

Robots - Simulatie

World of Robots



Auteur	Luke van Luijn	Docent	Chris van Uffelen
Student nummer	587478	Plaats	Nijmegen
Opleiding	HBO-ICT	Datum	30-10-2022
Profiel	Embedded Software Development (ESD)	Versie	1.0
Studiejaar	Jaar 3		

Inhoudsopgaven

- 1 [Ontwerp](#)
 - 1.1 [Packages](#)
 - 1.2 [Applicatie](#)
 - 1.2.1 [Cup node](#)
 - 1.2.2 [Arm node](#)
 - 1.2.3 [RobotArm](#)
- 2 [Requirements](#)

1. Ontwerp

In dit hoofdstuk zal dieper ingegaan worden op het ontwerp van de opdracht. Eerst zal de ros-structuur uitgelegd worden. Dit zal gebeuren aan de hand van een diagram die de samenhang tussen de verschillende nodes en topics weergeeft.

Vervolgens zal er dieper ingegaan worden op de samenhang van de broncode (source code). In dit onderdeel wordt aan de hand van verschillende diagrammen uitgelegd hoe de code in elkaar steekt.

1.1. Packages

De applicatie bestaat uit twee nodes; arm_node (weergegeven als: '*custom_arm_node*') en de cup_node (weergegeven als: '*custom_cup_node*').

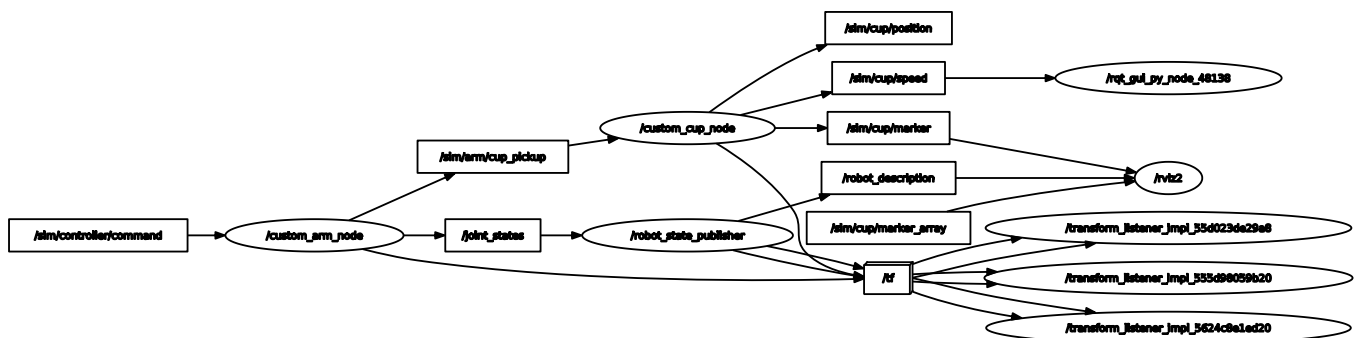


Diagram 1 - ROS nodes & topics

De arm_node luistert naar een topic genaamt */sim/controller/command*. Op dit topic kan de gebruiker van de applicatie en commando in de vorm van het AL5D protocol versturen. De arm_node reageert vervolgens op dit topic door joint_states te publiceren op het topic; */joint_states*.

De node; *robot_state_publisher* is een standaard node van ROS2 deze node luistert naar het eerder genoemde */joint_states* topic en genereerd op basis van die waardes een '*robot_discription*', deze wordt gepubliceerd op het topic */robot_description*.

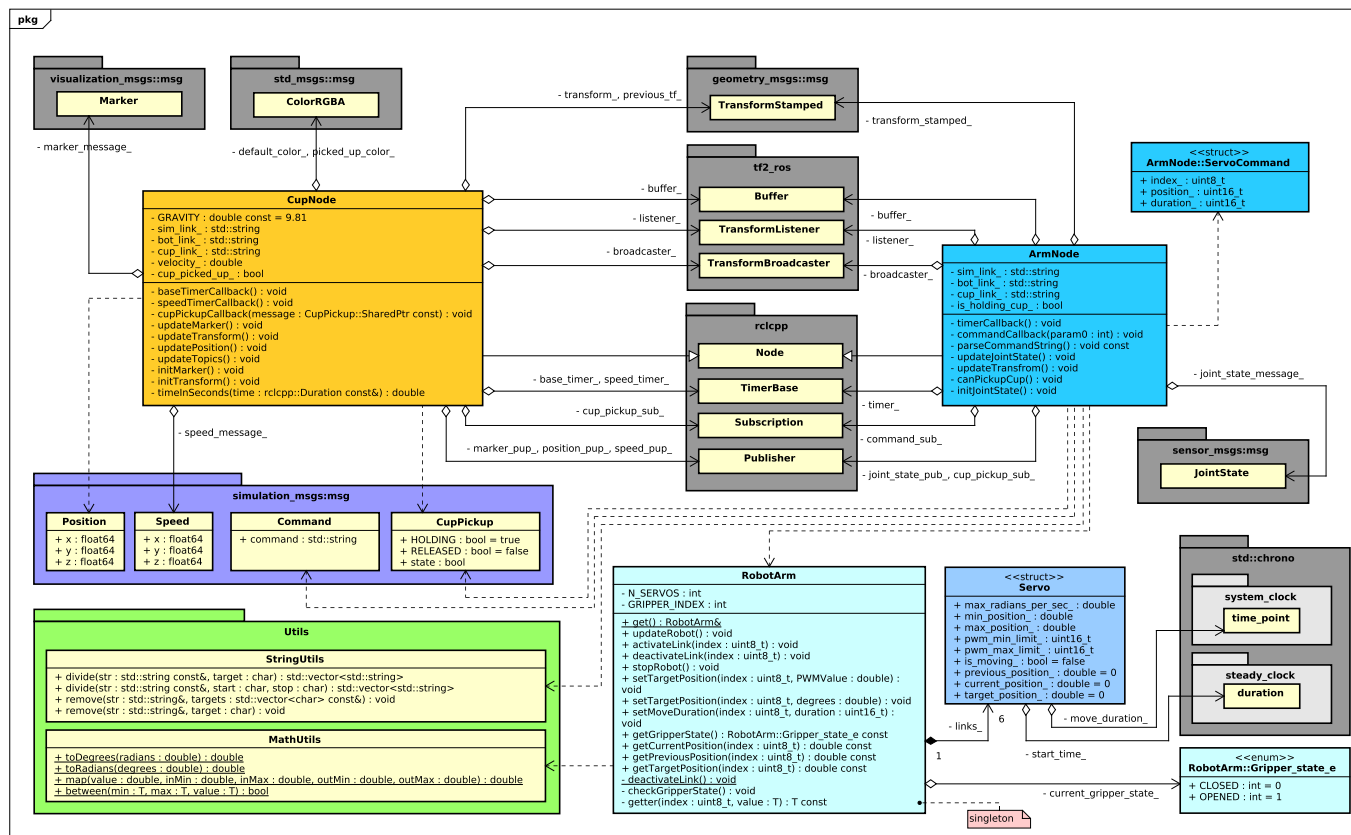
Zodra blijkt dat de arm het kopje aan het oppakken is zal de arm een publicatie doen naar het */sim/arm/cup_pickup* topic. Zodra de waarde van dit topic true is zal de cup_node weten dat deze opgepakt is door de robotarm en zal de positie van het kopje aanpassen op de positie van de gripper van de robotarm.

Verder is de cup_node verantwoordelijk voor het publiceren van drie onderdelen, ten eerste de positie; de cup_node bepaald en publiceert zijn positie naar het topic */sim/cup/position*. De gebruiker kan deze positie vervolgens uitlezen via de command line.

Het tweede onderdeel is de snelheid van het kopje. Het kopje bepaald zijn eigen snelheid ten opzichte van de wereld. De snelheid wordt gepubliceerd op het topic */sim/cup/speed*. Dit topic wordt vervolgens uitgelezen door RQT (weergegeven als: */rqt_gui_py_node_48138*) en de waardes geplot in de GUI.

Het derde en laatste onderdeel waar de cup_node verantwoordelijk voor is is het publiceren van een marker. De marker is een visuele weergaven van het daadwerkelijke kopje. De marker wordt naar het topic */sim/cup/marker* gepubliceert.

In dit onderdeel zal dieper ingegaan worden op de werking en samenhang van de applicatie. In de onderstaande afbeelding is het klasse diagram te zien van de gehele applicatie.



1.2.1. Cup node

De cup_node is verantwoordelijk voor het weergeven van de cup in Rviz. De weergaven van het kopje gebeurt aan de hand van een Marker ([visualization_msgs/msg/Marker](#)). Zodra de robotarm het kopje oppakt zal het kopje de locatie van de marker gelijkstellen aan de locatie van de 'hand' van de robot en op deze manier met de robotarm meebewegen. Wanneer de robotarm laat weten dat het kopje losgelaten is zal het kopje terug naar de grond vallen.

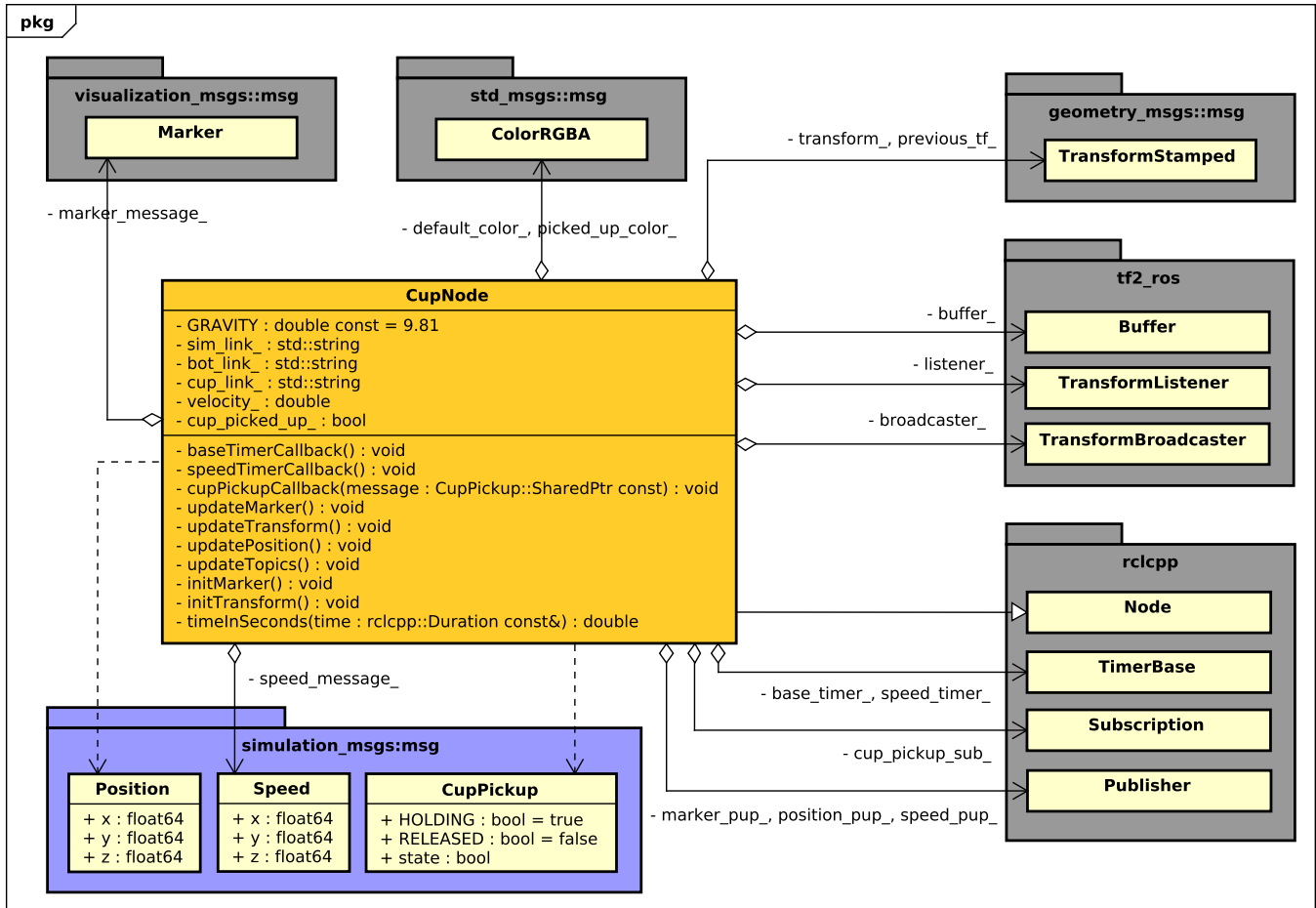


Diagram 3 - Class diagram: cup node

#	Naam	Beschrijving
01	GRAVITY	Const voor het berekenen van de neerwaartse snelheid van de cup.
02	default_color_	De kleur van het kopje wanneer deze niet vastgehouden wordt.
03	picked_up_color_	De kleur van het kopje wanneer deze vastgehouden wordt.
04	sim_link_	Naam van het frame gebruikt in de simulatie.
05	bot_link_	Naam van het frame van de ArmNode klasse.
06	cup_link_	Naam van het frame van de CupNode klasse.
07	velocity_	De huidige neerwaartse snelheid van het kopje.
08	cup_picked_up_	True wanneer het kopje is opgepakt, false wanneer het kopje niet vastgehouden wordt.
09	transform_	Huidige transform van het kopje.
10	previous_tf_	Voorgaande transform van het kopje, wordt gebruikt voor het berekenen van de 3D snelheid van het kopje.

#	Naam	Beschrijving
11	marker_message_	Het bericht dat gepubliceert wordt zodat het kopje weergegeven kan worden in Rviz.
12	speed_message_	Het bericht dat de snelheid van het kopje publiceert.
13	buffer_	Buffer die de verschillende transformaties bevat.
14	listener_	Luistert naar de verschillende transforms in de applicatie en plaatst deze in de buffer.
15	broadcaster_	Publiceert nieuwe transformaties.
16	base_timer_	Basis timer voor het updaten van de CupNode klasse (elke 10ms).
17	speed_timer_	Timer voor het publiceren van de snelheid van het kopje (elke 100ms).
18	marker_pup_	Publisher voor de marker berichten.
19	position_pup_	Publisher voor de position berichten.
20	speed_pup_	Publisher voor de speed berichten.
21	cup_pickup_sub_	Subscriber voor het CupPickup topic.

Tabel 1 - *CupNode: Members*

#	Naam	Beschrijving
01	baseTimerCallback	Wordt aangeroepen elke 10ms, update de verschillende topics.
02	speedTimerCallback	Wordt aangeroepen elke 100ms, berekend de huidige snelheid van het kopje en publiceert deze.
03	cupPickupCallback	Wordt aangeroepen wanneer de robotarm publiceert naar het cup_pickup topic. Als het kopje momenteel opgepakt is zal het meebewegen met de arm. Als het kopje losgelaten is zal het naar de grond vallen.
04	updateMarker	Update de variabelen van de marker_message_.
05	updateTransform	Update de variabelen van de transform_.
06	updatePosition	Update de variabelen van de position_message_.
07	updateTopics	Update de verschillende topics (zie bovenstaand).
08	initMarker	Initialiseert de marker_message_.
09	initTransform	Initialiseert de transform_.
10	timeInSeconds	converteert een rclcpp::duration naar tijd in seconden (double).

Tabel 2 - *CupNode: Methods*

1.2.2. Arm node

De arm node is verantwoordelijk voor het interpreteren van de door de gebruiker verstuurd commando's. Verder is deze klasse verantwoordelijk voor het publiceren van de huidige staat van de robotarm, deze informatie wordt gepubliceert door middel van JointStates ([sensor_msgs/msg/JointState](#)).

Zodra het commando voor het sluiten van de gripper is ontvangen zal de klasse checken of het toevallig op dezelfde locatie is met de gripper als het kopje. Als deze locaties hetzelfde zijn zal er een bericht gepubliceert worden naar het topic [/sim/arm/cup_pickup](#). Zodra de gripper weer opent zal er een bericht gestuurd worden naar hetzelfde topic.

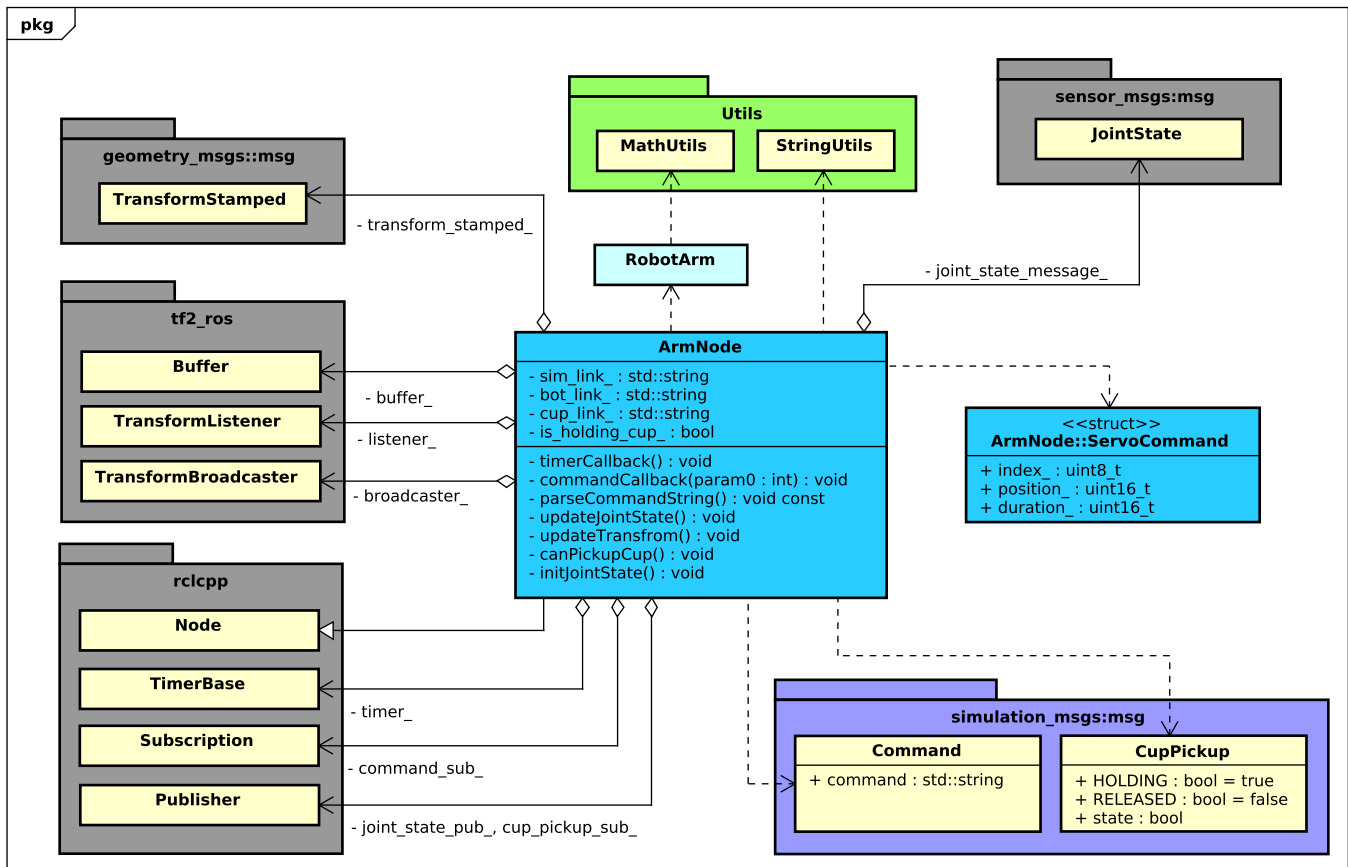


Diagram 4 - Class diagram: arm node

#	Naam	Beschrijving
01	transform_stamped_	Huidige transform van de arm.
02	buffer_	Buffer die de verschillende transformaties bevat.
03	listener_	Luistert naar de verschillende transforms in de applicatie en plaatst deze in de buffer.
04	broadcaster_	Publiceert nieuwe transformaties.
05	timer_	Basis timer voor het updaten van de ArmNode klasse (elke 10ms).
06	command_sub_	Subscriber voor het /sim/controller/command topic.
07	joint_state_pub_	Publisher JointState berichten.
08	cup_pickup_sub_	Subscriber voor het /sim/arm/cup_pickup topic.
09	joint_state_message_	Bericht dat de JointStates van de robotarm publiceert.

#	Naam	Beschrijving
10	sim_link_	Naam van het frame gebruikt in de simulatie.
11	bot_link_	Naam van het frame van de ArmNode klasse.
12	cup_link_	Naam van het frame van de CupNode klasse.
13	is_holding_cup_	True wanneer de robotarm het kopje vast heeft. False als de robotarm het kopje niet vast heeft.

Tabel 3 - *ArmNode: Members*

#	Naam	Beschrijving
01	timerCallback	Update de huidige transform, joint states, de robotarm en controleert of het kopje toevallig is opgepakt.
02	commandCallback	Wordt geactiveert zodra er een commando verstuurd is. Dit commando wordt geparsed en naar de robotarm verstuurd.
03	parseCommandString	Deze methode parsed het inkomende command bericht.
04	updateJointStates	Update de joint states op basis van de locatie van de robotarm.
05	updateTransform	Update het transform bericht.
06	canPickupCup	Checked of het kopje zich bevind tussen de gripper armen van de robotarm.
07	initJointState	Initialiseert het joint states bericht.

Tabel 4 - *ArmNode: Methods*

1.2.3. RobotArm

De RobotArm klasse is in essentie een driver voor een daadwerkelijke AL5D robotarm. De klasse simuleert de verschillende servo's en geeft verschillende mogelijkheden voor het besturen van deze servo's.

De RobotArm klasse maakt gebruik van het singleton pattern. In de 'echte' wereld zou deze klasse een directe link zijn met hardware en zou het dus ook niet mogelijk zijn om meerdere instanties van deze klasse te hebben. Een singleton pattern zorgt ervoor dat dit ook daadwerkelijk toegepast wordt.

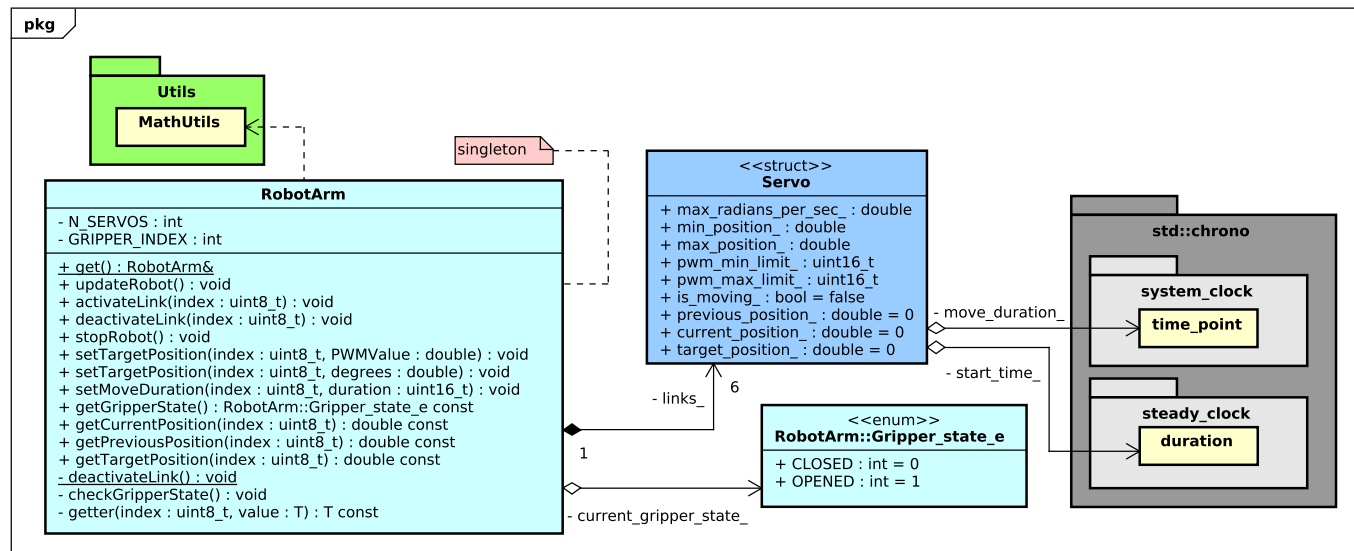


Diagram 5 - Class diagram: robotarm

#	Naam	Beschrijving
01	N_SERVOS	const waarde voor het aantal servo's in de robotarm.
02	GRIPPER_INDEX	const waarde voor de index van de servo gebruikt voor de gripper.
03	links_	Een vector met de verschillende servo's
04	current_gripper_state_	De huidige staat van de gripper (OPENEND / CLOSED)

Tabel 5 - ArmNode: Members

#	Naam	Beschrijving
01	get	Get instance van de RobotArm klasse.
02	updateRobot	Triggered een update van de verschillendes servo posities op basis van tijd.
03	activateLink	Start een link zodat deze geupdate zal worden.
04	deactiveLink	Stop een link zodat deze niet meer geupdate zal worden.
05	stopRobot	Stop alle links van de robot (noodstop).
06	setTargetPosition	Zet een target positie van een servo.
07	setMoveDuration	Zet de tijdsduur van een beweging van een servo.
08	getGripperState	retourneer de huidige gripper state.
09	getCurrentPosition	retourneer de huidige positie van een servo.
10	getPreviousPosition	retourneer de vorige positie van een servo.

#	Naam	Beschrijving
11	getTargetPosition	retourneer de target positie van een servo.
12	deactivateLink	Stop een link zodat deze niet meer geupdate zal worden.
13	checkGripperState	Check of de gripper momenteel gesloten of open is.
14	getter	Een algemene getter die eerst de servo index controleert en vervolgens de waarde retourneert.

Tabel 6 - *ArmNode: Methods*

2. Requirements

#	Prio	Behaald	Beschrijving
PA01	Should	✓	De directory structuur beschreven in de ROS2 tutorial (creating a package) is aangehouden tijdens de ontwikkeling van het project.
PA02	Must	✓	Het project is getest en geschreven met/voor de colcon build tool gebruikt door ROS2.
PA03	Must	✓	De verschillende classes en andere onderdelen van de code zijn geschreven aan de hand van in de OSM course geleerde OO principes.
PA04	Should	✓	De styleguide is voorafgaand aan de ontwikkeling van het project doorgenomen en toegepast tijdens. Verder is de .clang-format gebruikt voor het formateren van de code volgens de door ros bepaalde opmaak (.clang-format)
VS01	Must	✓	Zoals beschreven in de handleiding kan de virtuele controller aangestuurd worden door middel van seriele commando's opgezet volgens de in de lynxmotion beschreven handleiding. De ondersteunde commando's zijn: P, positie per servo, S, tijd per positie en stop, (STOP), voor het uitvoeren van een noodstop.
VS02	Must	✓	De joint_state berichten van de robotarm worden gepubliceerd op het topic: /joint_states. Deze berichten kunnen worden ingezien door middel van het commando: <code>ros2 topic echo /joint_states</code>
VS03	Must	✓	Door het commando voor het starten van de rviz applicatie (beschreven in de handleiding) uit te voeren is het model van de AL5D robot te zien.
VS04	Must	✓	De verschillende servo's hebben ieder een maximale snelheid meegekregen (gebaseerd op de datasheet per servo). De snelheid wordt verder nog aangepast op basis van de meegegeven tijd in milliseconden voor het uitvoeren van een commando.
VS05	Should	✓	In het launch document src/robot_simulation/launch/robot.launch.py zijn twee variabelen gedeclareerd (robot_pos_x & robot_pos_y) voor het bepalen van de positie van de robotarm. Deze variabelen kunnen aangepast worden waardoor de robot zal verplaatsen in rviz.
VC01	Should	✓	Zie VS05, de positie van de beker wordt bepaald op basis van de locatie van de robotarm.
VC02	Must	✓	Door middel van een marker wordt er een .stl document (src/robot_simulation/model/wor_sim_cup.stl) van een beker gepubliceerd in Rviz
VC03	Should	✗	Deze eis is niet gerealiseerd.
VC04	Could	✗	Deze eis is niet gerealiseerd.
VC05	Should	✓	Zodra de robotarm de beker vastpakt verandert de kleur van de beker van wit (255,255,255) naar cyan (0,255,255), verwijzing .
VC06	Must	✓	Zodra de robotarm de beker vastpakt verandert de locatie van de beker op basis van het middelpunt van de twee gripper armen, verwijzing .
VC07	Must	✓	Zodra de beker niet vastgehouden word door de robotarm en de beker niet op de grond staat zal er zwaartekracht toegepast worden zodat de beker naar de grond toe valt, verwijzing .

#	Prio	Behaald	Beschrijving
VC08	Must	✓	De virtuele beker publiceert zijn positie naar een topic: <code>/sim/cup/pose</code> . Dit topic kan uitgelezen worden door middel van het commando: <code>ros2 topic echo /sim/cup/pose</code> .
VC09	Should	✓	De virtuele beker publiceert zijn snelheid naar een topic: <code>/sim/cup/speed</code> . Dit topic kan uitgelezen worden door middel van het commando: <code>ros2 topic echo /sim/cup/speed</code> .
VC10	Could	✓	De actuele snelheid van de beker (hetzelfde als het topic) kan getoond worden in <code>rqt_plot</code> door middel van het volgende commando: <code>rqt --perspective-file ~/wor_sim_review/src/robot_simulation/config/rqt_config.perspective</code>
DI01	Must	✓	Er is een demo script opgesteld voor het demonstreren van de verschillende capaciteiten van de applicatie demo.sh
DI02	Could	✓	Zoals verteld bij eis VS05 is dit onderdeel opgenomen in het launch document.
DI03	Could	✓	Zoals verteld bij eis VC01 is dit onderdeel opgenomen in het launch document.
DM01	Must	✓	Zie hoofdstuk: Handleiding/Installatie instructies.
DM02	Must	✓	Zie hoofdstuk: Handleiding/Bewegen van de robot.
DM03	Must	✓	Zie hoofdstuk: Handleiding/Requirements.
DD01	Must	✓	Zie hoofdstuk: Ontwerp/Packages.
DD02	Must	✓	Zie hoofdstuk: Ontwerp/Applicatie.
DD03	Could	✗	Deze eis is niet gerealiseerd.
DD04	Should	✓	Zie hoofdstuk: Ontwerp/Applicatie.

Tabel 7 - Requirements