|  |
| --- |
| Gebruikershandleiding\_  Simulatie WoR Worlds |

Mike Ackerschott

26 oktober 2023

INHOUDSOPGAVE

[1 Installatie 3](#_Toc149232907)

[2 Opstarten en bewegen van de arm 3](#_Toc149232908)

[2.1 Bewegen m.b.v demo script 6](#_Toc149232909)

[3 Gerealiseerde eisen 7](#_Toc149232910)

[3.1 A.1. PACKAGE 7](#_Toc149232911)

[3.1.1 Toelichting 7](#_Toc149232912)

[3.2 A.2. VIRTUELE SERVO CONTROLLER 7](#_Toc149232913)

[3.2.1 Toelichting 7](#_Toc149232914)

[3.3 A.3. VIRTUEEL BEKERTJE 8](#_Toc149232915)

[3.3.1 Toelichting 8](#_Toc149232916)

[3.4 A.4. DEMONSTRATIE-INFRASTRUCTUUR 8](#_Toc149232917)

[3.4.1 Toelichting 9](#_Toc149232918)

[3.5 A.5. GEBRUIKERSHANDLEIDING 9](#_Toc149232919)

[3.5.1 Toelichting 9](#_Toc149232920)

[3.6 A.6. ONTWERPDOCUMENTATIE 9](#_Toc149232921)

[3.6.1 Toelichting 9](#_Toc149232922)

# Installatie

Voordat je begint met het installeren van dit programma, is het handig om eerst een ros2\_ws folder aan te maken. Kies een directory waar je het programma wilt testen en voer het volgende commando uit:

**mkdir ros2\_ws**

**cd ros2\_ws**

In deze nieuwe folder kun je nu het programma in zetten. De folderstructuur ziet er nu als volgt uit:

**ros2\_ws/**

**wor\_\_worlds\_simulatie/**

**docs/**

**msg\_srv/**

**simple\_sim\_movement/**

**demo.sh**

Nu dat het programma correct in de ros2 workspace folder staat, kun je beginnnen met het compileren van het programma. Open een terminal in de ros2\_ws directory en voer het volgende commando’s uit:

**. /opt/ros/humble/setup.bash   
rosdep install -i --from-path ./wor\_\_worlds\_simulatie/ --rosdistro humble -y**  
**colcon build**

Nadat dit klaar is en geen errors geeft kun je verder met het opstarten van de simulatie

# Opstarten en bewegen van de arm

Open een terminal in de ros2\_ws directory en voer het commando ‘ls’ uit. Als het bouwen volgens de handleiding is uitgevoerd ziet de folderstructuur er zo uit:

**ros2\_ws/**

**build/**

**install/**

**log/**

**wor\_\_worlds\_simulatie/**

**docs/**

**msg\_srv/**

**simple\_sim\_movement/**

**demo.sh**

Om het programma op te kunnen starten, moet je eerst je lokale ros2 workspace sourcen. Dit doe je door het volgende commando uit te voeren in de ros2\_ws directory:

**. install/setup.bash**

Nu dat het is gesourcet, kun je het programma uitvoeren door het volgende commando uit te voeren:  
  
**ros2 launch simple\_sim\_movement display.launch.py**

Er openen nu 2 terminals. Een terminal is voor het uitvoeren van de simulatie, zoals de robotarm, het bekertje en rviz zelf. De tweede terminal is voor het besturen van de robotarm volgens de AL5D commando-structuur. Hierin kun je dus de commando’s sturen zodat de robotarm beweegt.

Een commando moet voldoen aan de volgende eisen

- Een singleServoCommand (oftewel een commando die maar één servo aanstuurt) bevat minimaal een aan te sturen servo, een pwm-signaal en een bewegingsduratie of bewegingssnelheid.

Het kan er als volgt uitzien:

#0 P2500 T3000  
Dit zet servo 0 op pwm 2500. De beweging duurt 3 seconden

#0 P2500 S500

Dit zet servo 0 op pwm 2500. De servo beweegt met een snelheid van 500 pwm per seconde

#0 P2500 S500 T10000

Dit zet de servo 0 op pwm 2500. als de duratie langer is dan het bewegen van de servo van huidige positie naar positie 2500 met een snelheid van 500, wordt de meegegeven snelheid genegeerd. Dit betekend dus dat de beweging 10 seconden zou duren. In dit geval, omdat de verste stand die de servo kan staan pwm500 is, zal de snelheid dus altijd genegeerd worden, omdat een maximale bewegings-duratie met een snelheid van 500, 2 seconden is.

2500 – 500 = 2000

2000 / 500 = 2 seconden

10 seconden > 2 seconden

Net als bij de AL5D robotarm, kan er bij deze robotarm ook meerdere servos tegelijk aangestuurd worden. Hiervoor is het belangrijk om het volgende te onthouden

- Één channel wordt maar één keer aangeroepen per commando. Hiermee bedoel ik dat een commando niet meerdere keren dezelfde channel gebruikt. Dit mag dus NIET:  
#0 P1500 #0 P2000 T3000

Dit mag dus wel:  
#0 P1500 #1 P2000 T3000

- Een aangegeven pulse (bijvoorbeeld P1500) wordt maar één keer meegegeven voor elke channel. Dit mag dus NIET: #0 P1500 P2000 S500  
Dit mag dus wel: #0 P1500 S500

- Het meegeven van een snelheid (bijvoorbeeld S500) is verplicht voor elke aangegeven channel, zolang er geen duratie is meegegeven, aangezien er anders geen beweging geplant kan worden.   
Dit mag dus NIET: #0 P1500 S500 S1000  
Dit mag dus NIET: #0 P1500   
Dit mag dus wel: #0 P1500 S500

- Het meegeven van een duratie gebeurt maar één keer in een hele commando. Dit ziet er uit als volgt: #0 P1500 #1 P1600 T3000. Dit beweegt servo 0 naar pwm 1500 en servo 1 naar pwm 1600. De hele beweging duurt 3 seconden.  
Dit mag dus NIET: #0 P1500 T3000 #1 P1500 T4000  
Dit mag dus wel: #0 P1500 #1 P1500 T4000

- Het meegeven van een duratie is verplicht als een aangegeven channel met een pulse geen snelheid heeft meegekregen, aangezien er anders geen motion geplant kan worden.   
Dit mag dus NIET: #0 P1500  
Dit mag dus wel: #0 P1500 T3000

Een commando waar meerdere servo’s tegelijk aangestuurd worden ziet er bijvoorbeeld als volgt uit: #0 P1500 #1 P1500 #2 P1500 #3 P1500 #4 P500 #5 P1500 T3000

Dit beweegt de robotarm naar een soort ready positie, waar hij rechtop staat en de voorarm met een hoek van 90 graden horizontaal gebogen staat. De gripper is open. De beweging duurt in totaal 3 seconden

Een ander commando waar meerdere servo’s tegelijk aangestuurd worden ziet er als volgt uit:  
#0 P1500 #1 P1500 #2 P500 #3 P1500 #4 P500 #5 P1500 T3000

Dit beweegt de robotarm naar een straight-up positie.

## Bewegen m.b.v demo script

Aangezien het onhandig is om constant commando’s in het inputveld te typen om te kijken of de robotarm goed beweegt en of het bekertje goed opgepakt kan worden, is er een demoscript aanwezig waarin duidelijk te zien is dat het bekertje opgepakt kan worden en de robotarm realistisch beweegt

Om de demoscript te runnen is het belangrijk om eerst de simulatie te runnen. Dit doe je als volgt.  
In de ros2\_ws directory voer je de volgende commando’s uit:

**. install/setup.bash**

**ros2 launch simple\_sim\_movement display.launch.py**

Nu dat de simulatie op is gestart voer je in een andere terminal, ook in de ros2\_ws directory, het volgende commando uit:

**. install/setup.bash**

**./wor\_\_worlds\_simulatie/demo.sh**

Nu begint de robotarm met het uitvoeren van een aantal bewegingen, waarbij het bekertje wordt opgepakt, rondbewogen, laten vallen en weer bij de originele plek wordt teruggezet

# Gerealiseerde eisen

## A.1. PACKAGE

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Omschrijving | Prio | Gerealiseerd |
| PA01 | Alle code is gepackaged volgens de ROS-directorystructuur. | Should | Nee |
| PA02 | Package is te bouwen met colcon op ROS2 Humble Hawksbill | Must | Ja |
| PA03 | De applicatie wordt gebouwd met C++ volgens de Object Oriented principes die je geleerd hebt bij eerdere courses. | Must | Ja |
| PA04 | Alle code voldoet aan de ROS C++ Style Guide. | Should | Nee |

### Toelichting

Het programma is te bouwen met behulp van colcon build op ROS2 Humble Hawksbill. Tevens worden ook de Object Oriented principles uit eerdere courses toegepast in dit programma.

## A.2. VIRTUELE SERVO CONTROLLER

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Omschrijving | Prio | Gerealiseerd |
| VS01 | De virtuele controller luistert naar een topic waarop string messages in het formaat van de SSC-32U 1 worden geplaatst. Van de interface moeten ten minste commando’s zijn opgenomen voor het verplaatsen van de servo’s met een ingestelde duur en het stoppen van de servo’s. | Must | Ja |
| VS02 | De virtuele controller reageert op het topic (zie eis VS01) door bijbehorende joint\_state messages te publiceren. | Must | Ja |
| VS03 | De virtuele robotarm wordt gevisualiseerd in Rviz (een URDF-model van de arm is beschikbaar op OnderwijsOnline). | Must\* | Ja |
| VS04 | De virtuele robotarm gedraagt zich realistisch m.b.t. tijdgedrag (servo’s roteren kost tijd en gaat geleidelijk). | Must | Ja |
| VS05 | De virtuele robotarm kan op een willekeurige plaats in de virtuele wereld geplaatst worden. | Should | Nee |

### Toelichting

De virtuele controller ontvangt berichten via een topic volgens het SSC-32U 1 formaat. Hierbij kunnen alle servos worden bewogen met een ingestelde duur, door joint\_state messages te publiceren naar TF2. De beweging is gevisualiseerd in Rviz en de meegegeven URDF-model is gebruikt. Alle servo’s hebben een maximale bewegingssnelheid en zullen niet reageren op berichten waarbij een pwm die buiten de grenzen (500-2500) ligt, is meegegeven.

## A.3. VIRTUEEL BEKERTJE

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Omschrijving | Prio | Gerealiseerd |
| VC01 | Er kan op een willekeurige plek in de virtuele wereld een bekertje geplaatst worden. | Should | Nee |
| VC02 | Publiceert een 3D-visualisatie van het bekertje voor Rviz. | Must\* | Ja |
| VC03 | Detecteert de relevante punten van de gripper. | Should | Nee |
| VC04 | Visualiseert de gedetecteerde punten van de gripper. | Could\* | Nee |
| VC05 | Visualiseert wanneer de gripper het bekertje vastheeft. | Should | Nee |
| VC06 | Het bekertje beweegt mee met de gripper (als hij vastgehouden wordt). | Must | Ja |
| VC07 | Bekertje is onderhevig aan zwaartekracht wanneer losgelaten. | Must | Ja |
| VC08 | Bekertje bepaalt en publiceert zijn positie. | Must | Ja |
| VC09 | Bekertje bepaalt en publiceert zijn snelheid. | Should | Nee |
| VC10 | Snelheid wordt getoond met rqt\_plot. | Could | Nee |

### Toelichting

Het bekertje wordt 3D gevisualiseerd in Rviz met behulp van een .stl bestand. Wanneer de gripper volledig dicht is, dicht in de buurt van het bekertje, is het bekertje opgepakt en zal het meebewegen en meedraaien met de robotarm. Wanneer de gripper volledig open is, wordt het bekertje beinvloed door zwaartekracht en valt het recht omlaag. Tevens wordt de positie van het bekertje gepubliceerd op de topic /cup\_pos

## A.4. DEMONSTRATIE-INFRASTRUCTUUR

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Omschrijving | Prio | Gerealiseerd |
| DI01 | Een demoscript stuurt over de tijd een sequentie van commando’s naar de armcontroller. | Must | Ja |
| DI02 | Locatie van het bekertje wordt in de roslaunch-configuratie bepaald. | Could | Nee |
| DI03 | Locatie van de arm in de wereld wordt in de roslaunch-configuratie bepaald. | Could | Nee |

### Toelichting

Er is een demoscript aanwezig in de wor\_\_worlds\_simulatie directory genaamd demo.sh. Dit script publisht op de /robot\_command topic berichten in het format van msg\_srv/msg/RobotCommand naar de simulatie node (met de naam test\_node). Het programma handelt de commandos af en zorgt ervoor dat de juiste joint\_states worden gezet in TF2.

## A.5. GEBRUIKERSHANDLEIDING

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Omschrijving | Prio | Gerealiseerd |
| DM01 | Beschrijft hoe de code gebouwd kan worden. | Must | Ja |
| DM02 | Beschrijft stap voor stap hoe de arm bewogen kan worden middels enkele voorbeelden. | Must | Ja |
| DM03 | Beschrijft welke eisen gerealiseerd zijn. En geeft hierbij een (korte) toelichting. | Must | Ja |

### Toelichting

Dit document beschrijft hoe de code gebouwd kan worden met behulp van colcon build en geeft een aantal voorbeelden over hoe de robot wel en niet bewogen mag worden. Tevens is er in dit document ook beschreven welke eisen gerealiseerd zijn en zijn er toelichtingen aanwezig waarom dit zo is.

## A.6. ONTWERPDOCUMENTATIE

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Omschrijving | Prio | Gerealiseerd |
| DD01 | Beschrijft de structuur van de package (Nodes, topics, messages, et cetera). | Must | NEE | FIXEN |
| DD02 | Beschrijft de structuur en samenhang van de broncode (class-diagrams, beschrijving, et cetera). | Must | NEE | FIXEN |
| DD03 | Beschrijft hoe het gedrag van alle belangrijke componenten gerealiseerd is. | Could | Nee |
| DD04 | Beschrijft de API van alle publieke interfaces. | Should | Nee |

### Toelichting

