Ontwerpdocumentatie

Agit Tunc (1527782)

April 2025

Docent: Bram Knippenberg
Course: Wor World
Versie: 1.0

Contents

1 Introductie

In dit document wordt uitgelegd hoe de simulatiepackage is opgebouwd, welke componenten zijn ontwikkeld, hoe deze samenwerken en wat hun functie is binnen het geheel. De simulatie bootst een robotarm na die een kopje (de *cup*) kan oppakken en verplaatsen.

2 Architectuur

De simulatie bestaat uit twee zelf ontwikkelde nodes: `virtual_servo_controller_node` en `cup_node`, en maakt daarnaast gebruik van bestaande ROS 2-nodes zoals `robot_state_publisher` en `rviz2`.

De interactie tussen deze nodes zorgt voor een simulatie waarin een robotarm servo-commando's ontvangt, joint-posities publiceert en een visuele representatie toont van een arm die een kopje kan oppakken.

Figuur 1 toont een overzicht van de actieve ROS-nodes en de communicatie tussen deze nodes, verkregen via de `rqt_graph` tool.

2.1 `virtual_servo_controller_node`

Deze node ontvangt commando's via het topic `/ssc32u_command`. Deze commando's zijn gebaseerd op het SSC32U-protocol en kunnen bestaan uit:

- Servo "move" commando's (met tijd en/of snelheid)
- Een status-query (is de arm in beweging?)
- Een commando om alle servos te stoppen
- Een commando om *een* specifieke servo te stoppen

In plaats van een serviceinterface is bewust gekozen voor communicatie via topics, omdat dit een eis van de opdracht was.

De node berekent hoe elke servo over tijd beweegt richting zijn doelpositie. Elke 20ms wordt de voortgang gepublisht via het topic `/joint_states`. Dit is afgestemd op de realistische updatefrequentie van echte servo's. Eventuele reacties op commando's (zoals bij status-queries) worden gepubliceerd op `/ssc32u_response`.

2.2 `robot_state_publisher`

Deze standaard ROS 2-node ontvangt `joint_states` en gebruikt het meegegeven URDF-model (via `/robot_description`) om de bijbehorende TF-transformaties te berekenen. Hierdoor worden alle posities van armsegmenten automatisch geactualiseerd bij het bewegen van servos.

2.3 `cup_node`

De `cup_node` is verantwoordelijk voor het beheren en visualiseren van een virtuele beker in de wereld. In tegenstelling tot sommige implementaties die observatie en publicatie splitsen in aparte nodes, bevat deze node zowel de logica om te detecteren of de beker wordt vastgehouden, als de logica om de positie en het uiterlijk van de beker aan te passen. Alles is dus gebundeld in één enkele node.

De node:

- Publiceert een TF-transform van de beker elke 10 ms, zodat de locatie bekend blijft binnen het TF-netwerk
- Luistert naar het topic `/picked_up_cup` en past de positie van de beker aan op basis van of deze wordt opgepakt
- Detecteert via TF of de beker zich tussen beide grijpers bevindt en dus wordt vastgehouden
- Verandert de kleur van het bijbehorende URDF-model naar groen (opgepakt) of rood (niet opgepakt)
- Publiceert het URDF-model via `/cup_description` zodat RViz weet wat en waar de beker is

Als de beker wordt opgepakt, wordt het TF-frame van de `cup` gekoppeld aan het `hand`-frame van de robotarm. Zodra de beker wordt losgelaten, valt deze naar beneden en komt uiteindelijk tot stilstand ten opzichte van het `base_link`-frame. Dit gedrag simuleert het oppakken en neerzetten van een object op realistische wijze.

2.4 `rviz2`

RViz wordt gebruikt om de robotarm en het kopje visueel te tonen. De bijbehorende URDF-modellen worden aangeleverd via de topics `/robot_description` en `/cup_description`. De visualisatie toont real-time de positie van de arm en beker, inclusief TF-frames.

3 TF-structuur en bekerstatus

De positie van de beker wordt beheerd via TF-transformaties. Er zijn twee situaties:

- **Niet opgepakt:** De beker heeft zijn eigen positie onder `base_link`
- **Opgepakt:** De beker volgt de handpositie (TF van `hand` naar `cup`)

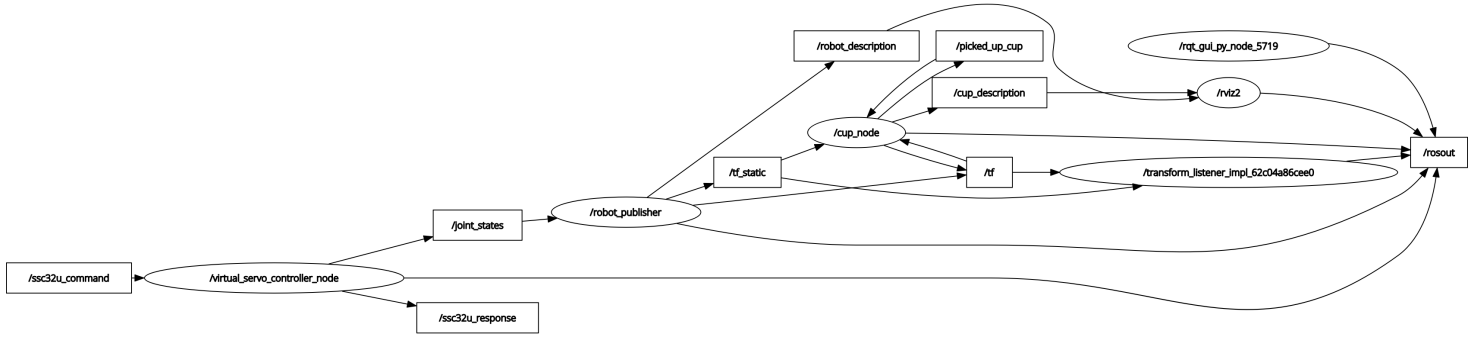

Figure 1: ROS-graph: overzicht van nodes en topics in de simulatie

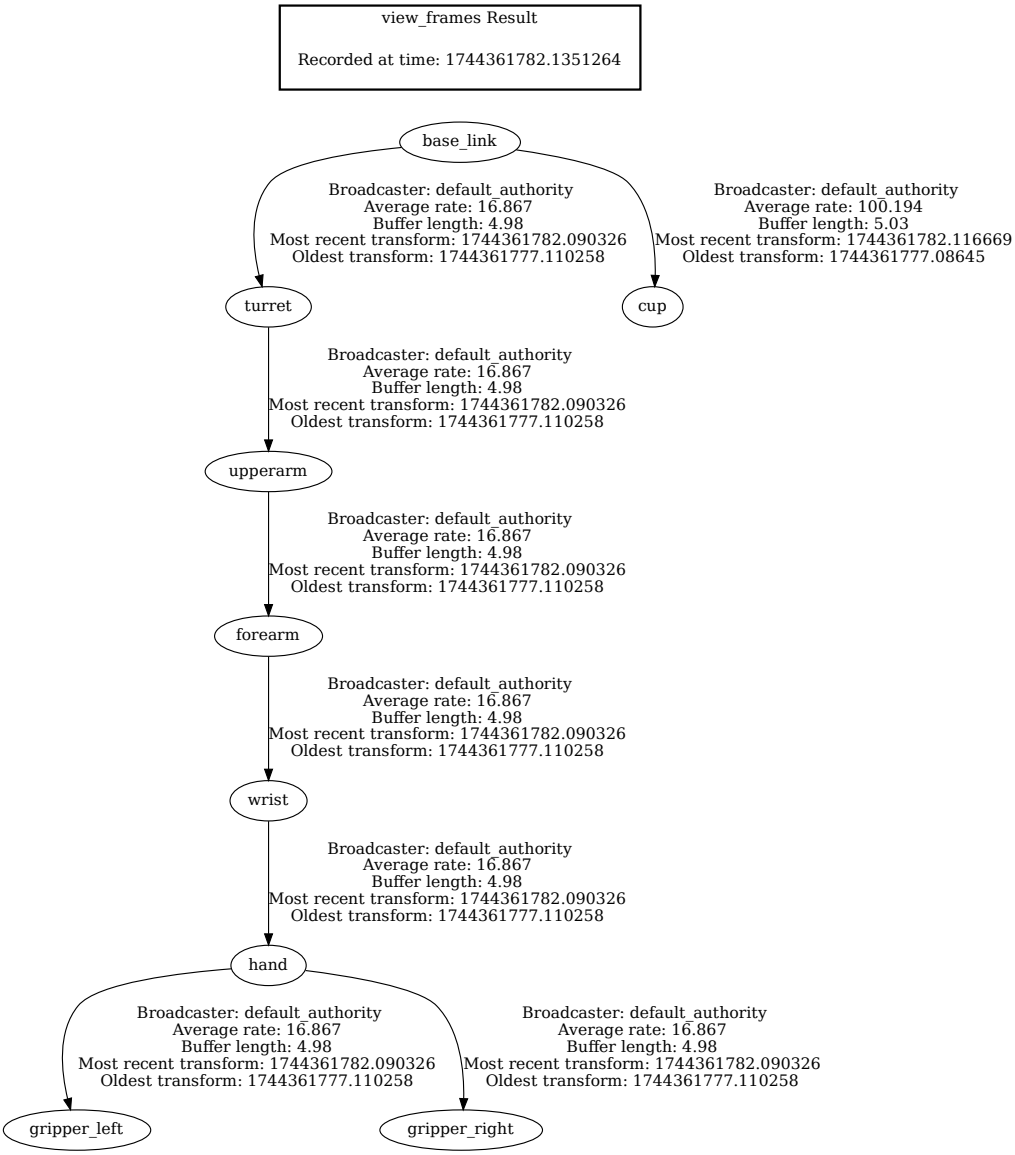

Figure 2: TF-structuur wanneer de beker niet is opgepakt

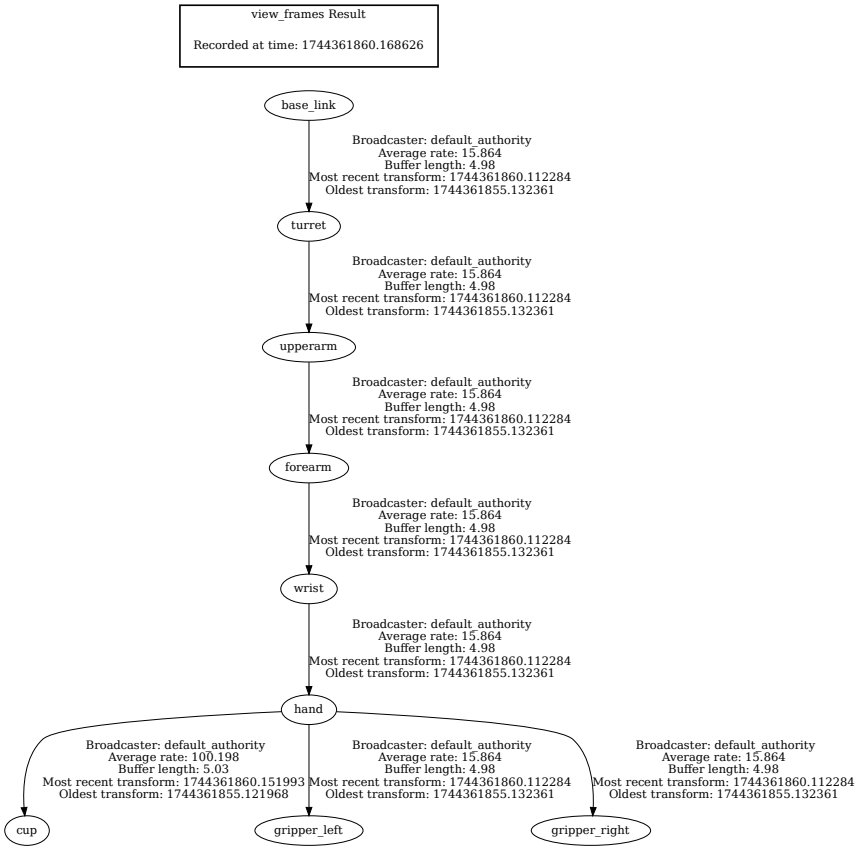

Figure 3: TF-structuur wanneer de beker is opgepakt

4 Structuur en gedrag van de broncode

In dit hoofdstuk worden de klassediagrammen besproken die de structuur van de belangrijkste componenten illustreren. Deze diagrammen geven inzicht in de gebruikte klassen, attributen, methoden en de onderlinge samenhang.

4.1 `virtual_servo_controller`

De onderstaande diagram toont de structuur van de node die verantwoordelijk is voor het interpreteren en uitvoeren van SSC32U-servo commando's. De klasse `VirtualServoController` erft van `rclcpp::Node` en bevat methoden voor het verwerken van inkomende berichten, het berekenen van servo-bewegingen, en het publiceren van actuele joint-posities.

De diagram toont ook de koppeling met de `SSC32UCommandParser`, die inkomende

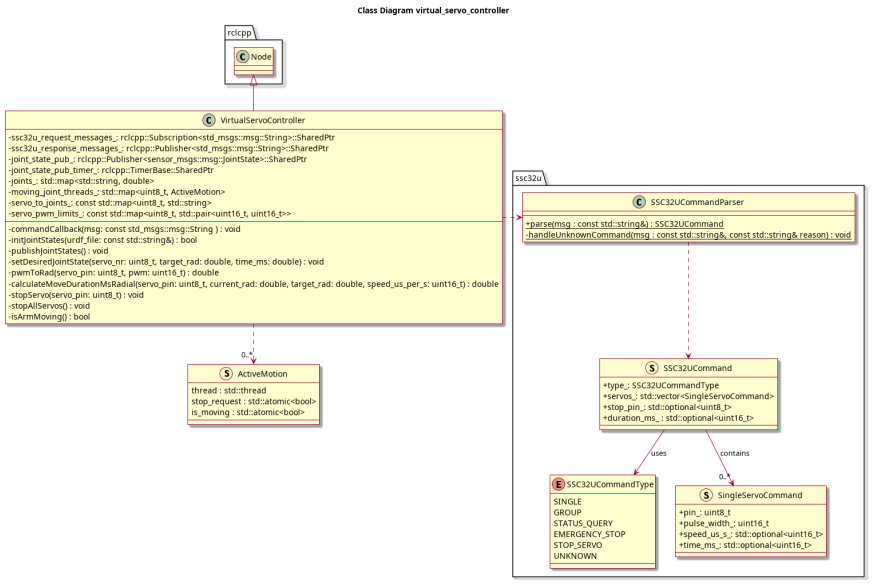

Figure 4: Klassediagram van de `virtual_servo_controller` node

string-commando's verwerkt tot gestructureerde commando-objecten, en de bijbehorende datastructuren zoals `SSC32UCommand` en `SingleServoCommand`. De klasse `ActiveMotion` wordt per servo gebruikt om een aparte thread te beheren voor beweging.

4.2 `cup`

De volgende diagram toont de `Cup`-klasse. Deze node is verantwoordelijk voor het dynamisch aanpassen van de positie en weergave van het kopje in de simulatie. Ook deze klasse erft van `rclcpp::Node` en combineert meerdere verantwoordelijkheden in één component: TF-publicatie, statusdetectie en URDF-manipulatie.

Belangrijke functies zijn onder andere:

- `publishCup()` — publiceert een `TransformStamped` TF-frame
- `updatePos()` — past de positie aan op basis van input
- `cupIsHeldCb()` — controleert via TF of de beker wordt vastgehouden
- `updateColorInUrdf()` — past de kleur aan in de URDF-beschrijving

5 API beschrijving

Voor een gedetailleerde API-beschrijving van de zelf ontwikkelde nodes wordt verwezen naar de README.md van het project.

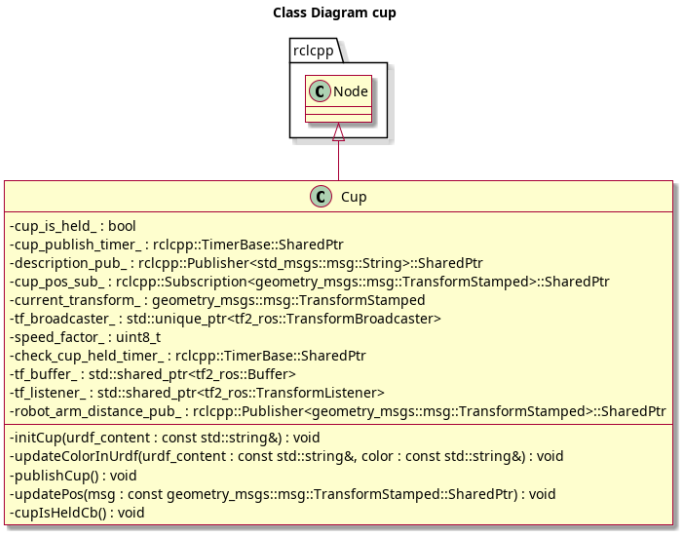

Figure 5: Klassediagram van de `cup` node