechanisme genoemd wordt en dat de bijhorende waarden een 1 in de subscript krijgen. Het ander planeetwielmechanisme krijgt overal een 2 in de subscript. We kunnen het probleem opsplitsen in twee verschillende gevallen. Het eerste geval is wanneer de ring van het eerste mechanisme vastgezet is ( $\omega_{r1} = 0$ ) en het tweede geval is wanneer de ring van het tweede mechanisme vastgezet is ( $\omega_{r2} = 0$ )

### Geval 1

In het eerste geval zoeken we  $\omega_{r2}$  bij een gegeven  $\omega_{s1}$ .

Eerst en vooral bepalen we de hoeksnelheid van de planeetdrager van het eerste mechanisme door gebruik te maken van formule (6). Namelijk

$$\omega_{c1} = \omega_{c2} = \frac{N_{s1} \omega_{s1} + N_{r1} \omega_{r1}}{N_{s1} + N_{r1}} = \frac{N_{s1} \omega_{s1}}{N_{s1} + N_{r1}}$$

Aangezien de planeetwieldragers van beide mechanismen aan elkaar vastgehangen zijn, draaien ze even snel ( $\omega_{c1} = \omega_{c2}$ ). Het is gegeven dat  $\omega_{r1} = 0$ , dus vandaar de vereenvoudiging. Vervolgens kan ik aan de hand van formule (6) berekenen wat de hoeksnelheid is van de ring van het tweede mechanisme. Namelijk

$$\omega_{r2} = \frac{(N_{s2} + N_{r2}) \omega_{c2} - N_{s2} \omega_{s2}}{N_{r2}}$$

Alle elementen zijn gekend behalve  $\omega_{s2}$ . Dat element kunnen we afleiden door gebruik te maken van het feit dat  $\omega_{p1} = \omega_{p2}$  aangezien de planeetwielen uit 1 stuk bestaan. Met formule [44] kunnen we afleiden dat

$$\omega_{p2} = \omega_{p1} = \frac{(N_{s1} + N_{p1}) \omega_{c1} - N_{s1} \omega_{s1}}{N_{p1}}$$

En vervolgens kunnen we  $\omega_{s2}$  met formule (4) als volgt bepalen

$$\omega_{s2} = \frac{(N_{s2} + N_{p2}) \omega_{c2} - N_{p2} \omega_{p2}}{N_{s2}}$$

## Geval 2

In het tweede geval zoeken we  $\omega_{r1}$ .

Eerst en vooral bepalen we de snelheid van de ring van het eerste mechanisme door gebruik te maken van formule (6). Namelijk

$$\omega_{r1} = \frac{(N_{s1} + N_{r1})\omega_{c1} - N_{s1}\omega_{s1}}{N_{r1}}$$

Aangezien de planeetwieldragers van beide mechanismen aan elkaar vastgehangen zijn, draaien ze even snel ( $\omega_{c1} = \omega_{c2}$ ). Het enige wat we nog moeten doen is het zoeken van  $\omega_{c1} = \omega_{c2}$ . Daarvoor moeten we eerst  $\omega_{p1} = \omega_{p2}$  zoeken. Deze twee hoeksnelheden zijn opnieuw gelijk aan elkaar omdat ze nog steeds aan elkaar vasthangen. Door respectievelijk gebruik te maken van de formules (4) en (5) kunnen we deze hoeksnelheden op twee verschillende manieren bepalen:

$$\begin{aligned}\omega_{p1} &= \omega_{p2} = \frac{(N_{s1} + N_{p1})\omega_{c1} - N_{s1}\omega_{s1}}{N_{p1}} \\ \omega_{p1} &= \omega_{p2} = \frac{(N_{r2} - N_{p2})\omega_{c2} - N_{r2}\omega_{r2}}{-N_{p2}} = \frac{(N_{r2} - N_{p2})\omega_{c1}}{-N_{p2}}\end{aligned}$$

Door het samenvoegen van de twee bovenstaande vergelijkingen kunnen we  $\omega_{c1}$  bepalen:

$$\begin{aligned}\frac{(N_{s1} + N_{p1})\omega_{c1} - N_{s1}\omega_{s1}}{N_{p1}} &= \frac{(N_{r2} - N_{p2})\omega_{c1}}{-N_{p2}} \\ \Leftrightarrow \omega_{c1} &= \frac{N_{s1}N_{p2}\omega_{s1}}{(N_{r2} - N_{p2})N_{p1} + (N_{s1} + N_{p1})N_{p2}}\end{aligned}$$

## Toegepast

Afhankelijk van welk mechanisme je als eerste mechanisme gebruikt, zal de overbrengingsverhouding anders zijn. We nemen voor elke berekening  $\omega_{s1} = 1\text{ rad/s}$ . Aangezien we overbrengingsverhoudingen aan het berekenen zijn maakt die snelheid niet zo veel uit.

**Eerste mogelijkheid** De eerste mogelijkheid heeft de volgende parameters:

$N_{ij}$	aantal tanden
$N_{s1}$	10
$N_{p1}$	14
$N_{r1}$	38
$N_{s2}$	12
$N_{p2}$	12
$N_{r2}$	36

Tab. 4: De parameters van de tweede mogelijkheid

Als we de formules voor het eerste geval toepassen en dus  $\omega_{r2}$  berekenen, krijgen we  $\omega_{r2} = 0.0198412699$ . Dat is een snelheidsreductie van 50.4 en de rotatiezin blijft ongewijzigd. Als we de formules voor het tweede geval toepassen en dus  $\omega_{r1}$  berekenen, krijgen we  $\omega_{r1} = -0.020242915$ , wat neerkomt op een snelheidsreductie 49.4 en een

verandering in draaizin. Zoals je kan zien maakt het dus echt wel uit welke ring (van het eerste of het tweede planeetwielmechanisme) je vastzet en welke je gebruikt als aangedreven ring.

**Tweede mogelijkheid** Er is natuurlijk nog een tweede mogelijkheid. Hierbij wisselen we het eerste en het tweede planeetwielmechanisme gewoon om. Het zonnewiel dat bij de eerste mogelijkheid als 'idler' diende, zal nu de aandrijver zijn en vice versa. De parameters van de tweede mogelijkheid zijn dan ook tegengesteld aan die van de eerste. Namelijk:

$N_{ij}$	aantal tanden
$N_{s1}$	12
$N_{p1}$	12
$N_{r1}$	36
$N_{s2}$	10
$N_{p2}$	14
$N_{r2}$	38

Tab. 5: De parameters van de eerste mogelijkheid

Als we de formules voor het eerste geval toepassen en dus  $\omega_{r2}$  berekenen, krijgen we  $\omega_{r2} = -\frac{1}{38}$ . Dat is een snelheidsreductie van 38 en een andere rotatiezin. Als we de formules voor het tweede geval toepassen en dus  $\omega_{r1}$  berekenen, krijgen we  $\omega_{r1} = \frac{1}{39}$ , wat neerkomt op een snelheidsreductie van 39 met een andere draaizin. Zoals je kan zien maakt het dus niet alleen uit welke ring (van het eerste of het tweede planeetwielmechanisme) je vastzet en welke je gebruikt als aangedreven ring, maar ook welk zonnewiel je gebruikt als aandrijvend tandwiel.

## C. Wiskundige uitwerking van een trapeziumvormige trajectory planner

Deze bijlage zal een aantal formules uitwerken om een trapeziumvormige trajectory planner te kunnen maken. Met trapeziumvormige trajectory planner bedoel ik een systeem dat closed-loop positiecontrole kan uitvoeren op een motor, rekening houdend met acceleratie, deacceleratie en de maximale toegelaten snelheid.

Wat we eigenlijk proberen te berekenen is het functievoorschrift van een trapeziumvormige grafiek dat de hoeksnelheid van de motor in functie van tijd voorstelt voor een willekeurige acceleratie  $\alpha_1 > 0$ , een willekeurige maximale snelheid  $\omega_m > 0$ , een willekeurige deacceleratie  $\alpha_2 > 0$  en een willekeurige positieverandering  $\theta_t > 0$ .

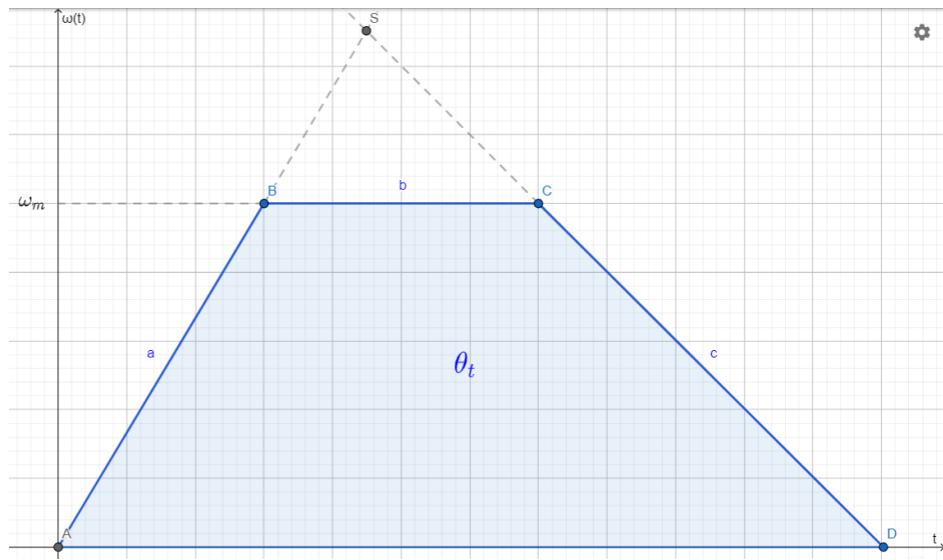


Fig. 93: De meest algemene vorm van de trapeziumvormige grafiek

In het begin gaan we er van uit dat de trapeziumvormige grafiek er zoals bovenstaande grafiek uitziet. Dan kunnen we de grafiek indelen in de drie rechten  $a$ ,  $b$ , en  $c$ . Op basis van de gegeven parameters ( $\alpha_1$ ,  $\alpha_2$ ,  $\theta_t$  en  $\omega_m$ ) hebben ze de volgende vergelijkingen:

$$\begin{aligned} a &\leftrightarrow \omega = \alpha_1 t \\ b &\leftrightarrow \omega = \omega_m \\ c &\leftrightarrow \omega = -\alpha_2 t + q \end{aligned}$$

Om het functievoorschrift  $\omega(t)$  te kunnen bepalen moeten we de coördinaten van punten B, C en D bepalen. Dit komt neer op het zoeken van snijpunten tussen bepaalde rechten. Na uitwerking volgt dat:

$$\begin{aligned} co(B) &= \left( \frac{\omega_m}{\alpha_1}, \omega_m \right) \\ co(C) &= \left( \frac{q - \omega_m}{\alpha_2}, \omega_m \right) \\ co(D) &= \left( \frac{q}{\alpha_2}, 0 \right) \end{aligned}$$

De enige onbekende dat we nog moeten berekenen is  $q$ . Dit valt te doen door gebruik te maken van het feit dat  $\theta_t$  ook gelijk is aan de integraal van deze grafiek (de oppervlakte

onder de grafiek)  $\int \omega(t)dt$ . De oppervlakte van de grafiek kan berekend worden met de standaardformule voor het berekenen van de oppervlakte van een trapezium. Namelijk:

$$\begin{aligned}\theta_t &= \int_0^{\frac{q}{\alpha_2}} \omega(t)dt = \frac{\omega_m}{2}(|BC| + |AD|) \\ &= \frac{\omega_m}{2} \left( \frac{q - \omega_m}{\alpha_2} - \frac{\omega_m}{\alpha_1} + \frac{q}{\alpha_2} \right)\end{aligned}$$

Waar na herwerking volgt dat:

$$q = \frac{\alpha_2 \theta_t}{\omega_m} + \frac{\alpha_2 \omega_m}{2\alpha_1} + \frac{\omega_m}{2}$$

Voordat we het functievoorschrift kunnen berekenen, moeten we een onderscheid maken tussen twee mogelijke gevallen:

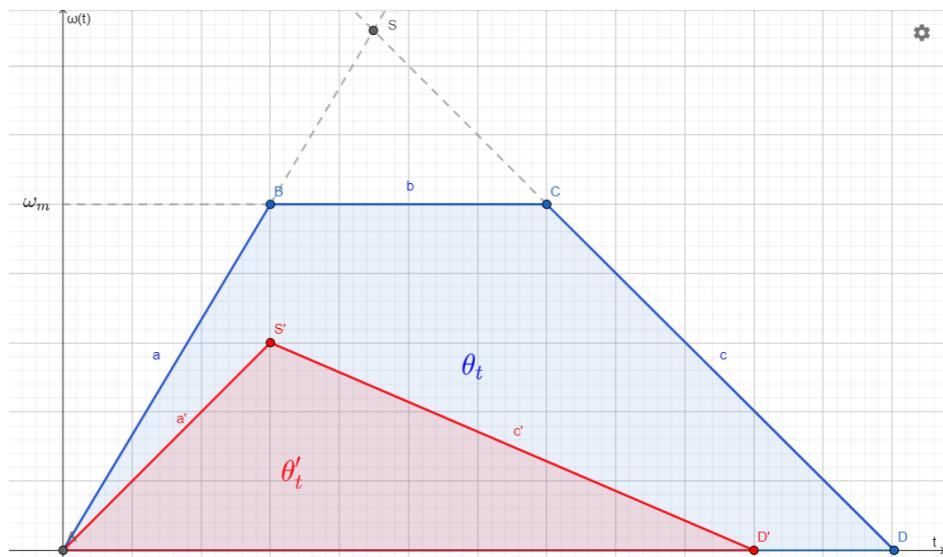


Fig. 94: De twee gevallen van de trapeziumvormige grafiek.

Bij het eerste geval (blauw) is de positieverandering groot genoeg zodat de motor even op volle snelheid zal draaien. Bij het tweede geval (rood) zal de motor nooit aan de maximale snelheid geraken. We kunnen de twee gevallen van elkaar onderscheiden door te controleren dat de y-coördinaat van het snijpunt  $S$  van de stijgende en de dalende rechte kleiner of groter is dan  $\omega_m$ . De coördinaat van het snijpunt tussen de stijgende rechte  $a$  en de dalende rechte  $c$  kan eenvoudig berekend worden:

$$co(S) = \left( \frac{q}{\alpha_1 + \alpha_2}, \frac{\alpha_1 q}{\alpha_1 + \alpha_2} \right)$$

Nu zijn alle onbekenden berekend en kunnen we de grenzen opstellen van het functievoorschrift van  $\omega(t)$ .

### Geval 1: $S_y > \omega_m$

In dit geval kunnen we het functievoorschrift als volgt bepalen:

$$\omega(t) = \begin{cases} \alpha_1 t & 0 \leq t < \frac{\omega_m}{\alpha_1} \\ \omega_m & \frac{\omega_m}{\alpha_1} \leq t < \frac{q-\omega_m}{\alpha_2} \\ -\alpha_2 t + q & \frac{q-\omega_m}{\alpha_2} \leq t < \frac{q}{\alpha_2} \end{cases}$$

De maximale snelheid is  $\omega_m$  en de uitvoeringstijd is  $\frac{q}{\alpha_2}$

### Geval 2: $S_y \leq \omega_m$

In dit geval kunnen we het functievoorschrift als volgt bepalen:

$$\omega(t) = \begin{cases} \alpha_1 t & 0 \leq t < \frac{q}{\alpha_1 + \alpha_2} \\ -\alpha_2 t + q & \frac{q}{\alpha_1 + \alpha_2} \leq t < \frac{\alpha_1 q}{\alpha_1 + \alpha_2} \end{cases}$$

De maximale snelheid is  $\frac{\alpha_1 q}{\alpha_1 + \alpha_2}$  en de uitvoeringstijd is  $\frac{q}{\alpha_2}$ .

## Sturing

Op basis van het functievoorschrift kunnen we snelheidscommando's naar de motor sturen of positiecommando's als we de integraal  $\int_0^t \omega(t) dt$  berekenen. Bij het eerste geval zou die er als volgt uitzien:

$$\int_0^t \omega(t) dt = \begin{cases} \frac{\alpha_1 t^2}{2} & 0 \leq t < \frac{\omega_m}{\alpha_1} \\ \frac{\omega_m^2}{2\alpha_1} + \omega_m t & \frac{\omega_m}{\alpha_1} \leq t < \frac{q-\omega_m}{\alpha_2} \\ \frac{\omega_m^2}{2\alpha_1} + \frac{\omega_m(q-\omega_m)}{\alpha_2} - \frac{\alpha_2 t^2}{2} + qt & \frac{q-\omega_m}{\alpha_2} \leq t < \frac{q}{\alpha_2} \end{cases}$$

en bij het tweede geval zou die integraal er als volgt uitzien:

$$\int_0^t \omega(t) dt = \begin{cases} \frac{\alpha_1 t^2}{2} & 0 \leq t < \frac{q}{\alpha_1 + \alpha_2} \\ \frac{\alpha_1 q^2}{2(\alpha_1 + \alpha_2)^2} - \frac{\alpha_2 t^2}{2} + qt & \frac{q}{\alpha_1 + \alpha_2} \leq t < \frac{\alpha_1 q}{\alpha_1 + \alpha_2} \end{cases}$$

## D. Ma commande pour mon épreuve intégrée

Monsieur

Je suis Lowiek Van den Stockt. Je suis un élève de sciences industrielles dans la dernière année sur le VTI Aalst. Le sujet de mon épreuve intégrée est un bras robotisé six axes.

Je pense que vous et votre entreprise Leroy S.A. pouvez m'aider avec mon épreuve intégrée. Ci-dessous vous trouvez ma commande:

- 1 x alimentation de 24V et 25A pour €24.95 par pièce
- 6 x moteur de 200W pour €19.98 par pièce
- 6 x commande électronique pour les moteurs pour €39.95 par pièce

Vous pouvez livrer ma commande à Processieweg 6, 9450 Haaltert en Belgique.

Merci d'avance et j'espère recevoir une réponse bientôt.

Cordialement

Lowiek Van den Stockt

## E. My integrated project

My thesis consists of designing, building and researching a six axis robotic arm. For the articulations I am using brushless motors. These motors have a very high efficiency. Hence, they are faster and more powerful than similarly sized non-brushless motors. The only disadvantage is that it is hard to achieve precise position control due to the necessity of external positional feedback.

My robotic arm will be controlled by entering the positional and rotational information of the end effector as an axis-angle pair. Developing a kinematical model for a robotic arm is quite challenging, so that is the reason that I will discuss three different kinematical models. Those kinematical models take the axis-angle pair as input, and after some calculations it will control the individual articulations to finally achieve the requested end effector state. Furthermore, there are some additional mathematical models required to make sure the robotic arm doesn't crash.

As you can see, the mathematics behind it aren't easy. On the contrary, a mathematical challenge doesn't scare me. Moreover, that is one of the main reasons why I chose this project. Almost everything is represented in this thesis: physics, mechanics, electricity, computer science, design, software development, mathematics, time management, self-study skills to name a few. I think that it is necessary to challenge yourself when making a thesis. Only by this way you get to know what your weak points are. Knowing your weak points is very useful information to have when you are going to university.

Another reason that I chose this project is because I wanted to improve a robotic arm that I assembled for Fablab Factory in December 2017. That robotic arm only has five axes and is rather slow as a result of the used motor technology.

After school, this robotic arm will not be stored in the attic to never move again. In the little free time I will have next year, I will keep on working on it to make improvements and upgrades, to finally release the whole robotic arm as an open-source project. One of the reasons is to give something back to the open source community. Without that community, the project would have failed in any case. Furthermore, there aren't a lot of open-source options if someone wants to experiment with a robotic arm that has six axes and brushless motors.

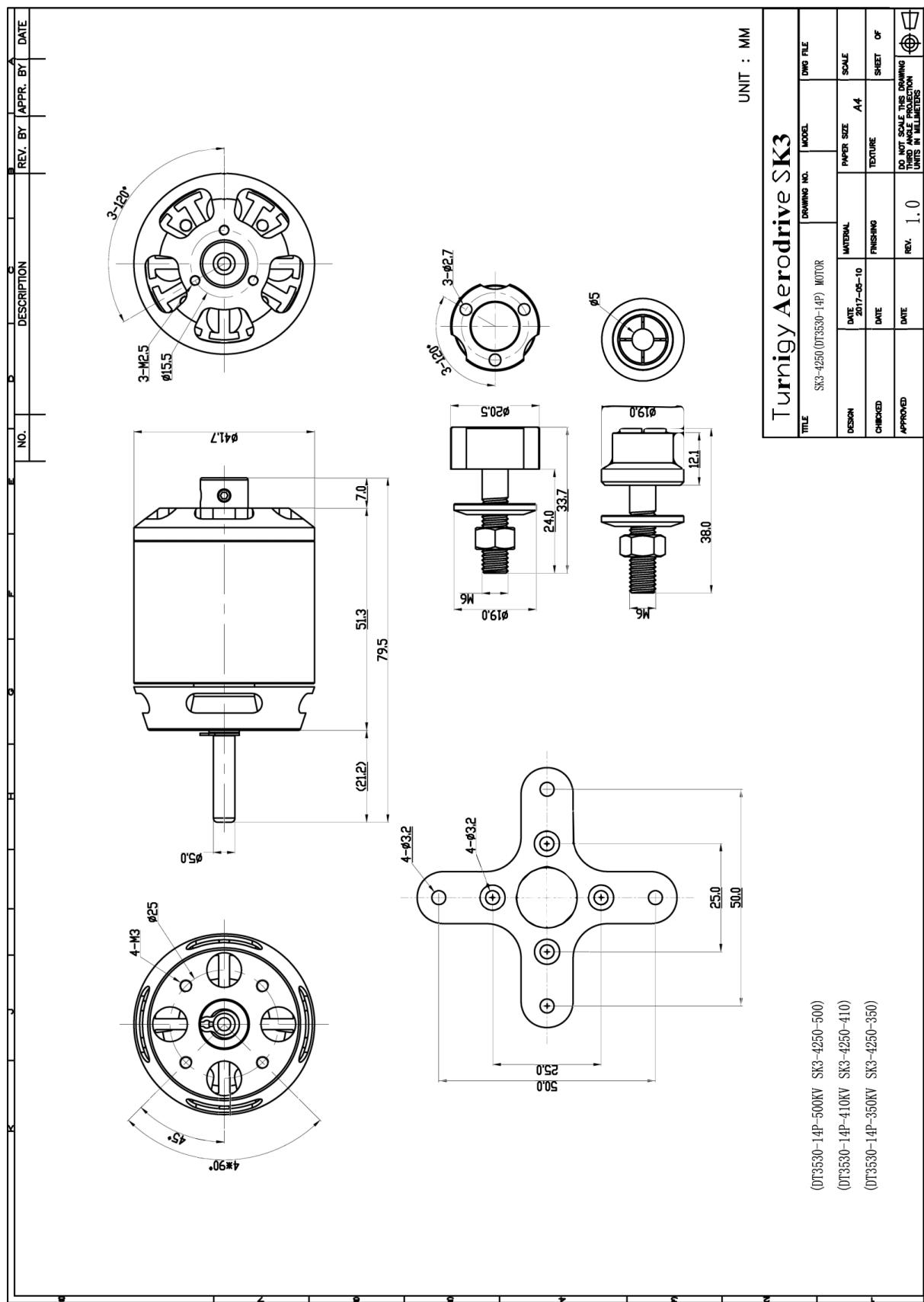
This paper is rather theoretically oriented, mostly containing discussions of mathematical subjects. To keep this paper readable for everyone, I did my best to group the theories into chapters and/or subchapters, each having their own difficulty grade.

To make this project succeed, I had to work a lot on it and I had to manage my time properly. This is because of the fact that sometimes I needed to wait for a very long 3D-print or items that still needed to be delivered.

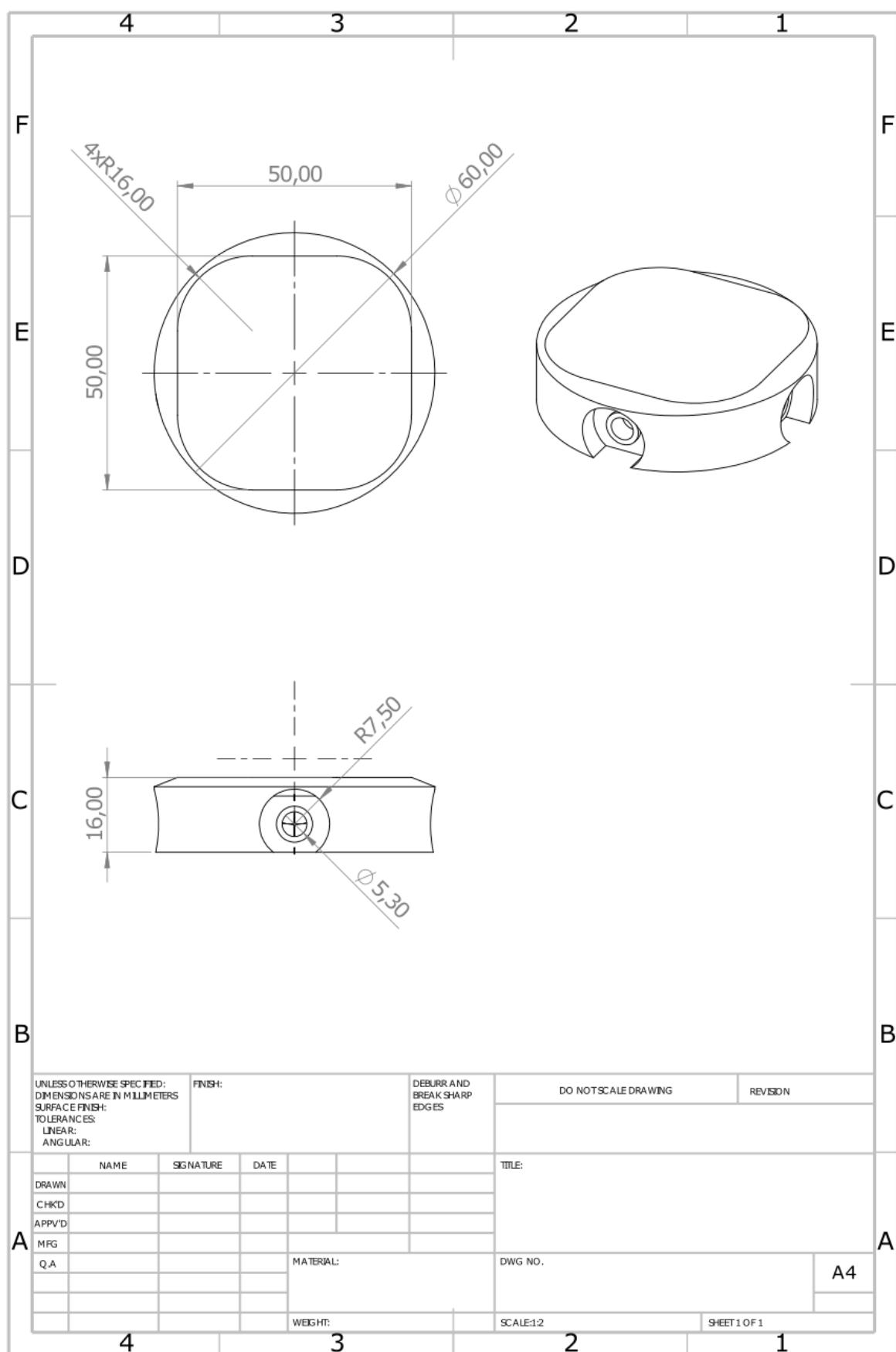
## F. Geschatte kostprijsberekening

ID	Part	Price/Piece	Quantity	Effective total
	<b>Bearings</b>			<b>€ 21,21</b>
1	16014	€ 3,070	1	€ 3,07
2	605-ZZ	€ 0,160	15	€ 2,39
4	6812-2RS	€ 2,340	3	€ 7,02
6	608-ZZ	€ 0,219	8	€ 1,75
7	6808ZZ	€ 0,740	1	€ 0,74
8	6710-ZZ	€ 3,120	2	€ 6,24
	<b>Fasteners</b>			<b>€ 25,50</b>
30	M5x16 DIN 7985	€ 0,075	50	€ 3,75
31	M4x16 DIN 7985	€ 0,080	100	€ 8,00
32	M4x50 DIN 7985	€ 0,250	10	€ 2,50
34	M5 Nut DIN 934	€ 0,065	50	€ 3,25
35	M4 Nut DIN 934	€ 0,030	100	€ 3,00
36	M3 Varia	€ 0,070	50	€ 3,50
37	M3 Nut	€ 0,030	50	€ 1,50
	<b>3D-printing &amp; Lasercutting</b>			<b>€ 120,00</b>
60	PLA	€ 100,000	1	€ 100,00
61	Lasercutting	€ 20,000	1	€ 20,00
	<b>Motors</b>			<b>€ 178,11</b>
90	Turnigy Aerodrive SK3 4250 350kv	€ 38,140	3	€ 114,42
91	Propdrive v2 3536 910kv	€ 21,230	3	€ 63,69
	<b>Encoders</b>			<b>€ 46,80</b>
120	Encoder 600P/R	€ 7,800	6	€ 46,80
	<b>Pulley &amp; Belts</b>			<b>€ 39,72</b>
150	HTD 3M 234mm	€ 3,480	2	€ 6,96
151	HTD 3M 204mm	€ 3,680	1	€ 3,68
152	GT2 belt	€ 0,200	1	€ 0,20
153	Pulley 20T 8mm	€ 6,600	2	€ 13,20
154	Pulley 20T 5mm	€ 6,400	2	€ 12,80
155	HTD 3M Belt 86T 9mm	€ 2,880	1	€ 2,88
	<b>Electronics</b>			<b>€ 446,08</b>
181	Power Supply 24V 16A (400W)	€ 29,360	3	€ 88,08
182	ODrive	€ 115,000	3	€ 345,00
183	Servo HS422	€ 13,000	1	€ 13,00
	<b>Other</b>			<b>€ 54,84</b>
210	8mm steel rod 100cm	€ 3,750	1	€ 3,75
211	Cables/connectors	€ 50,000	1	€ 50,00
212	Stopbutton	€ 1,090	1	€ 1,09
	<b>Total</b>			<b>€ 932,26</b>

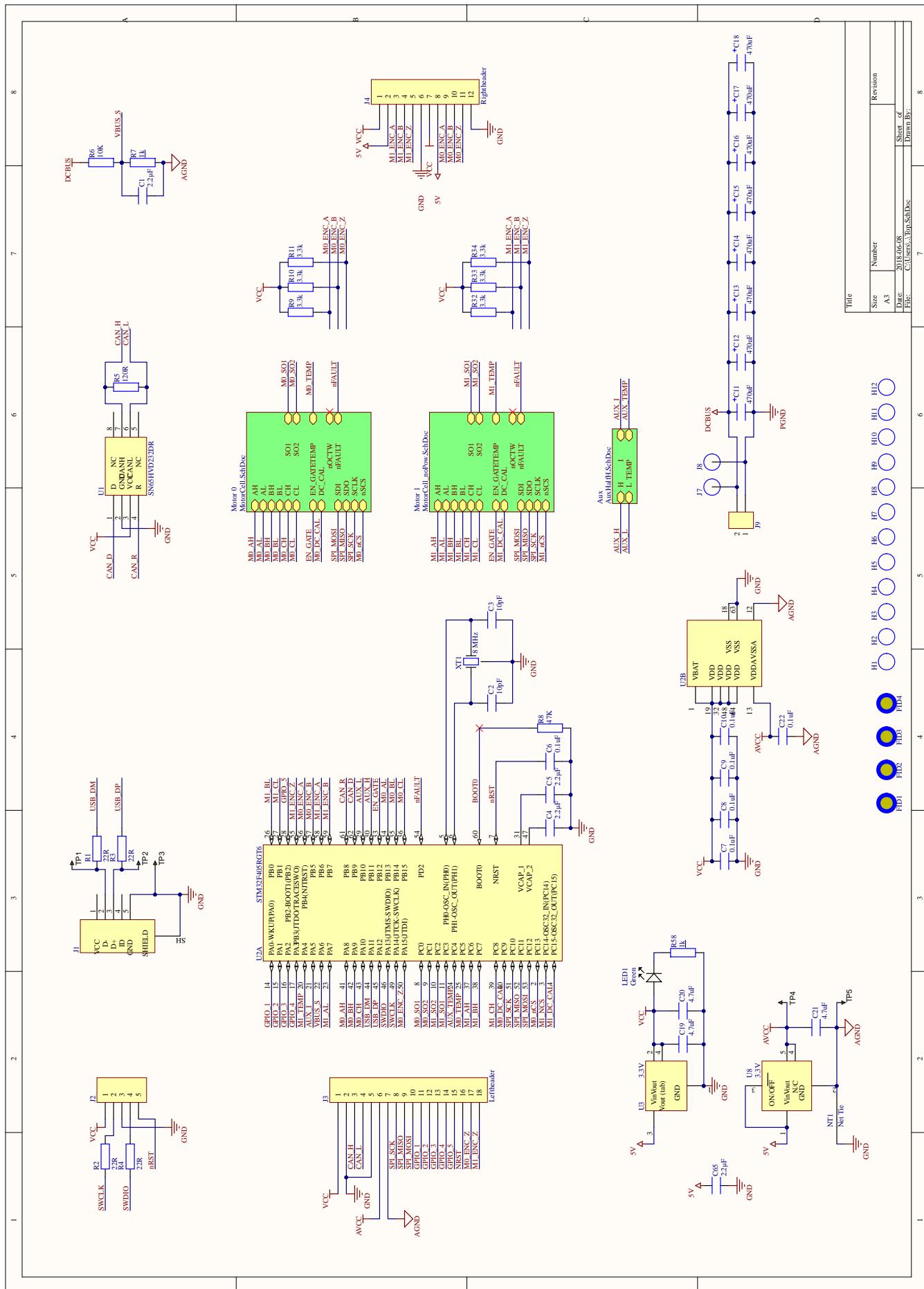
G. Afmetingen van de Turnigy SK3 Aerodrive 4250 - 350Kv



## H. Afmetingen van de algemene end-effector interface



# I. Elektrisch schema van een ODrive



Page	7	8
Title		
Size	A3	
Number		
Date	2018/06/18	
File	C:\Users\Jip.SchDoc	
Revision		
Sheet of		
Drawn By:		

