Ontwerpdocument Robotarm Interface

WoR World

Inhoudsopgave

Inleiding

In dit document is het ontwerp voor de hardware interface opdracht van WoR World terug te vinden. Allereerst een korte toelichting op de geschreven software voor deze opdracht:

Voor deze opdracht is er een ‘server’-applicatie gemaakt, welke bestaat uit een highlevel- en een lowlevel-driver. Het doel van deze server-applicatie is om aan de hand van binnenkomende instructies een AL5D-robotarm aan te sturen. De verschillende instructies worden binnen de server-applicatie gevalideerd, vertaald en vervolgens via een seriële verbinding naar de robotarm gestuurd. Naast de server-applicatie is er een client-applicatie ontwikkeld. Deze client-applicatie verstuurt een reeks van verschillende instructies naar de server-applicatie, om zo de robot verschillende acties uit te laten voeren. De applicatie maakt gebruik van het het framework ROS.

In dit ontwerpdocument komen er allereerst gemaakte ontwerpbeslissingen aan bod. Ook zullen er verschillende UML-diagrammen aan bod komen met begeleidende tekst. Het doel van dit document is om de werking van verschillende processen die voorkomen in het systeem inzichtelijk te maken. Daarnaast wordt er toegelicht hoe het systeem voldoet aan de QoS (Quality-of-Service) eisen.

Ontwerpbeslissingen

In dit hoofdstuk zijn een aantal belangrijke ontwerpbeslissingen terug te vinden.

Initialisatie applicatie

Volgens de opdracht is het de bedoeling dat de robotarm bij het opstarten van het systeem naar de “park” stand gaat. Echter is dit niet altijd mogelijk omdat er in de documentatie wordt aangegeven dat de snelheid bij het eerste ontvangen commando wordt genegeerd. Dit omdat de robotarm dan nog niet weet in welke stand hij staat.

“Because the controller doesn't know where the servo is positioned on power-up, it will ignore speed and time commands until the first normal command has been received.”

Dit betekent dat wanneer de robot net voor het eerst is aangezet, hij niet binnen de gevraagde 3 secondes naar de park stand zal gaan. De robot zal dit zo snel mogelijk doen.

Afbreken applicatie

De server-applicatie kan worden gesloten door een “shutdown” instructie. Wanneer er zo’n instructie binnenkomt zal de robot naar de “park” stand worden gestuurd. Vervolgens zal de server-applicatie beëindigd worden.

Out-of-range commando’s

Zoals in de opdracht aangegeven mogen er alleen hoekwaardes worden afgehandeld die binnen de toegestane range of motion vallen. Als de waardes niet binnen de range vallen zijn er echter twee opties, namelijk het limiteren op de hoogste/laagste toegestane waarde en die uitvoeren of het negeren van het commando. Wij hebben ervoor gekozen om in dit geval het commando volledig te negeren.

Noodstop

Een van de eisen was dat de robot gestopt moest kunnen worden. Dit is geimplementeerd, wanneer er een armInstruction message wordt verzonden met als instructie “stop” zal de robot onmiddellijk stoppen met bewegen. Daarnaast worden eventuele move-commando’s uit de move-queue verwijderd. De robot zal zich in een vergrendelde staat bevinden waarin hij alle binnenkomende move-commando’s zal weigeren uit te voeren. Wel is het mogelijk om de robot te laten afsluiten middels een instructie “shutdown”, waarbij de robot naar de park stand wordt gestuurd. De robot kan weer ontgrendeld worden door de instructie “release” te sturen. Move-commando’s die hierna worden verstuurd zullen dan weer uitgevoerd worden.

QoS Constraints

Timeliness

In dit hoofdstuk wordt bewezen dat de constraint die in SA03 staat gehaald wordt door de applicatie. De constraint is als volgt gedefinieerd : "*Als de gripper van AL5D-robotarm naar een locatie (een set van samen-gestelde servohoeken) wordt gestuurd moet deze binnen 2,3 seconden worden bereikt.*".

Om dit aan te tonen hebben we een aantal metingen gedaan waarin te zien is hoeveel tijd er zit tussen het sturen van een move-commando tot het daadwerkelijk bereiken van de beoogde posititie.

Hierbij houden we rekening met de volgende aspecten:

- Tijd voor het versturen van een ROS message van client naar server.
- Tijd die de server nodig heeft om de ontvangen message om te zetten naar een serieel commando.
- Tijd voor het verzenden van het commando naar de seriële poort.
- Tijd voor het versturen van de gewenste pulsbreedte (vanaf SSC-32U naar AL5D)
- Tijd voor het bewegen van de servo's naar de beoogde positie.

Hierbij laten we een aspect buiten beschouwing omdat dit niet te meten is, dit is het vertalen van het seriele commando door de SSC-32U. Hierover is in de documentatie geen informatie te vinden. In principe zou het inlezen van een serieel commando en het vertalen hiervan niet lang moeten duren. We gaan er hier vanuit dat deze tijd verwaarloosbaar is.

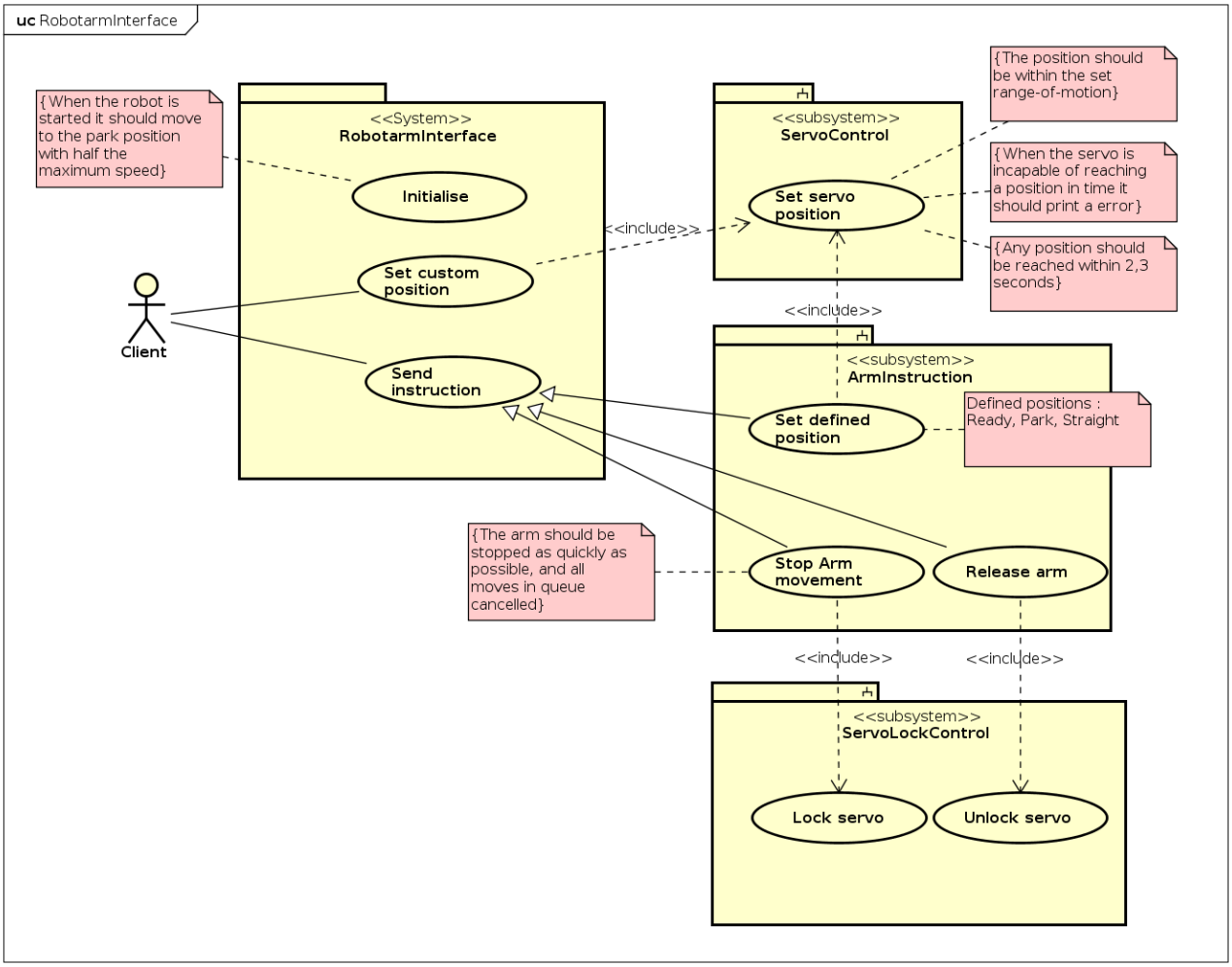
Voor de rest van de aspecten gaan we uit van de worst-case scenario.

Uit testen is gebleken dat deze aspecten zo lang duren:

- ROS message client → server : 175166 nanoseconden → 0.175 millisecondes.
- Server-applicatie vertalen message : 30.000~ nanoseconden → 0.03 millisecondes.
- Verzenden commando naar seriële poort: 20.000~ nanoseconden → 0.02 millisecondes.
- Tijd voor het versturen van de gewenste pulsebreedte: 0.5 – 2.5 millisecondes.
- Beweging servo (tot 180°) met een minimale snelheid van 60°/0.19sec → 1°/3.16666666667ms leidt tot een periode van maximaal 570 millisecondes.

Dit leidt tot een totale periode van 572.725ms, wat ruim binnen de vereiste 2.3 secondes zit.

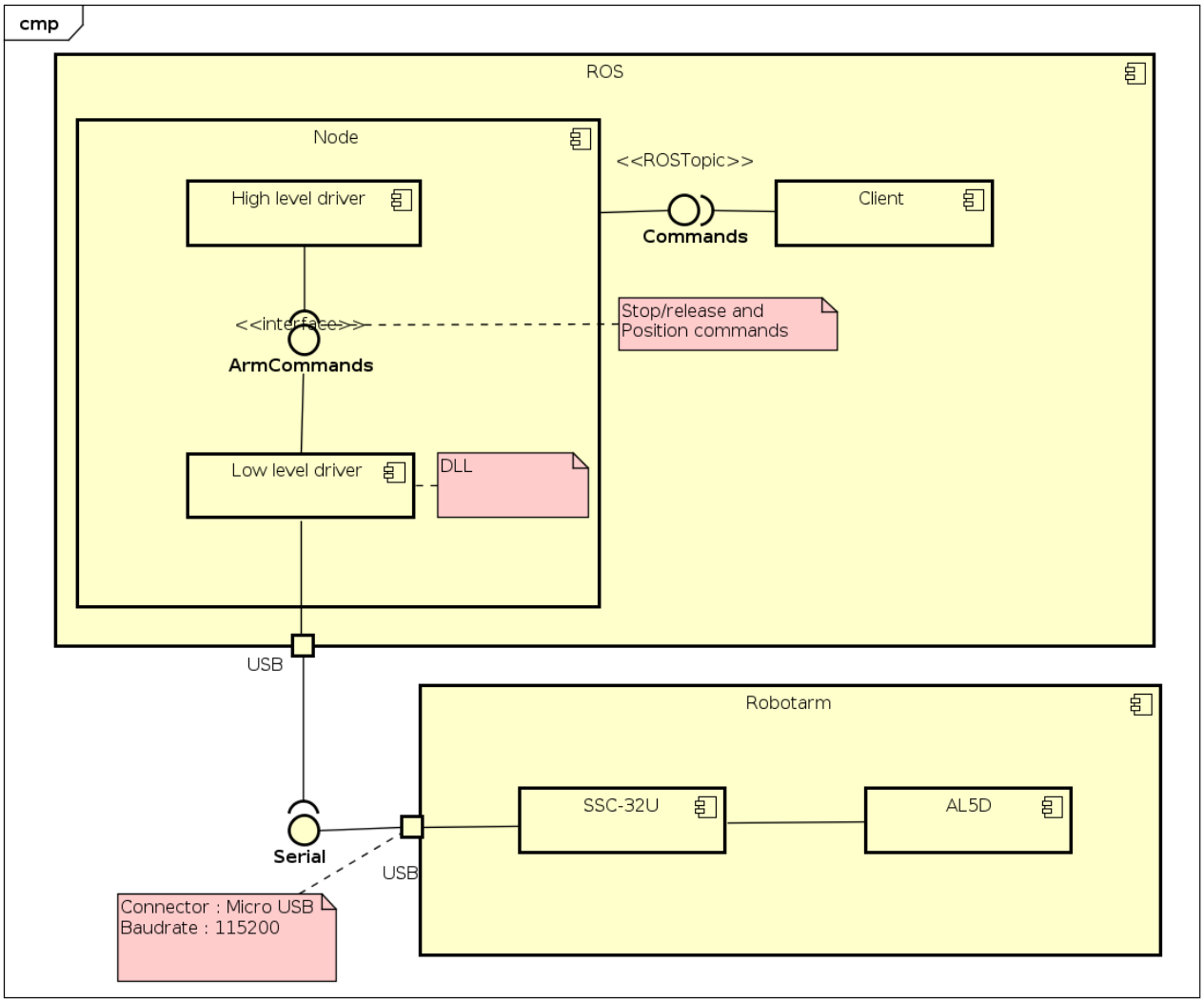
Usecase diagram



Afbeelding 1: Usecase Diagram

Binnen dit usecase diagram is er een client die gebruik maakt van de robotarm. Deze kan de robotarm naar een van te voren ingestelde positie of een zelfgekozen positie sturen. Bij het uitvoeren van deze usecases worden de benodigde usecases in de subsystemen aangeroepen. Zo bestaat er een subsysteem om een servo naar een bepaalde positie te zetten welke gebruikt wordt door zowel de van te voren ingestelde versie en de zelfgekozen versie. Voor het stoppen is een ander subsysteem gedefinieerd. De release en stop commando’s maken gebruik van de lock en unlock usecases.

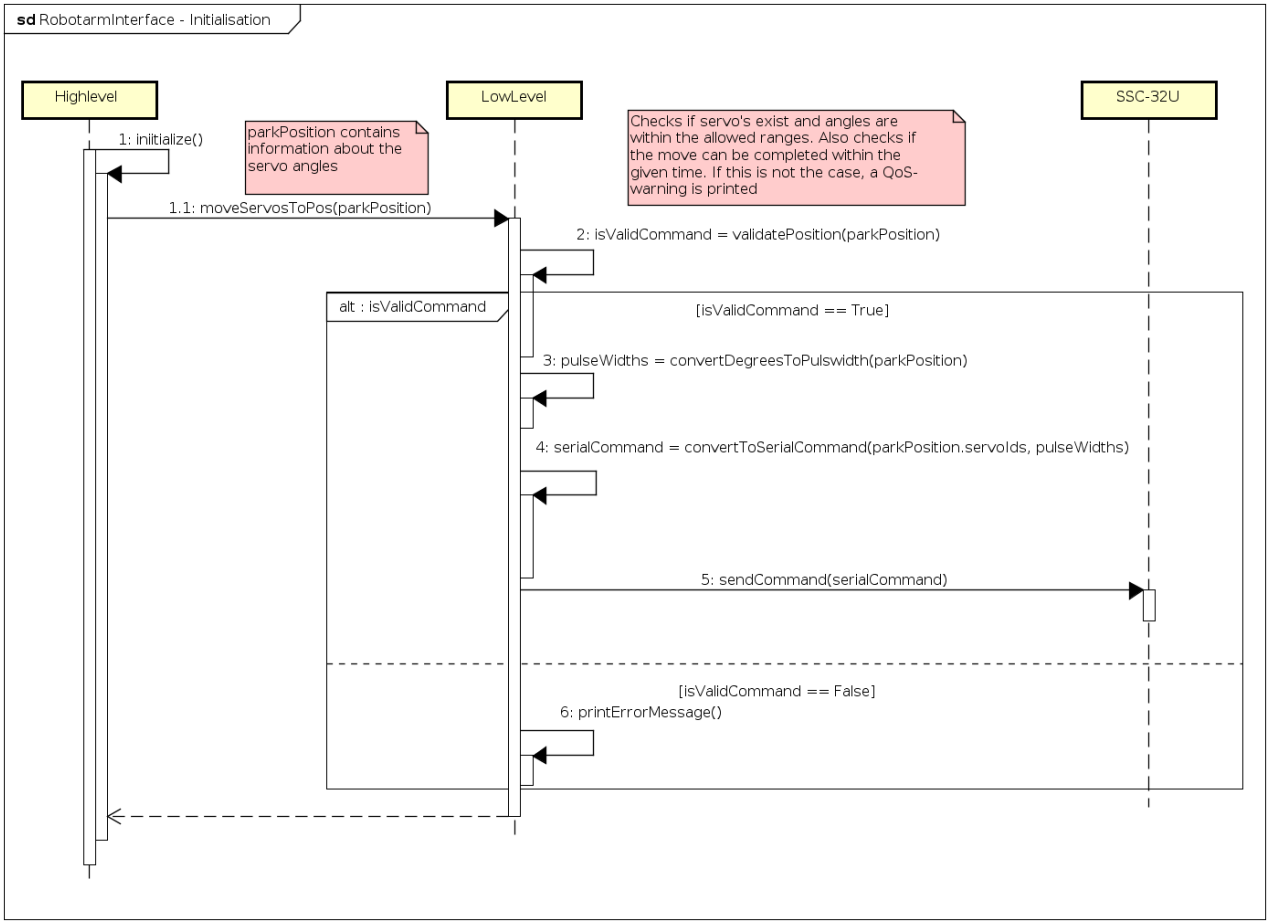
Component diagram



Afbeelding 2: Component Diagram

In dit component diagram zijn de verschillende componenten van het systeem te zien. De client kan gebruikt worden om commando's naar de highlevel-driver te sturen via een ROS topic. Binnen de highlevel-driver worden deze commando's vertaald en doorgeven aan de lowlevel-driver. Deze vertaalt de aanroepen naar een commando welke via de seriële poort naar de SSC-32U wordt verstuurd. De SSC-32U vertaalt op zijn beurt dit commando en stuurt de daadwerkelijke servo's in de AL5D aan.

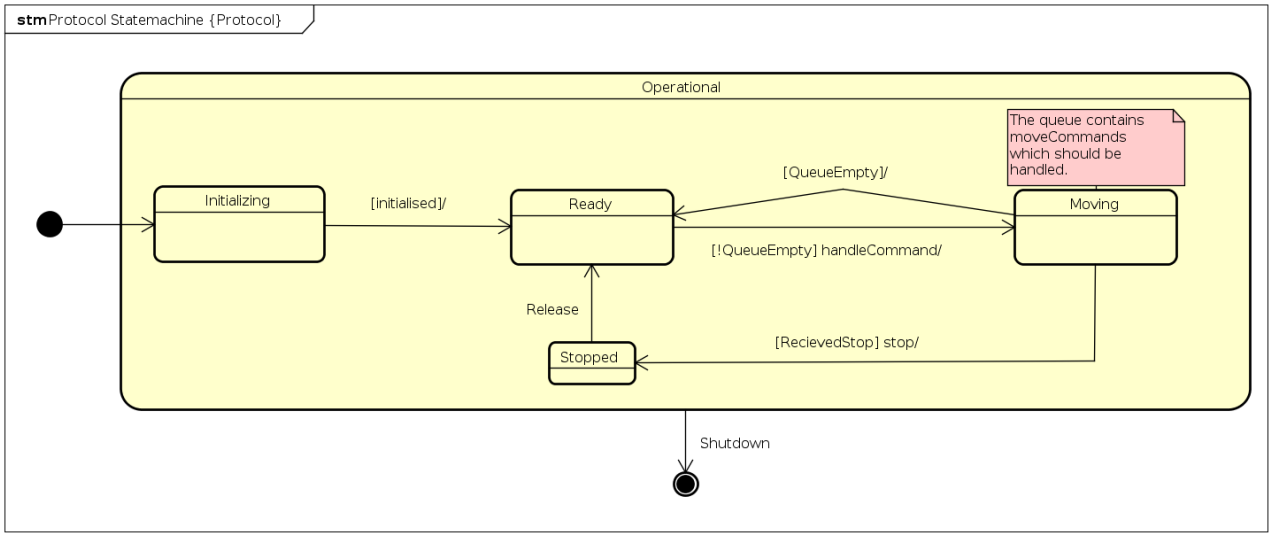
Sequence diagram



Afbeelding 3: Sequence Diagram Initialisatie

In het bovenstaande diagram is de opstartprocedure van de robotarm te zien in de vorm van een sequence diagram. Hierbij wordt de arm bij het opstarten naar de “park” positie gestuurd. Dit commando wordt zoals alle andere commando’s eerst gecontroleerd. Als het commando valide is dan wordt deze vertaald naar een string welke naar de SSC-32U gestuurd wordt verstuurd. Wanneer het commando niet valide is wordt er een error gegenereerd en wordt het commando genegeerd.

Protocol State Diagram



Afbeelding 4: Protocol-State Diagram

In dit protocol state diagram zijn de verschillende toestanden van de server-applicatie te herkennen en de toegestane toestandsveranderingen. Hier is te zien dat het programma bij het opstarten naar de initialisatie toestand gaat om de robotarm in de “park” stand te zetten. Hierna wordt er geluisterd naar binnenkomende commando’s. Binnenkomende move-commando’s worden toegevoegd aan een lijst. Hierdoor zal de toestand veranderen naar moving. Wanneer er tijdens de moving toestand een stop commando binnenkomt gaat het programma naar de stopped toestand waarna de robotarm gelocked wordt en wacht op een release event. Na een release event ontvangen te hebben gaat de applicatie weer naar de ready toestand.