

Documentatie: Robotarm simulatie

Mart Rietdijk (1673342)

24 oktober 2023

Klas: ITN-WOR-A-s

Docent: Jorg Visch

Course: Wor World

Versie: 1.0

Contents

1	Inleiding	2
2	De Requirements	2
3	ROS-structuur	4
3.1	Console package	4
3.2	Simulation package	4
3.2.1	Robotarm	4
3.2.2	Cup	5
3.2.3	RViz	5
4	Klassendiagram	7
4.1	RobotArmJointPublisher	7
4.2	CupStatePublisher	7
4.3	PickedUpCupListener	7
4.4	Parser	7
5	Robot-commando's	10

1 Inleiding

Er zijn veel redenen om hardware in de daadwerkelijke wereld te simuleren. Daarom is deze simulatie opgezet om de meeste risico's van het werken met een robotarm af te vangen.

In dit document is te vinden hoe de simulatie-opdracht is uitgewerkt.

2 De Requirements

ID	Wat is er gedaan?	Prio	Klaar?
PA01	Alle code is gepackaged volgens de ROS-directorystructuur.	Should	✓
PA02	Package is te bouwen met colcon op ROS2 Humble Hawksbill	Must	✓
PA03	De applicatie wordt gebouwd met C++ volgens de Object Oriented principes die je geleerd hebt bij eerdere courses.	Must	✓
PA04		Should	×

Table 1: Requirements tabel

ID	Wat is er gedaan?	Prio	Klaar?
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	✓
VS02	De virtuele controller reageert op het topic (zie eis VS01) door bijbehorende joint_state messages te publiceren.	Must	✓
VS03	De virtuele robotarm wordt gevisualiseerd in Rviz (een URDF-model van de arm is beschikbaar op OnderwijsOnline).	Must*	✓
VS04	De virtuele robotarm gedraagt zich realistisch m.b.t. tijdgedrag (servo's roteren kost tijd en gaat geleidelijk).	Must	✓
VS05		Should	×

Table 2: Requirements tabel

ID	Wat is er gedaan?	Prio	Klaar?
VC01		Should	×
VC02	Publiceert een 3D-visualisatie van het bekertje voor Rviz.	Must*	✓
VC03		Should	×
VC04		Could*	×
VC05		Should	×
VC06	Het bekertje beweegt mee met de gripper (als hij vastgehouden wordt).	Must	✓
VC07	Bekertje is onderhevig aan zwaartekracht wanneer losgelaten.	Must	✓
VC08	Bekertje bepaalt en publiceert zijn positie.	Must	✓
VC09	Bekertje bepaalt en publiceert zijn snelheid.	Should	✓

Table 3: Requirements tabel

ID	Wat is er gedaan?	Prio	Klaar?
DI01	Een demoscript stuurt over de tijd een sequentie van commando's naar de armcontroller. 2	Must	✓
DI02		Could	×
DI03		Could	×

Table 4: Requirements tabel

ID	Wat is er gedaan?	Prio	Klaar?
DM01	Beschrijft hoe de code gebouwd kan worden.	Must	✓
DM02	Beschrijft stap voor stap hoe de arm bewogen kan worden middels enkele voorbeelden.	Must	✓
DM03	Beschrijft welke eisen gerealiseerd zijn. En geeft hierbij een (korte) toelichting.	Must	✓

Table 5: Requirements tabel

ID	Wat is er gedaan?	Prio	Klaar?
DD01	Beschrijft de structuur van de package (Nodes, topics, messages, et cetera).	Must	✓
DD02	Beschrijft de structuur en samenhang van de broncode (class-diagrams, beschrijving, et cetera).	Must	✓
DD03		Could	×
DD04		Should	×

Table 6: Requirements tabel

3 ROS-structuur

In ROS is een structuur te beschrijven in Topic, services, Nodes, en Actions. Deze structuur wordt in dit hoofdstuk besproken per ROS-package. De structuur is te vinden in het onderstaande plaatje (fig. 1).

3.1 Console package

De Console package published een Node “Console”. Deze Node published weer een Topic command. Deze Topic is van het type Command beschreven in de package robot_arm_interface. Deze Topic bevat een string. Deze wordt gevraagd door de Node “Console” en verstuurd via deze Topic naar de Node robot_arm_interface.

3.2 Simulation package

In De Simulation package worden de volgende Nodes gepublished (met een kleine beschrijving):

- robot_arm_publisher: Published alle joints van de robotarm
- robot_publisher: Is de standaard robot_state_publisher van ROS2
- cup_picked_up: Is een tf listener die kijkt of dat de arm de cup vast heeft
- cup_publisher: Published de cup frame

Deze Nodes bieden hun eigen Topics weer aan (met een kleine beschrijving):

- joint_states: De Topic die de status van de joints van de robotarm published
- picked_up_cup: De Topic die aangeeft of de cup wordt opgepakt door de robotarm
- robot_description: De Topic die de robotarm URDF aan RViz geeft
- cup_description: De Topic die de cup URDF aan RViz geeft

3.2.1 Robotarm

De robot_arm_publisher aanvaardt het commando op het Topic command, en leest dit uit met de parser. Het commando dat op command wordt gepublished kan een commando zijn die zegt op welke PWM waarde de servo moet staan en hoe lang hij hier over mag doen, of een commando om alle servo’s die bewegen te stoppen. De commando’s komen in hoofdstuk 5 aan bod.

In de launchfile wordt de URDF aan de `robot_arm_interface` meegegeven. Daaruit leest de Node de URDF in en published de joint states naar de Topic `joint_states`.

Deze joint states worden opgevangen door de `robot_publisher` Node. Deze Node verwerkt de states en published op basis daarvan de frames naar het tf framework.

3.2.2 Cup

Naast de robotarm staat er ook een bekertje (cup) in de wereld. Daarvoor zorgt de `cup_publisher` Node. Deze published een cup frame naar het tf framework. Deze cup frame heeft een afstand ten opzichte van de robotarm.

De `cup_publisher` luistert naar de Topic `picked_up_cup`. Op deze Topic wordt of de afstand van de cup naar de onderkant van de robotarm gestuurd, of de afstand van de cup ten opzichte van de hand. Deze keuze wordt gebaseerd op het feit dat de robotarm de cup vast heeft of niet. Dit is te meten aan de afstand van de grippers van de robotarm ten opzichte van de cup.

Als de robotarm de cup vast heeft, wordt de afstand van de hand gestuurd, en houdt de cup deze afstanden vast. En als de robotarm de cup niet vast heeft, valt de cup op de grond ten opzichte van de onderkant van de robotarm.

3.2.3 RViz

Op de Topics `cup_description` en `robot_description`, worden de URDFs van de robotarm en cup gepublished als string. Dit wordt gedaan zodat RViz weet bij welke joint wat te tekenen.

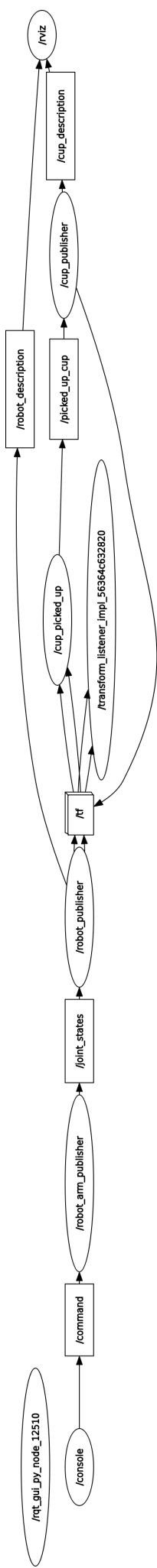


Figure 1: Volledige ROS-structuur

4 Klassendiagram

In dit hoofdstuk wordt het klassendiagram van de applicatie besproken. Het klassen diagram is te zien in afbeelding fig. 2. Hieronder volgt een beschrijving per klasse.

4.1 RobotArmJointPublisher

In deze klasse wordt de URDF van de robot-arm ingeladen. Van de URDF worden de joints opgeslagen in een map “joints”. Deze map bestaat uit de joint naam en de huidige positie van de joint. Nadat de joints worden aangemaakt worden deze om de 10 milliseconden gepublished. Dit wordt gedaan door de functie publishJointStates.

Wanneer een commando binnenkomt op de commandSub wordt de commandCallback methode aangeroepen. Daarna worden de PWM waarden van het commando omgerekend tot radialen. Daarna worden er stappen in radialen per milliseconden berekend. Deze stappen worden doorgegeven aan de sendJointToPos methode. Deze methode maakt een thread aan die voor het aantal milliseconden dat het commando aangeeft, de stappen bij de huidige positie worden opgeteld. Deze thread wordt opgeslagen in de movingThreads vector, zodat als er een stop commando wordt gegeven de thread nog gestopt kan worden.

Als er een commando “STOP” binnenkomt, worden de opgeslagen threads in de vector direct gestopt.

4.2 CupStatePublisher

In CupStatePublisher wordt eerst de URDF ingeladen met behulp van setupUrdf. Deze URDF wordt daarna op zijn tijd gepublished d.m.v. de methode publishCup.

De CupStatePublisher heeft ook een subscription op een Topic waar de positie ten opzichte van de hand of onderkant van de robotarm wordt geplublished. Op basis daarvan zorgt de methode updatePos of de positie van de cup beweegt of niet.

4.3 PickedUpCupListener

De PickedUpCupListener kijkt of de gripper de cup vast heeft ja of nee. Daar op basis geeft hij de afstand van de hand of de afstand van de onderkant van de robotarm. Dit doet hij in de functie timerCallback, die om de 100 milliseconden op de topic published.

4.4 Parser

Omdat er commando's in string formaat komen, in de vorm van een protocol van Lynxmotion SSC-32U USB Servo Controller Board (“lynxmotion_ssc-32u_usb_user_guide.pdf”, n.d.), moet er een Parser aan boord zijn van de RobotArmJointPublisher. Deze Parser maakt op basis van een statemachine bekend wat de servonr, pwm, en tijd is per bericht. Dit gebeurt door een Parser object aan te maken met een string. Direct in de constructor parsed hij het bericht. Door middel van de getters kunnen deze waarden opgehaald worden. Dit gebeurt dus ook per commando in RobotArmJointPublisher.

Naast een commando voor het bewegen van een servo, kan er ook een stop commando geparsed worden.

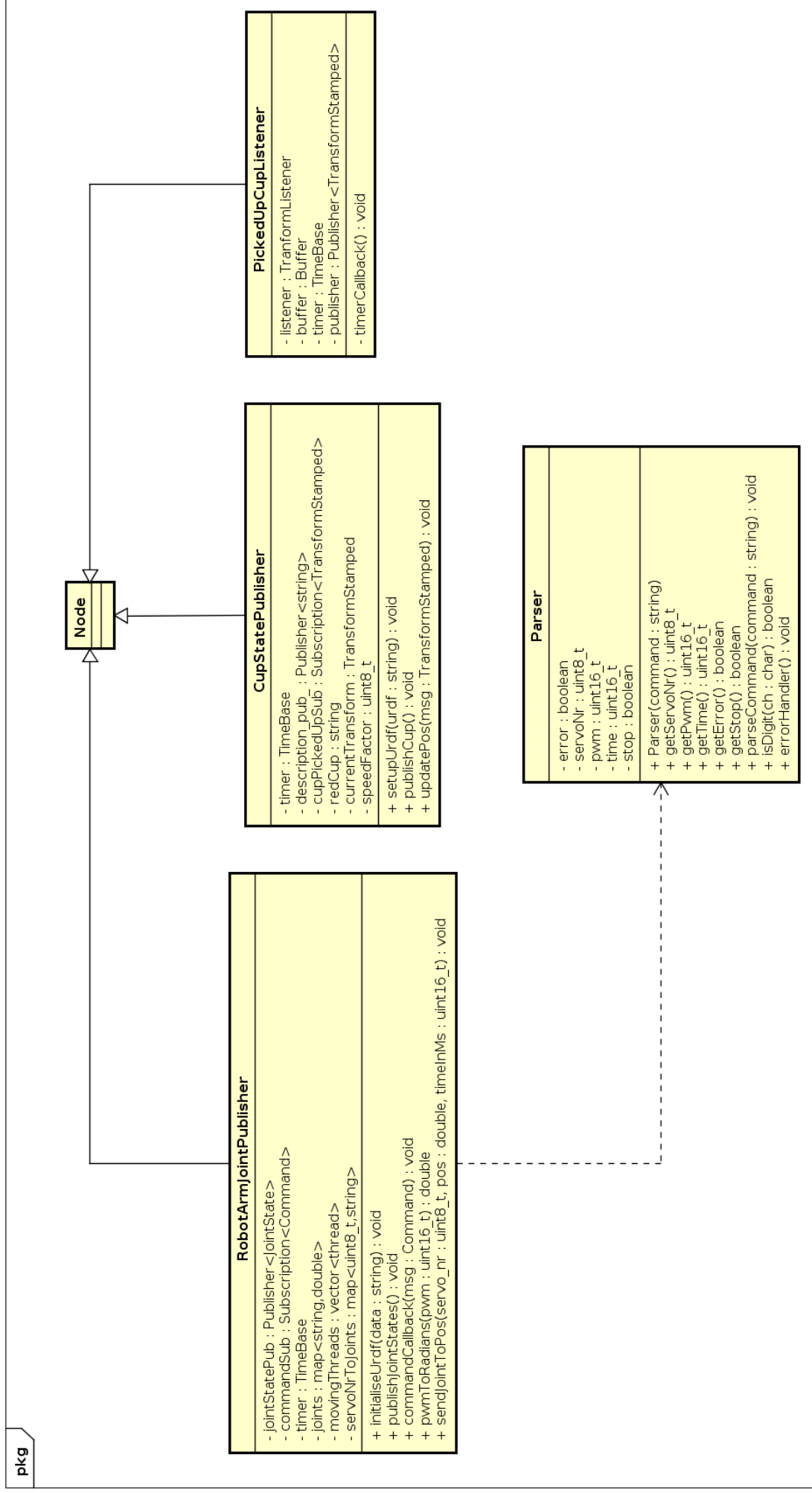


Figure 2: Klassendiagramm simulation

5 Robot-commando's

In het volgende tabel staan de commando's die gestuurd kunnen worden naar de robotarm. Deze commando's komen voort uit de SSC-32u protocol beschrijving (“lynxmotion_ssc-32u_usb_user_guide.pdf”, n.d.)

Commando	Beschrijving
$\# \langle servoNr \rangle P \langle pwm \rangle T \langle timeInMs \rangle \backslash r$	Beweegt een servo naar de aangegeven PWM. Dit duurt het aantal aangegeven milliseconden. De minimale waarde voor pwm is 1000 en maximale waarde is 2000.
STOP	Stopt alle bewegende servo's

Table 7: Commando's tabel

References

lynxmotion_ssc-32u_usb_user_guide.pdf [(Accessed on 10/26/2023)]. (n.d.).