|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ontwerpdocumentatie\_  WoR WORLDS Simulatie   |  |  |  |  | | --- | --- | --- | --- | | **Auteur** | Mike Ackerschott | **Docent** | Jorg Visch | | **Student nr** | 1652342 | **Plaats** | Nijmegen | | **Opleiding** | HBO-ICT | **Datum** | 26-10-2023 | | **Profiel** | Embedded Software Development | **Versie** | 1.0 | | **Studiejaar** | Jaar 3 |  |  | |

26 oktober 2023

INHOUDSOPGAVE

[1 Ros Structuur 3](#_Toc149233228)

[1.1 Nodes 3](#_Toc149233229)

[1.2 Topics 3](#_Toc149233230)

[1.3 Services 3](#_Toc149233231)

[2 Broncode structuur 3](#_Toc149233232)

# Ontwerp simulatie

In dit hoofdstuk zal er in meer detail mijn uitwerking van de opdracht beschreven worden. Eerst zal de structuur en samenhang van alle ROS-onderdelen uitgelegd worden aan de hand van een diagram en extra toelichting.

Verder zal dit document ook nog dieper ingaan op de code zelf aan de hand van een klasse diagram.

## ROS-structuur

Het programma bestaat in totaal uit drie verschillende nodes, namelijk /communicator\_node, /custom\_arm\_node en /custom\_cup\_node.

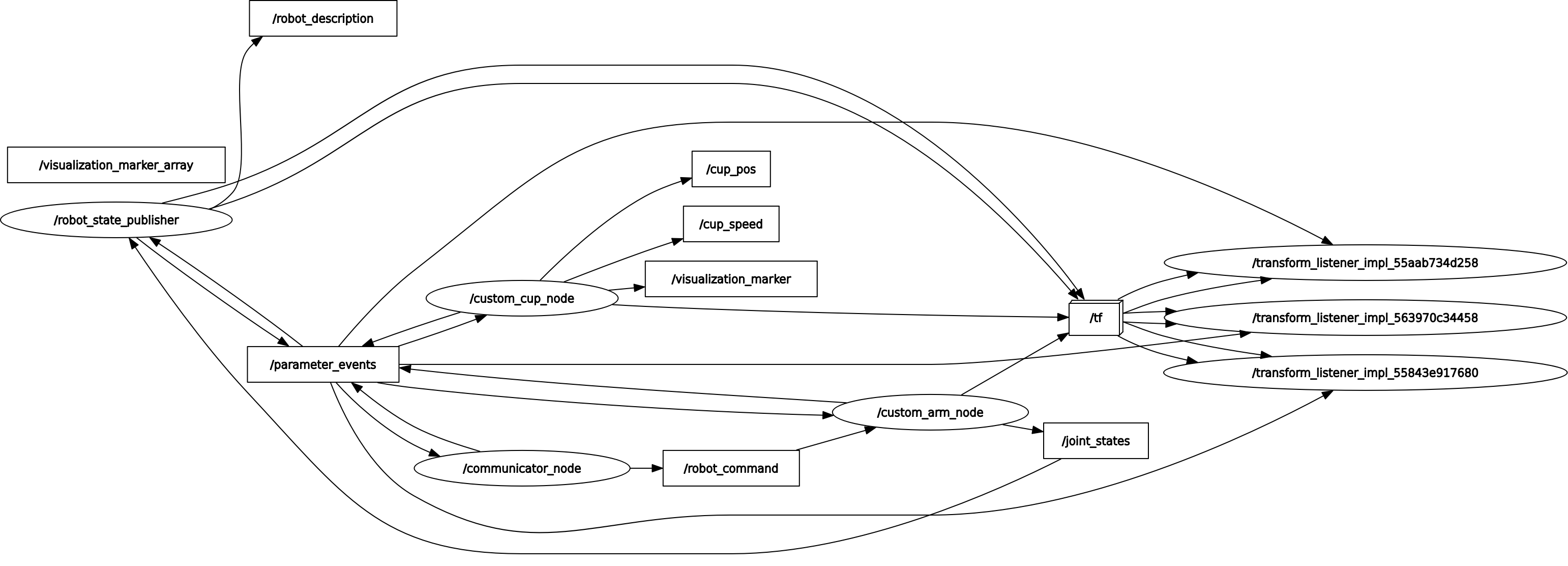
Het bewegen van de robotarm start bij de /communicator\_node. Dit is een node die een programma draait waar er wordt gewacht op user input via een commandline interface. Als dit ontvangen is, wordt de input over de topic /robot\_command als string verstuurt naar de node /custom\_arm\_node. Hier wordt eerst geverifieerd of het commando in het formaat van het AL5D protocol is. Als dat niet zo is wordt het commando genegeerd.

Diagram 1: ROS structuur

Als het commando is geverifieerd, wordt er een beweging geplant gebaseerd op de huidige positie van de robotarm en de meegegeven waarden. Als het plannen klaar is, wordt er elk 10 milliseconde voor de algehele duratie van de beweging een nieuwe positie gepost via de /joint\_states topic naar de node /robot\_state\_publisher. Hierdoor beweegt de robotarm geleidelijk.

Wanneer de robotarm zijn gripper volledig dicht is na het uitvoeren van een beweging, wordt er vanuit de /custom\_arm\_node node een client request gedaan naar de /pickup\_cup service op de /custom\_cup\_node node. Hierbij wordt in de request de variabele pickup op true gezet. De /custom\_cup\_node node checkt dan of de hand van de robotarm dicht bij de beker is. Als dat zo is, wordt de beker opgepakt en bindt de beker zich aan de hand van de robotarm via TF2. De service antwoord dan met een pickup\_success = true als het bekertje is opgepakt.

Wanneer een beker is opgepakt en de gripper volledig wordt geopend, wordt er vanuit de /custom\_arm\_node over dezelfde service (namelijk /pickup\_cup) een request gestuurd, waarbij de variabele pickup op false wordt gezet. Dit ontvangt de service op de /custom\_cup\_node node en weet dan dat het bekertje zich moet losbinden van de hand van de robotarm en zwaartekracht toegepast moet worden. Hierbij bindt het bekertje via TF2 zich weer aan de sim\_link frame, wat betekend dat het op dezelfde plek in de wereld blijft. Ten alle tijden is de /custom\_cup\_node node op de topics /cup\_pos en /cup\_speed zijn positie en snelheid op dat moment aan het posten. Hierdoor kun je bijhouden waar de beker is en hoe de beker bewoog.

## Node interfaces

Om het duidelijk te maken over hoe er gecommuniceerd kan worden met behulp van dit programma, is hieronder voor elke node de topics, services en actions gedocumenteerd en welke berichttypes ze gebruiken

### /communicator\_node

**Topics (Publishers):**

1. Topicnaam: "robot\_command"
   * Type: msg\_srv::msg::RobotCommand
   * Naam in sourcecode: robotCommandPub\_
   * Beschrijving: Dit topic wordt gepubliceerd om opdrachten naar de hoog-niveau driver te sturen. Een commando ziet er als volgt uit: #0 P1500 #1 P1500 #2 P1500 T3000

**Services (N/A):**

### /custom\_arm\_node

**Topics (Publishers):**

1. Topicnaam: "joint\_states"
   * Type: sensor\_msgs::msg::JointState
   * Naam in sourcecode: joint\_state\_pub\_
   * Beschrijving: Dit topic wordt gepubliceerd om de toestand van de gewrichten ("joint states") van de robot door te geven.

**Topics (Subscribers):**

1. Onderwerp: "robot\_command"
   * Type: msg\_srv::msg::RobotCommand
   * Naam in sourcecode: robot\_command\_sub\_
   * Beschrijving: Deze subscriber ontvangt opdrachten ("robot command") van externe bronnen om acties op de robot uit te voeren.

**Services (Clients):**

1. Servicenaam: "pickup\_cup"
   * Type: msg\_srv::srv::PickupCup
   * Naam in sourcecode: pickupCupClient\_
   * Beschrijving: Deze serviceclient wordt gebruikt om een verzoek te sturen om een beker op te pakken of los te laten, afhankelijk van de meegegeven parameters.

### /custom\_cup\_node

**Topics (Publishers):**

1. Topicnaam: "visualization\_marker"
   * Type: visualization\_msgs::msg::Marker
   * Naam in sourcecode: markerPub
   * Beschrijving: Dit topic wordt gepubliceerd om markerinformatie (waaronder het 3D .stl bestand) voor de "cup" naar visualisatietools zoals RViz te sturen.
2. Topicnaam: "cup\_pos"
   * Type: msg\_srv::msg::Pos
   * sourcecode: posPub
   * Beschrijving: Dit topic wordt gepubliceerd om de positie van de "cup" te verzenden.
3. Topicnaam: "cup\_speed"
   * Type: msg\_srv::msg::Pos
   * sourcecode: speedPub
   * Beschrijving: Dit topic wordt gepubliceerd om de snelheid van de "cup" te verzenden.

**Services (Server):**

1. Service: "pickup\_cup"
   * Type: msg\_srv::srv::PickupCup
   * sourcecode: pickupCupService
   * Beschrijving: Dit is een service die kan worden aangeroepen om een "pickup" -actie uit te voeren voor de "cup." Het wordt gebruikt om de "cup" op te tillen of neer te zetten, afhankelijk van het verzoek.

# Broncode structuurCommunicatorNode

Om de structuur van het programma beter te begrijpen, is hieronder een klassediagram aanwezig (ook te openen als afbeelding of .asta in de docs directory. Ik raad dit sterk aan).

## CommunicatorNode

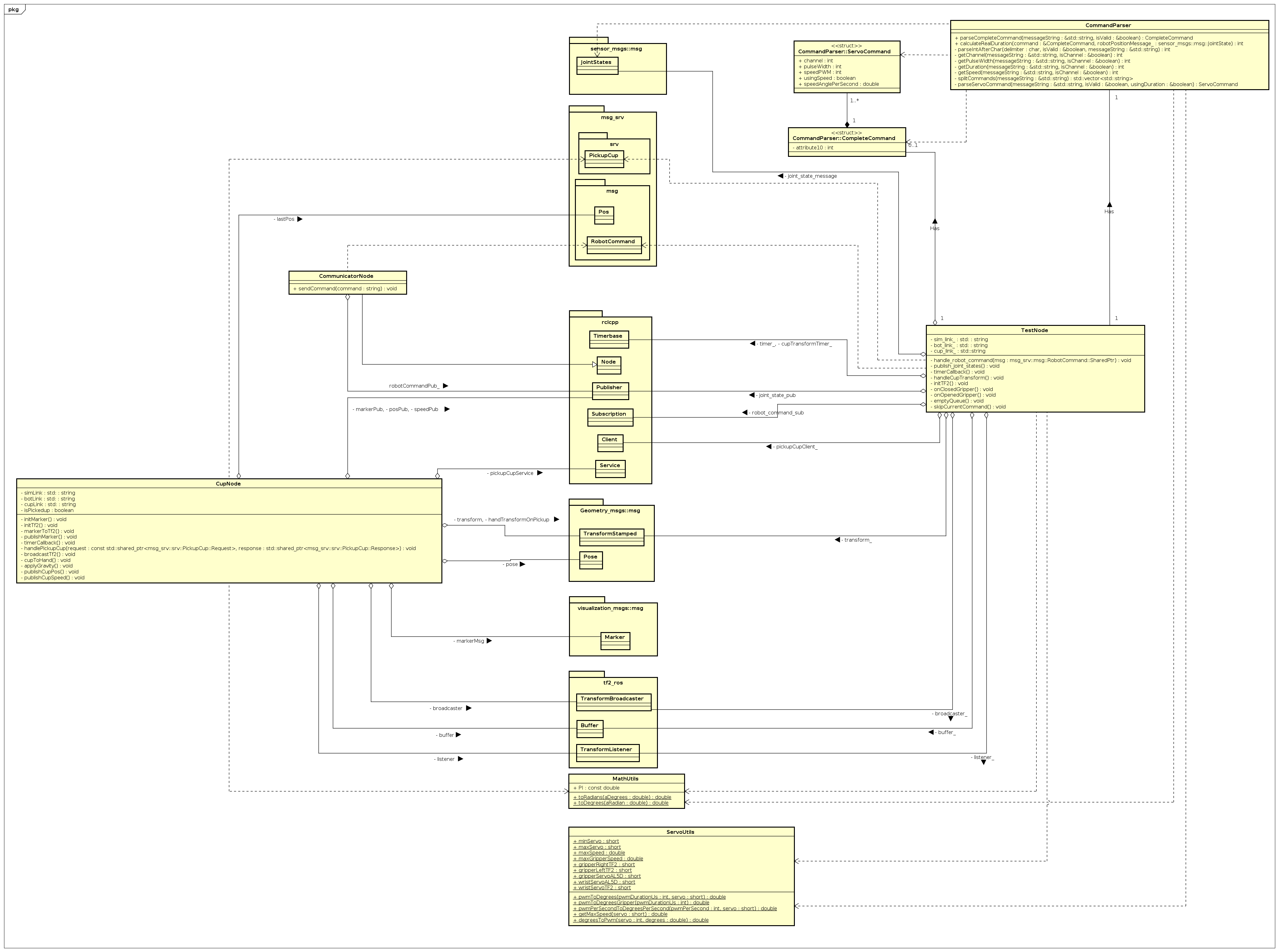


Diagram 2: Klasse diagram van de broncode (ook te zien in de folder docs)

**Membervariabelen:**

1. robotCommandPub\_: Een rclcpp::Publisher voor het publiceren van berichten van het type msg\_srv::msg::RobotCommand. Het wordt gebruikt om opdrachten naar een robotstuurprogramma te sturen.

**Memberfuncties:**

1. CommunicatorNode::CommunicatorNode(): De constructor van de klasse. Het initialiseert de CommunicatorNode en creëert de robotCommandPub\_ Publisher voor het verzenden van opdrachten. Het stelt ook het logniveau in op DEBUG.
2. CommunicatorNode::~CommunicatorNode(): De destructor van de klasse. Het heeft geen specifieke implementatie in dit geval.
3. void CommunicatorNode::sendCommand(std::string command): Een functie waarmee je een opdracht als een string kunt doorgeven. Deze functie maakt een bericht van het type msg\_srv::msg::RobotCommand, vult het met de opgegeven opdracht en publiceert het via robotCommandPub\_. Dit wordt gebruikt om opdrachten naar de robot te sturen.

## CupNode

**Membervariabelen:**

1. simLink, botLink, cupLink: Strings die de namen van verschillende frames definiëren, zoals sim\_link\_, bot\_link\_, en cup\_link\_. Deze frames worden gebruikt in tf2-transformaties.
2. markerPub: Een rclcpp::Publisher voor het publiceren van markers van het type visualization\_msgs::msg::Marker. Gebruikt voor het weergeven van het bekertje met behulp van een .stl bestand.
3. posPub: Een rclcpp::Publisher voor het publiceren van de positie van de beker als een bericht van het type msg\_srv::msg::Pos.
4. pickupCupService: Een rclcpp::Service waarmee je de beker op kunt pakken of los kan laten, gebaseerd op de meegegeven parameters.
5. markerMsg: Een instantie van visualization\_msgs::msg::Marker voor het opslaan van markergegevens van de gevisualizeerde beker.
6. pose: Een instantie van geometry\_msgs::msg::Pose die wordt gebruikt voor het definiëren van de positie en oriëntatie van de gevisualizeerde beker.
7. timer: Een timer die periodiek de timerCallback-functie aanroept.
8. transform: Een instantie van geometry\_msgs::msg::TransformStamped die wordt gebruikt om de transform tussen de beker en de robotarm of wereld te onthouden.
9. buffer, listener, broadcaster: Instanties van respectievelijk tf2\_ros::Buffer, tf2\_ros::TransformListener, en tf2\_ros::TransformBroadcaster, die worden gebruikt voor het publiceren, bufferen en opvangen van TF2 berichten.
10. handTransformOnPickup: Een instantie van geometry\_msgs::msg::TransformStamped die de transformatie van de beker naar de hand opslaat wanneer de beker is opgepakt.l
11. isPickedup: Een bool die aangeeft of de beker is opgepakt.
12. lastPos: Een instantie van het berichtstype msg\_srv::msg::Pos die wordt gebruikt om de laatst bekende positie van de beker op te slaan. Dit wordt vergeleken met de nieuwe opgehaalde positie om de snelheid te bepalen.

**Memberfuncties:**

1. CupNode::CupNode(): De constructor van de klasse. Het initialiseert de CupNode en voert enkele initialisaties uit, zoals het opzetten van markers, timers en tf2-transformaties.
2. void CupNode::initMarker(): Initialiseert de marker voor de beker, inclusief het inladen van het .stl bestand, kleur en positie.
3. void CupNode::initTf2(): Initialiseert de tf2-transformatie tussen frames.
4. void CupNode::publishMarker(): Publiceert de visualisatie van de beker.
5. void CupNode::timerCallback(): Wordt opgeroepen door de timer en voert periodieke taken uit, zoals het toepassen van zwaartekracht op de beker, het publiceren de positie van de beker, het publiceren van de snelheid van de beker, het updaten van de marker van de beker en het uitzenden van tf2-transformatie tussen de beker en de wereld of robotarm.
6. void CupNode::handlePickupCup(const std::shared\_ptr<msg\_srv::srv::PickupCup::Request> request, const std::shared\_ptr<msg\_srv::srv::PickupCup::Response> response):   
   Behandelt het serviceverzoek voor het oppakken van de beker, waarbij wordt gecontroleerd of de beker binnen bereik is om op te pakken. Wanneer de beker niet opgepakt wordt, omdat de robotarm te ver weg is, wordt response→pickup\_success op false gezet.  
   Als request→pickup op true is gezet, wordt er geprobeerd om de beker op te pakken.  
   Als request→pickup op false is gezet, wordt de beker los gelaten.
7. void CupNode::broadcastTf2(): Publiceert de tf2-transformatie tussen de beker en de robotarm of wereld. Wanneer de beker is opgepakt zal dit tussen de beker en robotarm zijn, anders zal het tussen de beker en de wereld zijn.
8. void CupNode::cupToHand(): Zorgt ervoor dat de transform tussen beker en wereld verandert naar beker en hand. Hierdoor blijft de beker meebewegen met de robotarm.
9. void CupNode::applyGravity(): Past zwaartekracht toe op de beker als deze niet is opgepakt en niet al op de grond ligt. Hierdoor valt de beker met een matige snelheid naar beneden
10. void CupNode::publishCupPos(): Publiceert de positie van de beker in X, Y, Z formaat
11. void CupNode::publishCupSpeed(): Publiceert de snelheid van de beker op basis van het verschil tussen de huidige positie en de vorige opgeslagen positie van de beker. Dit gebeurt door gebruik te maken van de informatie uit de lastPos variabele en de huidige positie van de beker. De snelheid wordt gepubliceerd als een bericht van het type msg\_srv::msg::Pos op het cup\_speed-onderwerp.

## CommandParser

**Membervariabelen:**

1. struct ServoCommand: Een struct die informatie bevat over een beweging van een enkele servo, inclusief het kanaal, pulsduur, snelheid PWM en of snelheid wordt gebruikt.
2. struct CompleteCommand: Een struct die informatie bevat over een volledige opdracht, inclusief een vector van servo-opdrachten, de duur van de opdracht en of de duur wordt gebruikt.

**Memberfuncties:**

1. CommandParser::CommandParser(): De constructor van de klasse.
2. std::vector<std::string> CommandParser::splitCommands(std::string &messageString): Splitst een invoerstring in afzonderlijke commando's en retourneert deze als een vector van strings.  
   Input is bijvoorbeeld: “#0 P1500 #1 P1500 #2 P1500 T300”  
   Output is dan: {“#0 P1500”, “#1 P1500”, “#2 P1500 T300”}
3. int CommandParser::parseIntAfterChar(char delimiter, bool &isValid, std::string &messageString): Haalt een geheel getal op dat volgt op een bepaald teken (delimiter) in de invoerstring.
4. int CommandParser::getChannel(std::string &messageString, bool &isChannel): Haalt het kanaal op uit de invoerstring en controleert of het geldig is.
5. int CommandParser::getPulseWidth(std::string &messageString, bool &isChannel): Haalt de pulsduur op uit de invoerstring en controleert of deze binnen een geldig bereik ligt.
6. int CommandParser::getDuration(std::string &messageString, bool &isChannel): Haalt de duur op uit de invoerstring.
7. int CommandParser::getSpeed(std::string &messageString, bool &isChannel): Haalt de snelheid op uit de invoerstring.
8. CommandParser::ServoCommand CommandParser::parseServoCommand(std::string &messageString, bool &isValid, bool &usingDuration): Parseert een enkele servo-opdracht en retourneert deze als een ServoCommand struct. Hier wordt gecontroleerd of de opdracht geldig is. Een messageString kan er als volgt uitzien:“#0 P1500 S500”  
   De struct ziet er dan zo uit:  
   SingleServoCommand{  
    channel = 0;  
    pulseWidth = 1500;  
    speedPWM = 500;  
     
    usingSpeed = true;  
    speedAnglePerSecond = UNSET;   
   }  
     
   Na het uitvoeren heb je dus een struct. SpeedAnglePerSecond wordt pas gezet in de functie calculateRealDuration
9. CommandParser::CompleteCommand CommandParser::parseCompleteCommand(std::string &messageString, bool &isValid): Parseert een volledige opdracht en retourneert deze als een CompleteCommand struct. Deze functie maakt gebruik van de bovenstaande parseServoCommand-functie om individuele servo-opdrachten te analyseren. De messageString van deze functie komt vanuit de /communicator\_node node binnen via de topic /robot\_command.
10. int CommandParser::calculateRealDuration(CompleteCommand &command, sensor\_msgs::msg::JointState robotPositionMessage\_): Berekent de werkelijke duur van een opdracht op basis van de huidige posities van de servomotoren in de robot. Het berekent ook de benodigde tijd om van de huidige positie naar de gewenste positie te bewegen en selecteert de langste tijd als de werkelijke duur, zodat alle servos op de gesimuleerde robotarm tegelijk klaar zijn met bewegen, zoals bij de echte AL5D robotarm ook zo is.

## TestNode

**Membervariabelen:**

1. sim\_link\_, bot\_link\_, cup\_link\_: Namen van frames die worden gebruikt in TF2-transformaties. sim\_link is hierbij de 0,0 positie van de wereld. bot\_link\_ is de positie van de robot op de sim\_lin\_ en cup\_link\_ is de frame van de beker. Deze kan dus gekoppeld zijn aan de hand-frame van de robot wanneer opgepakt, of aan de sim\_link\_ wanneer niet opgepakt.
2. joint\_state\_message\_: Een bericht dat de gezamenlijke toestand van alle joints van de robot beschrijft
3. joint\_state\_pub\_: Een publisher voor het publiceren van gezamenlijke toestandsberichten. Hiermee wordt de robot dus verplaatst
4. robot\_command\_sub\_: Een subscription voor het ontvangen van robotcommando's die binnenkomen vanaf de CommunicatorNode.
5. timer\_: Een timer voor periodieke uitvoering van een callback die het bewegen van de robotarm uitvoert.
6. cupTransformTimer\_: Een timer voor periodieke updates van de bekertransformatie, zodat ze niet wegvagen in de Rviz simulatie.
7. pickupCupClient\_: Een client voor het verzenden van verzoeken naar de pickup\_cup-service. Hiermee kan de beker opgepakt of gedropt worden.
8. parser: Een instantie van de CommandParser-klasse voor het analyseren van binnenkomende commando’s vanuit de CommunicatorNode.
9. commandQueue: Een wachtrij voor opdrachten die moeten worden uitgevoerd. Alhoewel dit een wachtrij is, staat er altijd maar 1 of 0 commando’s in de queue. Als de robotarm namelijk bewogen wordt en er is een nieuwe commando binnengekomen vanuit de CommunicatorNode, wordt de huidige beweging geannuleerd, net zoals bij de echte AL5D robotarm.
10. buffer\_, listener\_, broadcaster\_: Instanties van TF2-componenten voor het bufferen, ontvangen en versturen van TF2 berichten.

**Memberfuncties:**

1. TestNode::TestNode(): De constructor van de klasse.
2. void TestNode::publish\_joint\_state(): Publiceert de huidige toestand van de joints van de robotarm. Het aanpassen van de joint\_state\_message betekend dus een verandering van de positie van een of meerdere joints van de robotarm. Hiermee wordt dus de robotarm bewogen.
3. void TestNode::timerCallback(): Wordt periodiek door een timer uitgevoerd. Deze functie kijkt of er een commando in de queue staat en als dat zo is, beweegt de te bewegen servo’s volgens de aangegeven snelheid. Hierdoor beweegt de robotarm geleidelijk. Als de servo’s op of dichtbij hun eindpunt zijn, wordt de huidige commando uit de queue verwijdert. Tevens zorgt deze functie er voor dat er een Client request wordt gedaan naar de /pickup\_cup service wanneer de gripper volledig open of dicht is. Hierdoor wordt de beker dus opgepakt wanneer de gripper volledig dicht is en dichtbij de beker is en wordt de beker laten vallen wanneer de gripper volledig open is.
4. void TestNode::onClosedGripper(): Handelt de gebeurtenis af wanneer de grijper wordt gesloten door een verzoek naar de pickup\_cup-service te sturen met parameter pickup op true.
5. void TestNode::onOpenedGripper(): Handelt de gebeurtenis af wanneer de grijper wordt geopend door een verzoek naar de pickup\_cup-service te sturen met parameter pickup op false.
6. void TestNode::handle\_robot\_command(const msg\_srv::msg::RobotCommand::SharedPtr msg): Handelt inkomende robotcommando's af door ze te vertalen naar CommandParser::CompleteCommand met behulp van de parseCompleteCommand functie van de parser. Daarna wordt de CommandParser::CompleteCommand toegevoegd aan de queue.
7. void TestNode::emptyQueue(): Leegt de commandQueue door alle opdrachten te verwijderen.
8. void TestNode::skipCurrentCommand(): Slaat de huidige opdracht over door deze uit de commandQueue te verwijderen.
9. void TestNode::handleCupTransform(): Handelt de bekertransformatie af tussen de simulatiewereld en de beker. Hierdoor verdwijnen de frames in Rviz niet.
10. void TestNode::initTF2(): Initialiseert een transformatie van het simulatielink naar het basislink met behulp van TF2, wat dus de positie bepaalt van de robot ten opzichte van de gesimuleerde wereld. In dit geval is het statisch en wordt de robotarm op X0 Y0 Z0 gezet.

## MathUtils

De gegeven C++-klasse MathUtils maakt deel uit van een namespace genaamd Utils en bevat enkele statische hulpprogrammafuncties voor het omzetten van hoeken tussen graden en radialen. Hieronder is een beschrijven van de member variables en functies

**Membervariabele:**

1. PI: Een constante dubbele waarde die π (pi) vertegenwoordigt. Dit is niet opgeslagen in MathUtils, maar in de namespace Utils.

**Statische Functies:**

1. static double MathUtils::toRadians(double aDegrees): Een statische functie die een hoek in graden ontvangt en deze omzet in radialen door de waarde te vermenigvuldigen met de constante PI en te delen door 180 graden. Het resultaat wordt als een double geretourneerd.
2. static double MathUtils::toDegrees(double aRadian): Een statische functie die een hoek in radialen ontvangt en deze omzet in graden door de waarde te vermenigvuldigen met 180 graden en te delen door de constante PI. Het resultaat wordt als een double geretourneerd.

## ServoUtils

**Statische Membervariabelen:**

1. static short minServo: Het minimumservonummer (0).
2. static short maxServo: Het maximumservonummer (5).
3. static double maxSpeed: De maximale snelheid in graden per seconde waarmee een servomotor kan bewegen (360 graden).
4. static double maxGripperSpeed: De maximale snelheid van de grijper in graden per seconde. Alhoewel een gripper niet een hoek heeft of draait, gebruikt TF2 radialen voor het bepalen van de positie van de grippers. Om te voldoen aan de eis VS04, oftewel “De virtuele robotarm gedraagt zich realistisch m.b.t. tijdgedrag (servo’s roteren kost tijd en gaat geleidelijk).” moet de gripper dus ook een maximale snelheid hebben. Als de maximale snelheid van 360 graden wordt gebruikt, gaat de gripper namelijk veel te snel. Daarom zijn hier en daar ook wat extra functionaliteiten voor het omrekenen van grippersnelheden in mijn programma.
5. static short gripperLeftTF2: Het TF2-jointnummer voor de linker grijper.
6. static short gripperRightTF2: Het TF2-jointnummer voor de rechter grijper.
7. static short gripperServoAL5D: Het servonummer voor de grijper op de AL5D-robotarm.
8. static short wristServoAL5D: Het servonummer voor de pols op de AL5D-robotarm.
9. static short wristServoTF2: Het TF2-jointnummer voor de pols.

**Statische Functies:**

1. static double ServoUtils::pwmToDegrees(int pwmDurationUs, short servo): Zet een PWM-duur in microseconden om naar graden. De omzetting wordt berekend met behulp van lineaire interpolatie op basis van de minimale en maximale PWM-duur.
2. static double ServoUtils::pwmToDegreesGripper(int pwmDurationUs): Deze functie zet PWM-duur specifiek voor de grijper om naar graden. Het past een offset toe op de PWM-duur en voert vervolgens een eenvoudige conversie uit. Hierdoor is PWM 2500 bij de gripper volledig dicht en PWM500 bij de gripper volledig open
3. static double ServoUtils::pwmPerSecondToDegreesPerSecond(int pwmPerSecond, short servo): Zet PWM per seconde om naar graden per seconde. Deze conversie gebruikt ook de PWM-naar-graden omzetting en de opgegeven PWM per seconde.
4. static double ServoUtils::degreesToPwm(int servo, double degrees): Zet graden om naar PWM-duur in microseconden. Deze omzetting is het omgekeerde van pwmToDegrees.
5. static double ServoUtils::getMaxSpeed(short servo): Retourneert de maximale snelheid van een servomotor, afhankelijk van het type servomotor (gewone servomotor of grijper).

