Software Design Description

studentnummer: 1527782
opleiding: HBO-ICT / Afstuderen
profiel: ESD
docenten: Joost Kraaijeveld, Chris van Uffelen
bedrijfsbegeleider: Jeroen Veen
versie: 0.8

Agit Tunç

16 april 2025

HAN_UNIVERSITY
OF APPLIED SCIENCES

INHOUDSOPGAVE

1 REVISIE HISTORIE

Versie	Datum	Aanpassing	Auteur
0.1	25-09-2024	Initieel opzet in word	Agit
0.1	03-10-2024	Eerste versie component diagram. Initiele versie radar component en camera component. (In Word)	Agit
0.2	16-12-2024	Overgestapt naar latex. Nieuwe initiele versie. Deployment + Component diagram toegevoegd	Agit
0.3	17-12-2024	Toegevoegd: Apart radar HLD+LLD component diagram + class en sequence diagram radar LLD	Agit
0.4	11-02-2025	Update deployement en componentdiagram: microfoon en speaker zijn weggehaald en provided en required interface zijn omgedraaid van elk component.	Agit
0.5	21-02-2025	Update deployement componentdiagram naamgeving componenten. Radar Camera Controller Eye component toegevoegd in gedetailleerd design	Agit
0.6	27-02-2025	Update Camera HLD: Afstand berekening doen we op basis van gezichtsbreedte	Agit
0.7	12-03-2025	Update Camera HLD: Alle afhankelijkheden m.b.t. iris detectie verwijderd. Update InteractionController: Transformatie van coördinaten toegevoegd. Update Eye Display LLD: Sequence diagram pupil dialation toegevoegd.	Agit
0.8	31-03-2025	Audio component alleen toegevoegd aan hoofdstuk architectuur. Geen detailed design aanwezig. Zie hiervoor de README op github. Hoofdstuk 6 aangevuld.	Agit

2 BEGRIPPENLIJST

Begrip	Omschrijving
BuddyBot	Het ontwikkelde systeem. Een prototype sociale robot in ROS2 ontwikkeld.
HLD / hld	Afkorting voor High Level Driver. De afkorting wordt gebruik samen met andere woorden in het document. Voorbeeld: Radar_HLD = Radar High Level Driver
LLD / lld	Afkorting voor Low Level Driver. De afkorting wordt gebruik samen met andere woorden in het document. Voorbeeld: Radar_LLD = Radar Low Level Driver

3 INTRODUCTIE

In dit document is het ontwerp vastgelegd van het prototype sociale buddy robot. Dit prototype wordt als systeem gedurende het document gerefereerd als ”BuddyBot”. Het doel van dit project was om een prototype van een sociale robot in ROS2 te ontwikkelen met behulp van het algemene robotmodel. De gewenste interacties die BuddyBot moet kunnen, zijn te vinden in het SRS document.

In dit document is het ontwerp van BuddyBot te vinden. Eerst wordt op architectuurniveau het systeem toegelicht, vervolgens wordt in de opvolgende hoofdstukken elke softwarecomponent in detail toegelicht.

4 ARCHITECTUUR

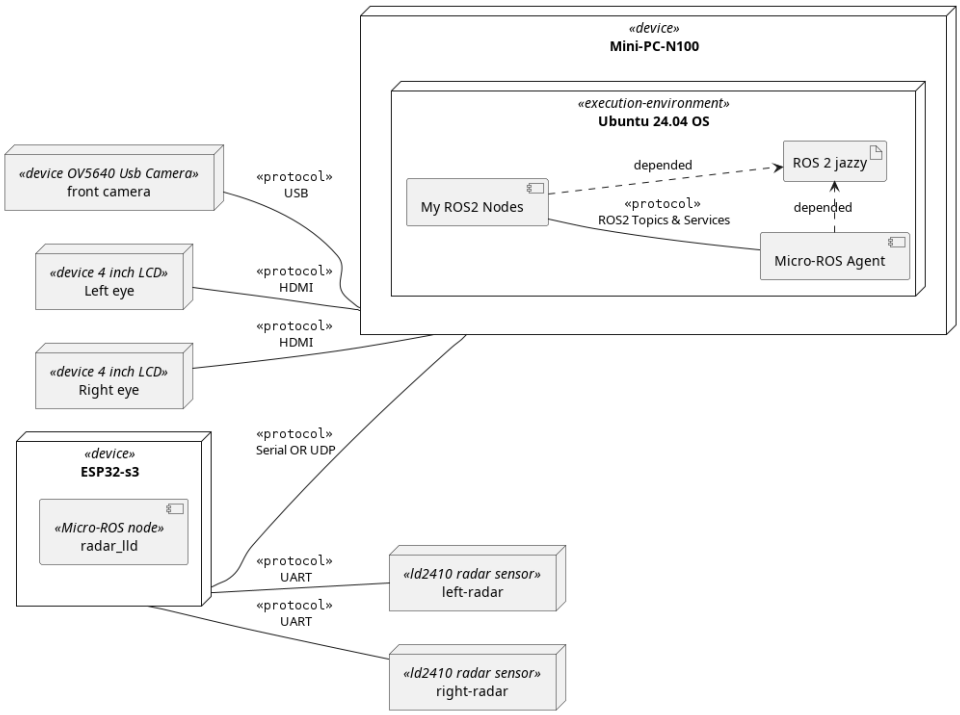
4.1 ALGEMENE BESCHRIJVING

In dit hoofdstuk wordt de software op hoog niveau uitgelegd. Eerst kaarten we de gebruikte hardware componenten met hun software componenten met behulp van een deployment diagram. Vervolgens kaarten we **alleen** de ontwikkelde software componenten aan met behulp van een component diagram.

4.2 HARDWARE COMPONENTEN

BuddyBot bestaat uit diverse hardwarecomponenten te vinden in Figuur 4.1. Op de ”Mini-PC-N100” draait de ROS2 software van BuddyBot. De ontwikkelde ROS nodes op dit device zijn weergeven met alleen het component ”My ROS2 Nodes”, om het diagram overzichtelijk te houden. In het component-diagram worden de ontwikkelde ROS2 componenten verder in detail toegelicht. De micro-ROS Agent component is een package van ROS2 waarmee berichten tussen micro-ros nodes en ROS2 nodes uitgewisseld kunnen worden. Een micro-ros node die we gebruiken in het prototype is de ”Radar_LLD”. Deze node leest meerdere radaraanwezigheids sensoren af en stuurt dit in een keer, als een custom ROS2 message, door naar de Mini-PC-N100.

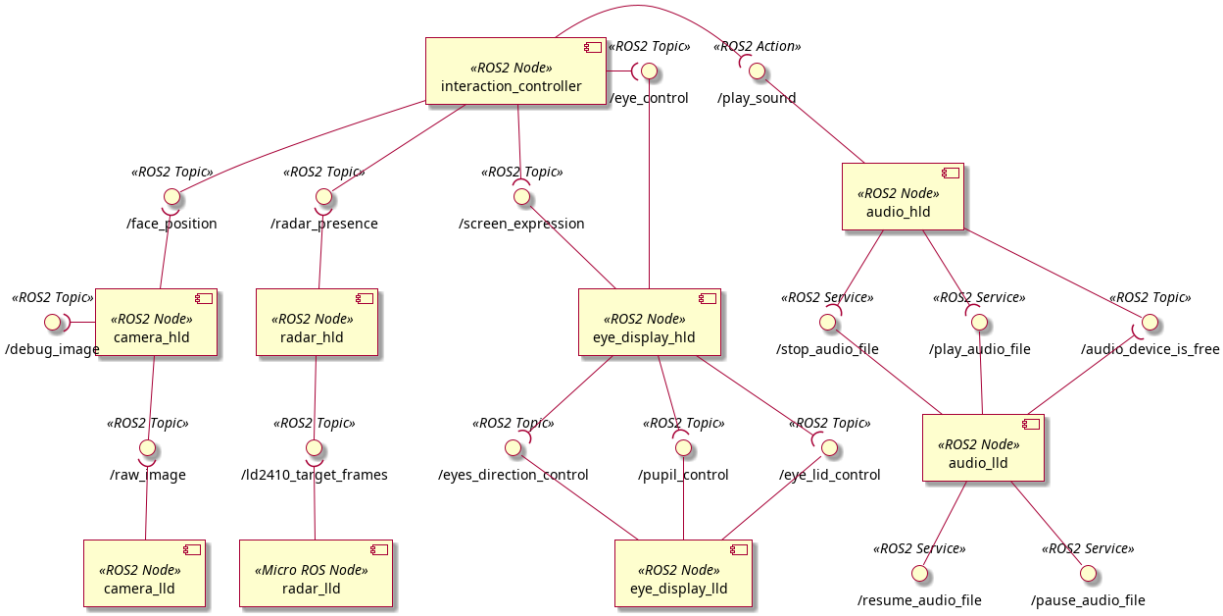
De camera wordt gebruikt om een gezicht te detecteren. De LCD schermen worden gebruikt om ogen te visualiseren, die een gedetecteerde gezicht hoort te volgen. De speaker wordt gebruikt om een gebruiker te begroeten en afscheid te nemen.



Figuur 4.1: Deployment diagram BuddyBot.

4.3 SOFTWARE COMPONENTEN

In Figuur 4.2 zijn de ontwikkelde software componenten vastgelegd. De software is volgens het algemene robotmodel ontwikkeld. Er is een controller aanwezig genaamd “interaction_controller“. Deze controller beslist wat voor interactie met de gebruiker uitgevoerd word op basis van sensor inputs. De sensoren en actuatoren zijn verdeeld over high level en low level driver componenten. De samenwerking tussen alle software componenten zijn met ros topics, services of action gerealiseerd, te zien met de stereotype «ROS2 topic», «ROS2 service» of «ROS2 action».



Figuur 4.2: Component diagram BuddyBot.

4.3.1 INTERFACES TUSSEN COMPONENTEN

Het onderstaande tabel omschrijft de interfaces tussen de ROS nodes. Er wordt hier nadruk gelegd wat voor soort berichten uitgewisseld wordt en hoe de stroming van informatie overdracht gaat. De bericht types zijn klikbaar en brengen naar hun definities. Het type interface is verwerkt in het “interface naam“ kolom, om het tabel leesbaar te houden.

Interface naam	Bericht type	Omschrijving
«ROS2 Topic» /raw_image	sensor_msgs/Image	De camera_lld stuurt **ruwe** plaatjes, die hij uit de camera leest, naar de camera_hld, zodat de camera_hld een gezicht kan herkennen.
«ROS2 Topic» /face_position	geometry_msgs/PointStamped	Wanneer een gezicht is gevonden, wordt de coördinaten van het gezicht doorgestuurd naar de interaction_controller. Als coordinaten wordt de middel punt van het gezicht verstuurd (x,y,z) als ROS coördinaten (Right-hand rule). De coördinaten representeert de positie relatief tot de camera frame .
		Vervolg op volgende pagina

Interface naam	Bericht type	Omschrijving
«ROS2 Topic» /ld2410_target_frames	LD2410TargetDataFrameArray	De radar_lld stuurt de ruwe sensor waardes van elk aangesloten ld2410 sensor in een keer in lijst vorm op naar de radar_hld, zodat die kan bepalen of een persoon binnen het interesse gebied van de robot aanwezig is.
«ROS2 Topic» /radar_presence	PresenceDetection	De radar_hld stuurt een detectie bericht naar de interaction_controller wanneer een verandering van ”detectie” plaats vind.
«ROS2 Topic» /screen_expression	ScreenExpression	De interaction_controller geeft aan dat de ogen moeten openen of sluiten aan de eye_display_hld. De ogen gaan open wanneer een PresenceDetection bericht aangeeft dat een persoon binnen het interesse gebied aanwezig is en sluiten wanneer er niemand aanwezig is binnen het gebied.
«ROS2 Topic» /eye_control	EyeControl	De interaction_controller berekent op basis van gezicht positie de kijk hoek voor de ogen in radialen en de afstand van de ogen naar het gezicht in centimeters. Deze informatie wordt naar de eye_display_hld verstuurd.
«ROS2 Topic» /eyes_direction_control	EyesDirection	De eye_display_hld vertaalt de kijkhoek in radialen uit het EyeContol bericht naar kijkhoek in grades en verstuurd dit naar de eye_display_lld zodat de ogen in de juiste richting gevisualiseerd worden.
«ROS2 Topic» /pupil_control	PupilControl	De eye_display_hld vertaalt de afstand uit het EyeContol bericht naar een pupil verwijding in percentages en stuurt dit naar de eye_display_lld zodat de pupilen de juiste grootte krijgen.
«ROS2 Topic» /eye_lid_control	EyeLidControl	De eye_display_hld vertaalt het open en sluiten van de ogen naar welke posities de oogleden van elk oog moeten zijn en verstuurt dit naar de eye_display_lld.
«ROS2 Action» /play_sound	PlaySound	De interaction_controller begroet met spraak een gebruiker met behulp van het “play_sound“ action service als iemand binnen een straal van twee meter volgens de radar aanwezig is en de camera ziet dat een gezicht voor de robot staat. Wanneer er niemand in zijn straal aanwezig is en er is al eerder begroet dan neemt de robot ook afscheid met spraak.
«ROS2 Service» /play_audio_file	PlayAudioFile	De audio_hld vertaalt PlaySound request naar een audio afspeel request dat de audio_lld afspeelt.
«ROS2 Service» /stop_audio_file	std_srvs/Trigger	Een audio bestand kan gestopt worden wanneer dit afgespeeld wordt in de audio_lld. Een situatie waar dit voor kan komen is dat gebruiker heel snel voor de robot komt te staan en wegloopt. Dan wordt het begroeten gestopt en gelijk afscheid genomen.
«ROS2 Topic» /audio_device_is_free	std_msgs/Bool	De audio_lld geeft zijn beschikbaarheid aan of die vrij is voor gebruik. Hierdoor weet de audio_hld of het afspelen van een audio is voltooid en klaar is voor een volgende afspeel ronde.

Tabel 4.1: Interface beschrijving tussen componenten

4.3.2 EXTRA INTERFACES

In het onderstaande tabel worden extra interfaces beschreven dat momenteel **niet** gebruikt wordt door de ontwikkelde software componenten. Deze interfaces kunnen uiteraard via de ROS CLI aangeroepen worden of later door andere nodes gebruikt worden.

Interface naam	Bericht type	Omschrijving
«ROS2 Topic» /debug_image	sensor_msgs/Image	Met dit topic kan een ontwikkelaar zien waar de gezicht in de camera view zich bevind en hoever het van de camera staat. Aangeraden is om ROS CLI “rqt_view“ te gebruiken, omdat dit een kant en klare image viewer heeft waar je op dit topic kan luisteren.
«ROS2 Service» /pause_audio_file	std_srvs/Trigger	Een afgespeelde audio bestand kan met behulp van deze service op pauze gezet worden.
«ROS2 Service» /resume_audio_file	std_srvs/Trigger	Een gepauzeerd audio bestand kan hervat worden met behulp van deze service.

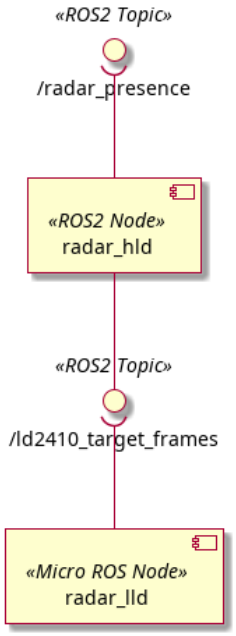
Tabel 4.2: Interface beschrijving extra interfaces

5 GEDETAILLEERD DESIGN

In dit hoofdstuk zoomen we in op elk component uit het component diagram en wordt in detail beschreven hoe elk component werkt. Voor alle klasse diagrammen geld dat dat constructor en destructor niet is weergeven, om het diagram leesbaar te houden.

5.1 RADAR COMPONENT

In dit hoofdstuk wordt de low level en high level driver van het radar component toegelicht.



Figuur 5.1: Component diagram radar.

5.1.1 GERELATEERDE REQUIREMENTS EN RICHTLIJNEN

INITIELE REQUIREMENTS

Voor de opdrachtgever is het gewenst om een gebruiker binnen een straal van 4 meter te kunnen meten. Daarnaast is het gewenst om te weten in welk hoek vanuit de robot de gebruiker aanwezig is. De bestelde radar sensoren zijn radaraanwezigheids sensoren (ld2410), helaas blijkt dat deze sensoren geen hoek informatie opgeeft. Een betere radar sensor voor een vervolg project, dat ook hoek informatie kan aanbieden, zou een van de volgende sensoren zijn LD2450, LD2461.

NIEUWE REQUIREMENTS

De requirement voor dit project is hierdoor veranderd. Het is voldoende om te weten of een gebruiker binnen een ”configurable” straal in meters aanwezig is. De standaard drempelwaarde bij geen configuratie is een straal van 2 meter.

RICHTLIJNEN

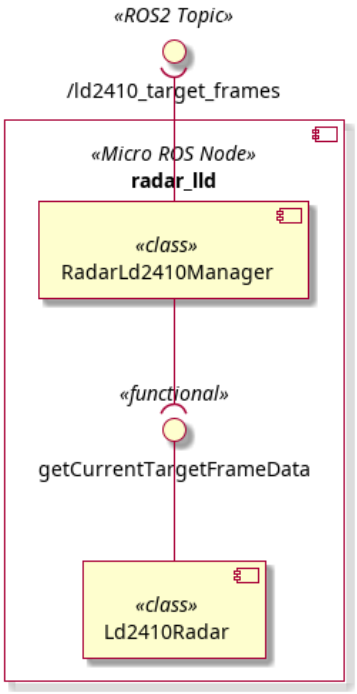
De volgende richtlijnen zijn gehanteerd voor de ontwikkeling van de radar low level en high level driver componenten.

Richtlijn	Omschrijving
Programmeertaal	Voor de LLD is er gebruik gemaakt van C++11 met het PlatformIO framework en de micro_ros bibliotheek. Voor de HLD is de software geschreven in C++17.
Coding guidelines	Voor beide componenten wordt de ROS2 Jazzy coding style guide gebruikt.
Codekwaliteitscontrole	Code wordt gecontroleerd met ROS2 linter en cppcheck.
Compiler instellingen	De software voor de HLD wordt gecompileerd met: `-Wall -Wextra -Wpedantic`.
Afhankelijkheid	Om ROS data te kunnen versturen, moet aan de ontvangende kant een ”micro-ros agent” draaien.

Tabel 5.1: Richtlijnen voor radarcomponent

5.1.2 RADAR LLD

De low level driver is als een micro-ROS node ontwikkeld dat draait op de esp32-s3. Het bestaat uit twee onderdelen te zien in Figuur 5.2.



Figuur 5.2: Component diagram radar LLD.

Type interface	Interface naam	Omschrijving
Functie	getCurrentTargetFrameData	Hiermee kan de huidige sensorwaarde uitgelezen worden.

Tabel 5.2: Provided interface

Het eerste onderdeel is de Ld2410Radar class. De Ld2410Radar class is een wrapper class om de

arduino-ld2410 bibliotheek. De wrapper class zorgt ervoor dat een radarsensor alleen uitgelezen kan worden. De wrapper class zorgt er vooral voor dat de ruwe data van de sensor op de correcte wijze geinteperteerd wordt. **De arduino-library doet dit niet correct!** De functie ”getCurrentTargetFrame” zorgt ervoor dat de ruwe data van de sensor geinterpeteerd wordt volgens de seriele protocol van de ld2410, te vinden op pagina 22 en 23, en retouneerd dit als een TargetFrameData. Dit is data volgens de seriele protocol in niet-engineering mode. De Ld2410Radar heeft meerdere publieke functies, maar alleen de belangrijkste is vermeld in het component diagram.

Het tweede onderdeel is de RadarLd2410Manager class. De RadarLd2410Manager class zorgt er voor dat meerder ld2410 sensoren uitgelezen kan worden en dit naar de /ld2410_targert_frames topic gepubliseert wordt. Dit process gebeurt elke seconde. De functie ”collectAndPublishRadarData” is hier verantwoordelijk voor.

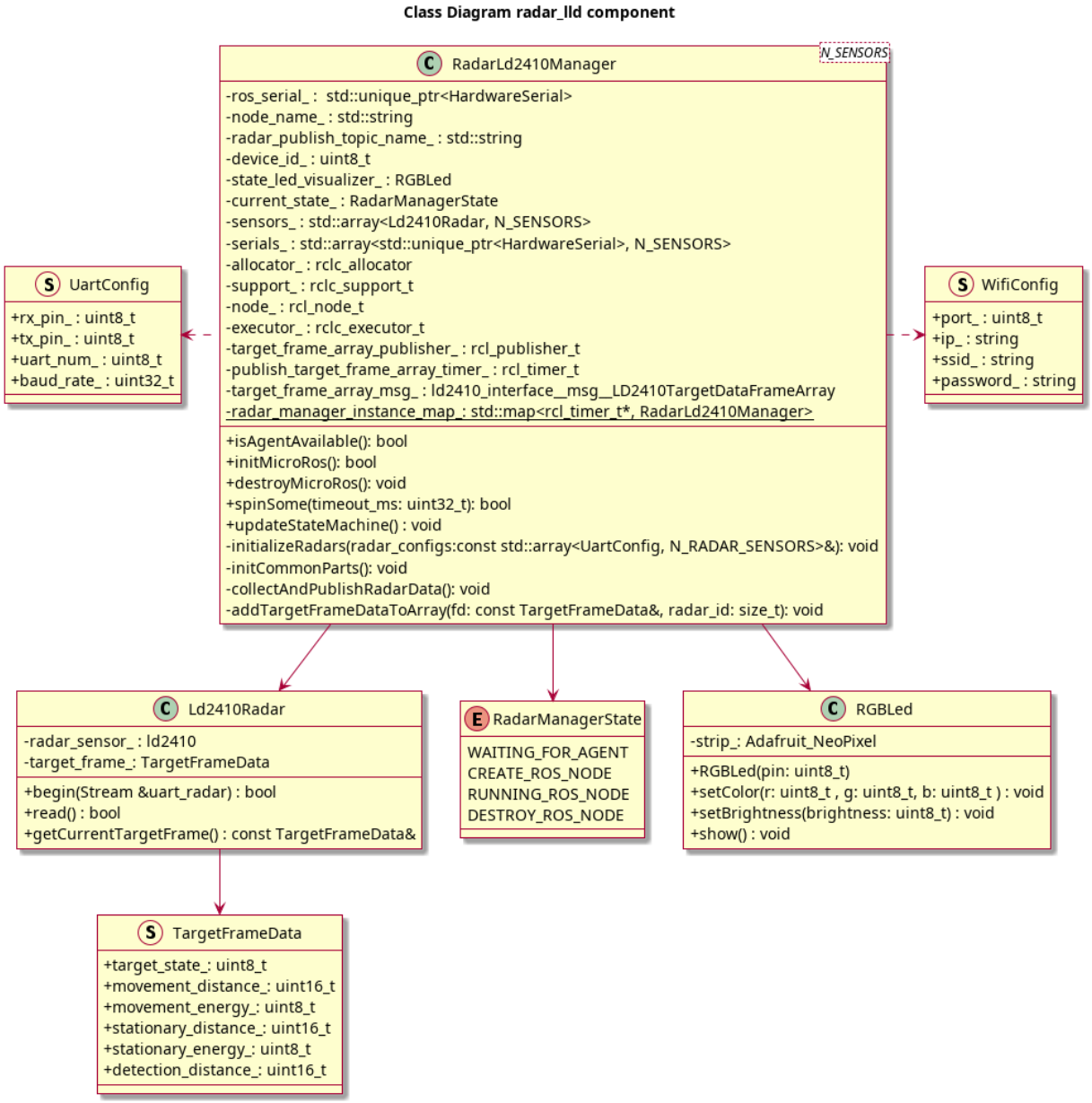
In Figuur 5.3 is het klasse diagram van radar_lld component te vinden. Hier zijn beide klasses uit het component diagram en de overige afhankelijke data structuren te vinden.

De RadarLd2410Manager class is ontworpen als een template class waarbij we als template argument opgeven hoeveel sensoren we willen gebruiken. De RadarLd2410Manager class kan alleen op een van de twee modus geïnstantieerd worden (niet tegelijk). Dit is in serieel modus of wifi modus.

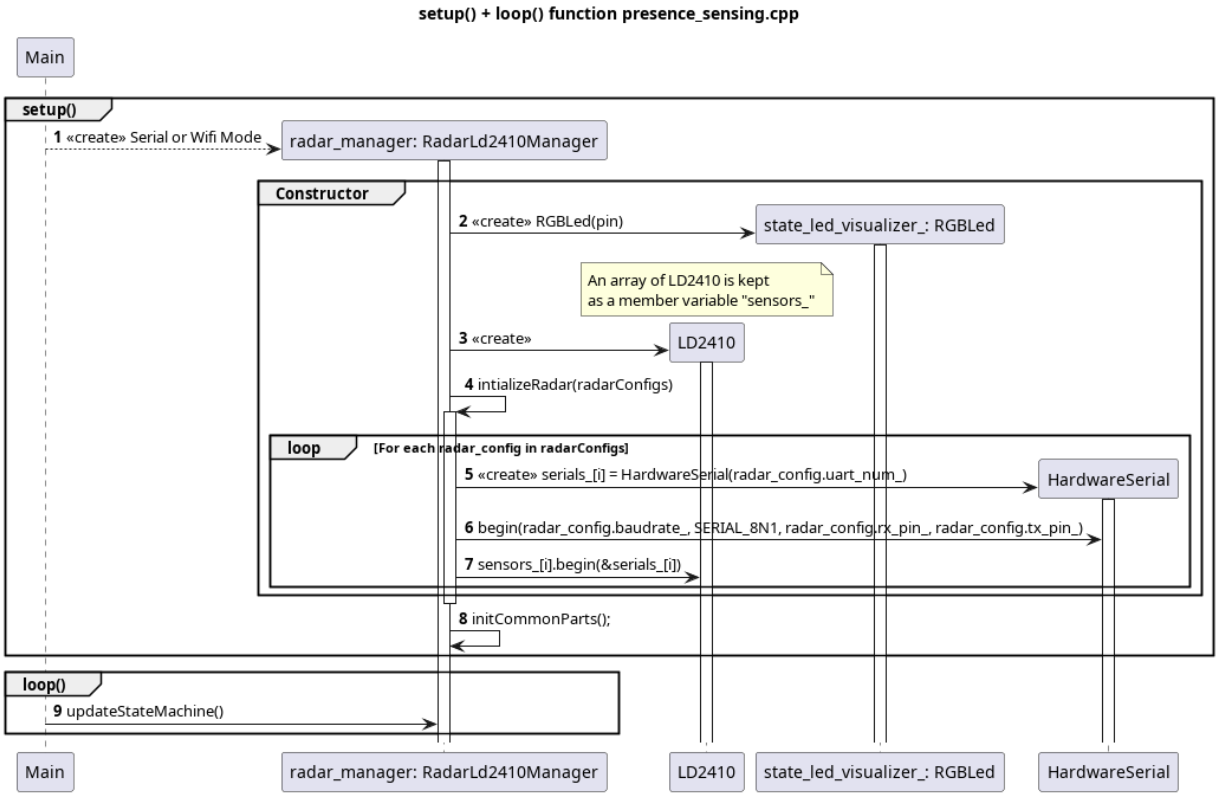
Met de constructor waarbij UartConfig struct een onderdeel is van de constructor argument wordt de RadarLd2410Manager class geïnstantieerd in serieel modus. Dit betekent dat het verzenden van ROS data via UART verloopt. Met de constructor waarbij WifiConfig struct een onderdeel is van de constructor argument wordt ROS data via WIFI verzonden.

In Figuur 5.45.4.a is te zien hoe de low level driver opgestart wordt. De programma voert in de main loop alleen maar de functie ”updateStatemachine” aan. In Figuur 5.45.4.b is de state diagram te zien dat ”updateStatemachine” uitvoert. De conditie en de actie die bij een overgang is in pseudo code genoteerd voor de leesbaarheid. Uit de statediagram is te zien dat de connectiviteit van de micro-ros client met de micro-ros agent en de interne statemachine van micro-ros bepaald in welke state de class zich bevindt. De interne statemachine van micro-ros wordt uitgevoerd met de methode ”spinSome”. Deze functie voert alle ”handelers” (subscribers,timers,publishers) die gekoppelt zijn aan de micro-ros node uit.

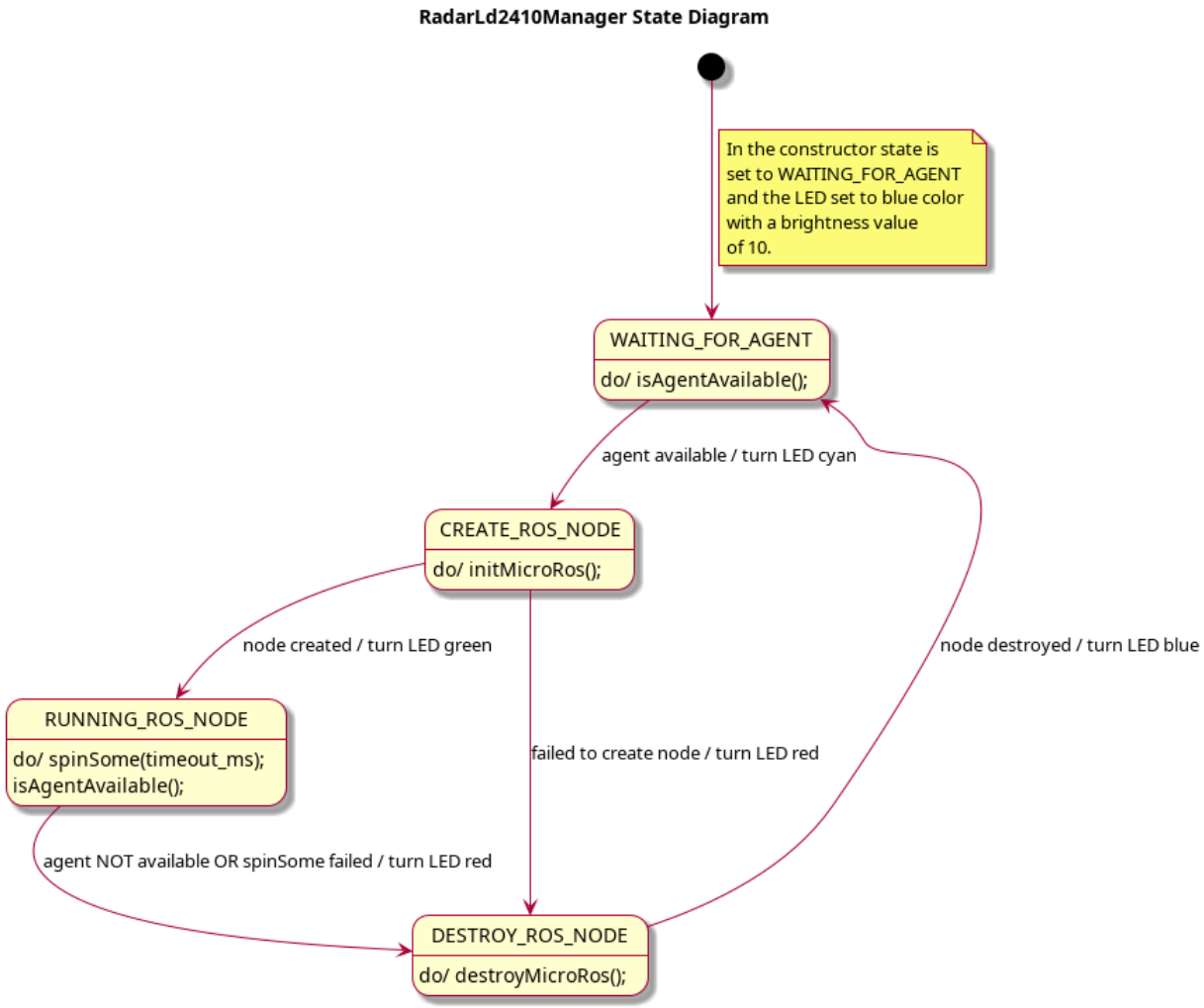
De mico-ros node heeft maar 1 publisher. Hieraan is de functie ”collectAndPublishRadarData” aan gekoppelt dat elke seconde uitgevoerd wordt. In Figuur 5.5 is te zien hoe dit gebeurd.



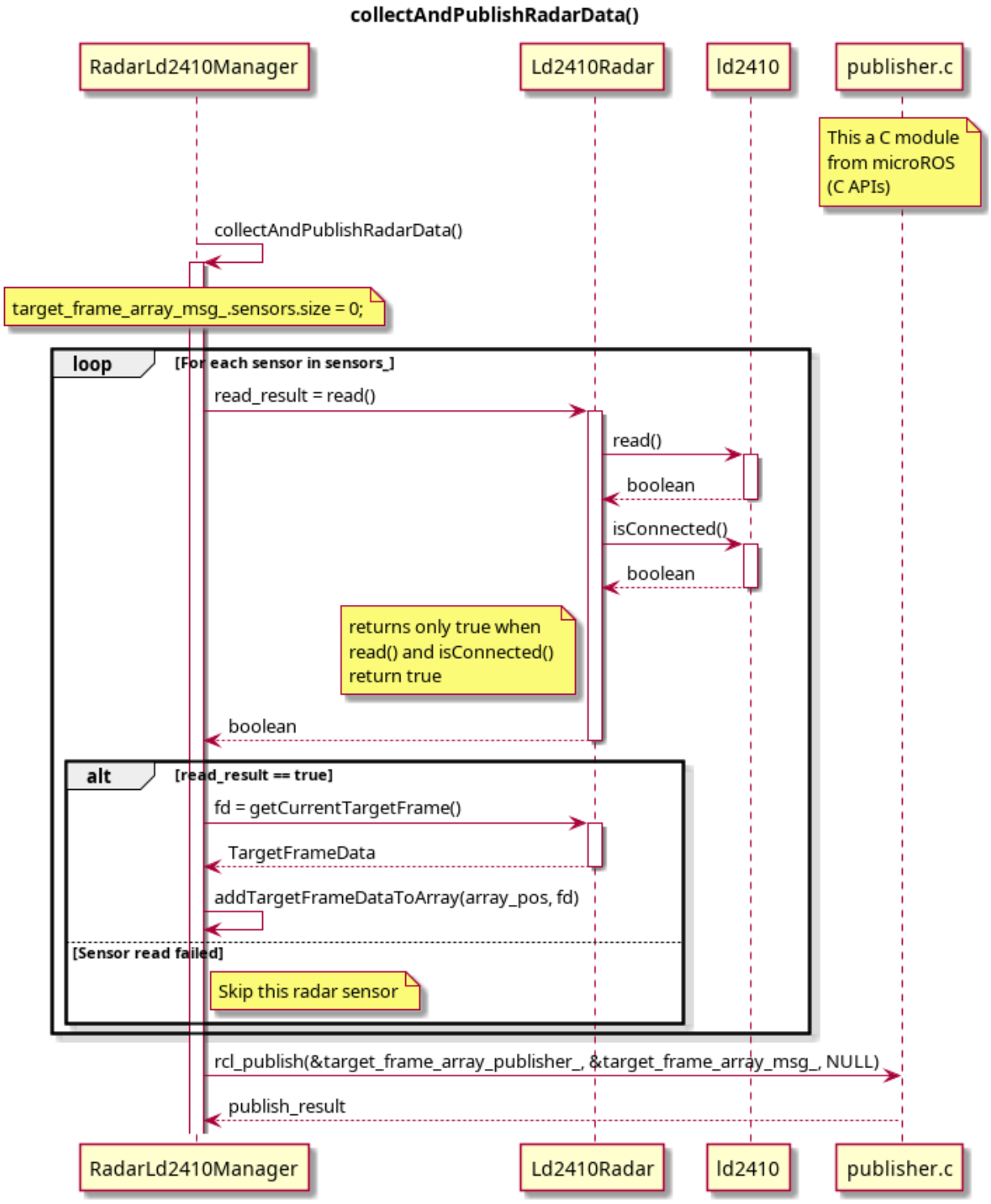
Figuur 5.3: Klasse diagram radar LLD.



(5.4.a) Sequence diagram start van hoofdprogramma.



(5.4.b) State diagram RadarLd2410Manager.



Figuur 5.5: Sequence diagram publiceren van LD2410 radar data.

5.1.3 ONTWERPBESLISSINGEN RADAR LLD

Gekozen voor hardwareserial voor radarsensoren, omdat software serial niet werkt met de standaard baudrate (256000) van de sensoren. Mogelijke optie nog uit te proberen is om de sensoren te configureren naar een lagere baudrate en dan kijken of het met softwareserial werkt. Hier was geen tijd voor. Omdat er nu voor hardwareserial hebben gekozen ben je afhankelijk van de microcontroller hoeveel radar sensoren je kan aansluiten. De esp32-s3 heeft 3 hardware UART controllers. 1 hiervan wordt gebruikt voor de seriele communciatie met de micro-ros agent. Dan blijven er maar twee dus over voor de

radar sensoren.

De seriele connectie tussen de esp32 en micro-ros agent moet met een hardwareserial. Er is getest met een softwareserial (baudrate 9600 en 115200) er kwam geen verbinding tot stand.

Er is gekozen om de RadarLd2410Manager class als een template class te maken, hiermee kan je simuleren dat je meerdere esp32 (elk template class representeerd een node) met meerdere sensoren data verstuurd.

Het bij houden van radar sensoren is met een vaste array afhankelijk van de template argument gedefinieerd. Omdat er geen sprake is van runtime sensoren toevoegen.

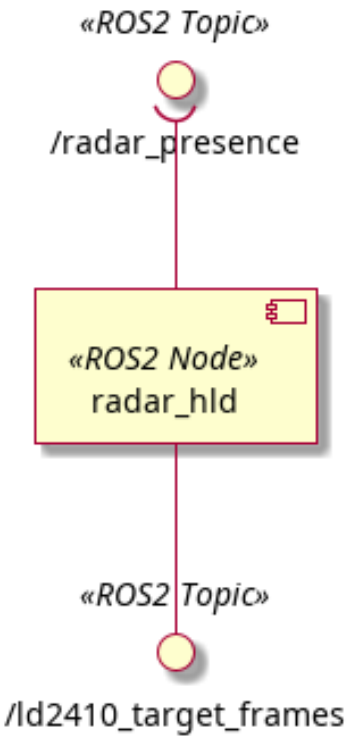
De wrapper class kan alleen een sensor uitlezen. Configureren hebben we erbuiten gelaten i.v.m. tijd. Hadden we voor nu niet nodig.

Constructor van RadarLd2410Manager met define gemaakt, omdat de onderliggen transport implementatie met een define beschikbaar zijn. Een andere opties is om een custom transport laag te maken waarmee geswitcht kan worden.

Voor de uart configuratie voor de serieele communicatie met de ros-agent is nog een edge case bedacht. Met de huidige UartConfig struct moet je altijd een tx en rx pin opgeven, echter is het vaak dat een esp32 een usb bridge heeft dat vast is aangesloten op een uart controller met de nummer 0. Om hiervan gebruik te maken moet je een UartConfig struct opgegeven aan de constructor waarbij de velden: uart_num_ , rx_pin_ en tx_pin_ de waarde 0 moeten hebben. Deze edge-case wordt alleen toegepast voor de ROS communicatie!

5.1.4 RADAR HLD

De high level driver bied 1 interface aan dat benaderd kan worden via de /ld2410_target_frames topic.



Figuur 5.6: Component diagram radar hld.

Topic naam	Functie naam	Omschrijving
/ld2410_target_frames	void radarPresen-ceCallback(const ld2410_interface::msg:: LD2410TargetDataFrameArray:: SharedPtr ra-dar_presence_msg)	Hiermee kan de high-level dri-ver alle sensor data verwerken en bepalen of iemand binnen het geconfigureerde gebied aanwezig is.

Tabel 5.3: Provided interface radar_hld

Listing 5.1: Definitie van LD2410TargetDataFrameArray.msg

```
# Custom ROS2 Message: LD2410TargetDataFrameArray.msg
uint8 device_id
LD2410TargetDataFrame[] sensors
```

Een LD2410TargetDataFrameArray bericht bevat een device_id en sensors. Met de device_id kunnen we achterhalen vanuit welk esp32 dit verstuurd is. In praktijk doen we voorlopig hier niks mee, omdat we maar 1 esp32-s3 hebben. De sensors variable bevat de ruwe data van alle aangesloten ld2410 sensoren waarvan een succesvolle ”read” operatie is uitgevoerd.

Listing 5.2: Definitie van LD2410TargetDataFrame.msg

```
# Custom ROS2 Message: LD2410TargetDataFrame.msg
uint8 rader_id

# Constants
uint8 NO_TARGET = 0
uint8 MOVING_ONLY = 1
uint8 STATIONARY_ONLY = 2
uint8 MOVING_AND_STATIONARY = 3

#raw sensor data
uint8 target_state
uint16 movement_distance
uint8 movement_energy
uint16 stationaty_distance
uint8 stationaty_energy
uint16 detection_distance
```

Een ld2410 sensor weergeeft ruwe informatie gedefinieerd onder ”raw sensor data”(in niet-engineer mode). Van deze informatie is een custom ros bericht gemaakt genaamd ”LD2410TargetDataFrame”. Hierbij is een radar_id toegevoegd om te kunnen herleiden bij welk aangesloten sensor deze waarde hoort. In de high level driver doen we momenteel hier niks mee. Er zijn ook constantes gedefinieerd in het bericht. Deze constantes zijn waardes die horen bij de target_state variabele.

In Figuur 5.7 is het klasse diagram van de high level driver te vinden dat bestaat uit 1 klasse die de ROS node representeert. De klasse bevat een subscriber om ”LD2410TargetDataFrameArray” data te ontvangen en een publisher om ”PresenceDetecion” berichten te versturen. De definitie van ”Presence-Detecion” is te vinden in Listing 5.3.

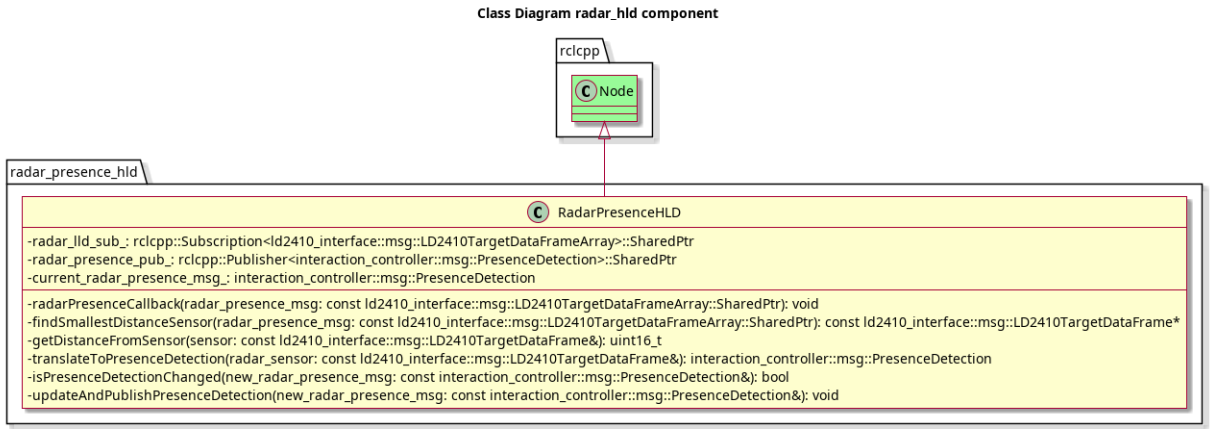
In Figuur 5.8 is te zien hoe berichten uit de /ld2410_target_frames topic verwerkt worden en hoe dit lijdt tot het versturen van een “PresenceDetecion“ bericht.

We proberen eerst met de functie “findSmallestDistanceSensor“ een sensor uit de array lijst te vinden die de kleinste afstandswaarde heeft en als target_state geen “NO_TARGET“ waarde heeft. Dit betekent dan dat de gedetecteerde afstandsmeting voor ons een valide meting is.

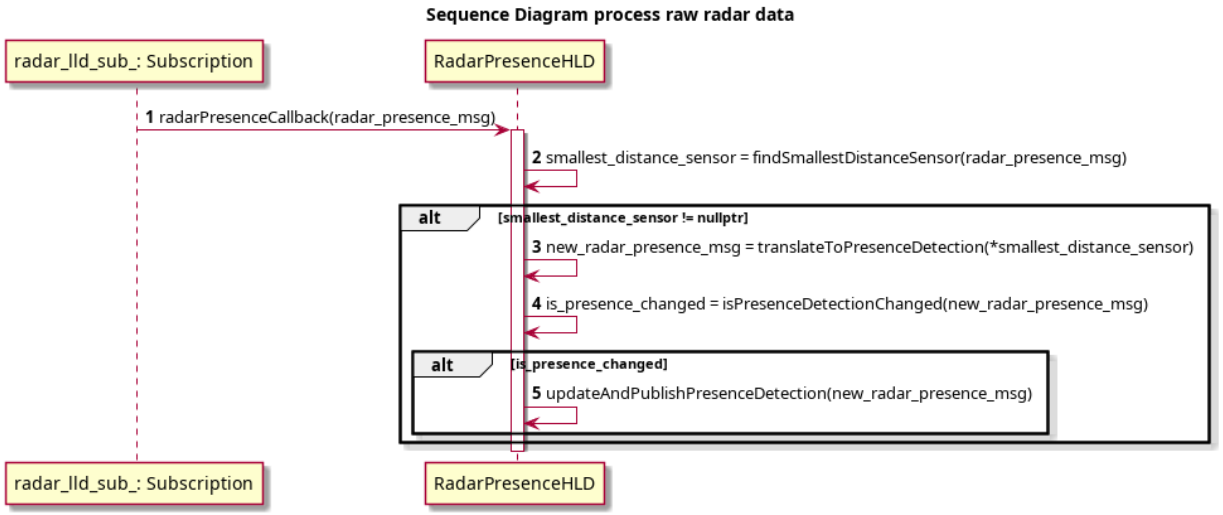
Wanneer we een sensor gevonden hebben vertalen we dit bericht naar een “PresenseDetection“ bericht met de functie “translateToPresenceDetection“. Deze functie controleert ook of de afstandsmeting onder of boven onze ingestelde grenswaarde zit. Als we er onder zitten dan wordt presence_state waarde van het “PresenceDetection“ bericht op “TARGET_IN_RANGE“ gezet anders wordt het op “TARGET_OUT_OF_RANGE“ gezet. Daarnaast wordt de target_state variable van “PresenceDetection“ gezet op “TARGET_STANDING“

wanneer de target_state van een sensor “STATIONARY_ONLY“ aangeeft. In alle andere gevallen wordt de target_state waarde van “PresenceDetecion“ op “TARGET_MOVING“ gezet. Het is volgens de flow van de callback functie niet mogelijk om een “NO_TARGET“ waarde te hebben in “translateToPresenceDetecion“, echter is hier wel rekening mee gehouden. Wanneer een sensor waarde toch wel deze waarde heeft wordt er een PresenceDetecion bericht geretouneerd met de zelfde waarde als de member class variable ”current_radar_presence_msg_“.

Vervolgens controleren we of onze nieuwe PresenceDetection bericht verschilt met de huidige, current_radar_presence_msg_, PersenceDetecion bericht. Wanneer een verandering is in het bericht updaten we onze huidige/opgeslagen PresenceDetection bericht en publiseren we dit naar de topic /radar_presence.



Figuur 5.7: Klasse diagram radar HLD.



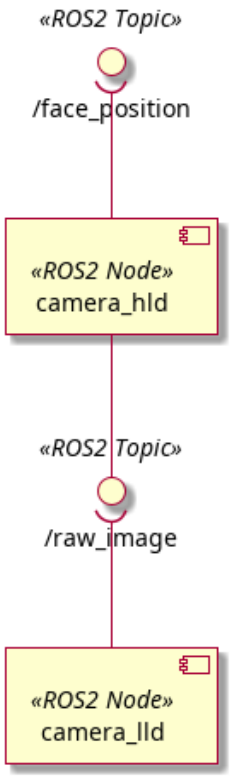
Figuur 5.8: Sequence diagram aanwezigheids detectie.

5.1.5 ONTWERPBESLISSINGEN RADAR HLD

<Op dit moment geen>

5.2 CAMERA COMPONENT

In dit hoofdstuk wordt de low en high level driver van het camera component toegelicht.



Figuur 5.9: Component diagram camera.

5.2.1 GERELATEERDE REQUIREMENTS EN RICHTLIJNEN

INITIELE REQUIREMENTS

Voor de opdrachtgever is het gewenst om een 2D positie van het gezicht te weten. Zodat we met de ogen naar het gezicht kunnen kijken.

NIEUWE REQUIREMENTS

Het is ook gewenst om de afstand van een gezicht tot de camera te weten. Op basis hiervan willen we de pupillen van robot verwijden of vernauwen. Het camera component zou dus een x y z positie van het gezicht moeten geven t.o.v. het camera. Hiermee kan uiteindelijk de ogen van de robot naar de juiste positie kijken en de pupillen verwijden.

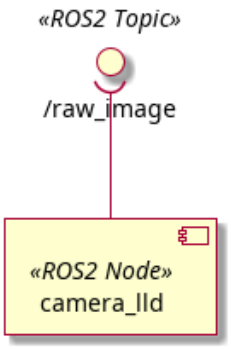
RICHTLIJNEN

De volgende richtlijnen zijn gehanteerd voor de ontwikkeling van de camera low level en high level driver componenten.

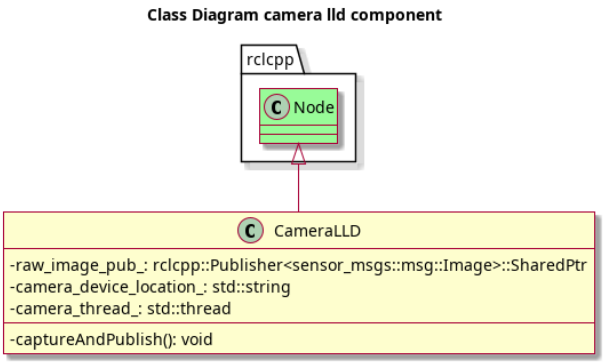
Richtlijn	Omschrijving
Programmeertaal	De software is geschreven in C++17.
Coding guidelines	Voor beide componenten wordt de ROS2 Jazzy coding style guide gebruikt.
Codekwaliteitscontrole	Code wordt gecontroleerd met ROS2 linter en cppcheck.
Compiler instellingen	De software wordt gecompileerd met: `-Wall -Wextra -Wpedantic`.
Afhankelijkheid	MediaPipe BlazeFace model wordt gebruikt in de high level driver voor het herkennen van een gezicht. Hiervoor wordt de wrapper class FaceDetector gebruikt. MediaPipe Iris model wordt gebruikt in de high level driver om afstand tot het gezicht te berekenen door de iris als referentie te gebruiken. Hiervoor wordt de wrapper class IrisMesh gebruikt.

5.2.2 CAMERA LLD

De camera low level driver is de meest simpele component van het gehele applicatie. Zo simpel dat er geen sequence diagram voor nodig is. Het leest een plaatje uit de camera en stuurt dit z.s.m. door naar de /raw_image topic. Het uit lezen en versturen van de ruwe plaatjes wordt gedaan met de functie ”captureAndPublish”. Deze functie wordt uitgevoerd in een aparte thread, omdat de functie een while loop bevat dat het lezen en versturen continue uitvoert.



Figuur 5.10: Component diagram camera lld.



Figuur 5.11: Klasse diagram camera lld.

5.2.3 ONTWERPBESLISSINGEN CAMERA LLD

Er is gekozen om een camera thread aan te maken waarin het uitlezen en versturen van plaatje in een while loop gebeurd. Hiermee sturen we ruwe plaatjes zo snel mogelijk. Een andere alternatief zou kunnen zijn om een timer aan te maken dat elke keer het uitlezen en versturen van plaatjes uitvoert, maar dan zit er wel een vaste interval voor dit proces.

5.2.4 CAMERA HLD

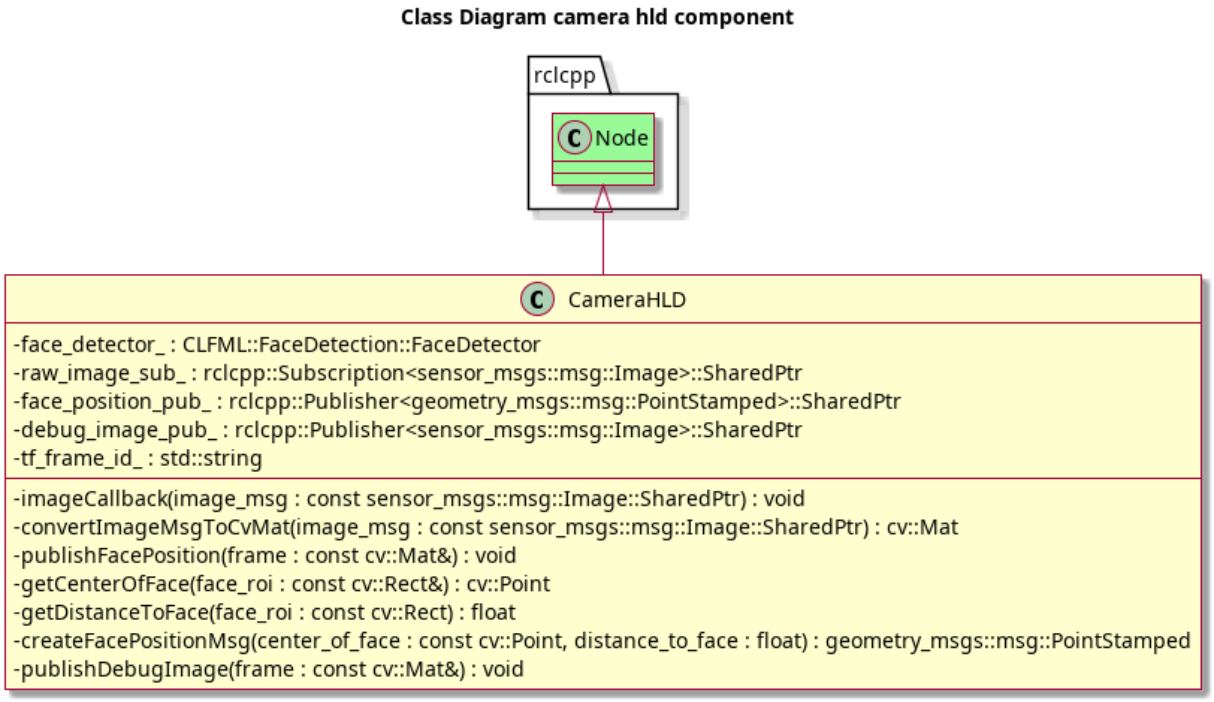
De camera high level driver zorgt er voor dat er een x y z positie van het middelpunt van een gezicht relatief tot de camera frame wordt gepubliseerd naar de /face_position topic volgens het ROS2 coördinaten stelsel.

Figuur 5.12: Component diagram camera hld.

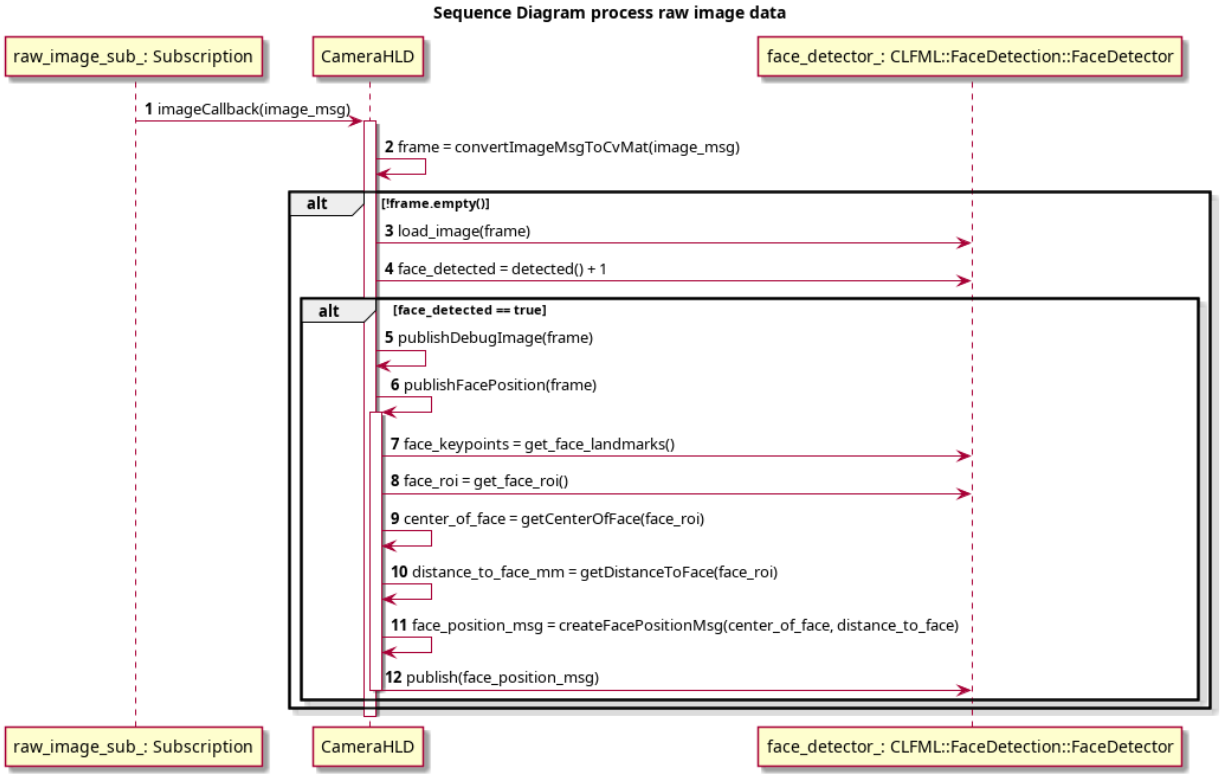
Topic naam	Functie naam	Omschrijving
`/raw_image`	void imageCallback(const sensor_msgs::msg::Image:: SharedPtr image_msg)	Hiermee kan de high-level driver op een ruwe plaatje gezichtsdetectie toepassen om een dichts bij zijnde x y z positie van een zicht te berekenen.

Tabel 5.5: Provided interface camera_hld

In Figuur 5.14 is te zien hoe de positie van een gezicht wordt bepaald. Wanneer een gezicht is gevonden met de “detect“ functie van FaceDetector vragen we de gezichtspunten en de regio van het gezicht op. Met de regio van de gezicht kunnen we het middelpunt van het gezicht bepalen in 2D. Met een bekende brandpunstafstand, breedte van het gezicht en een gemeten breedte van het gezicht kunnen we de afstand naar het gezicht berekenen. Dit gebeurd in de functie getDistanceToFace. Met een 2D coordinaat + afstand hebben hebben we infeite de x y z waarde van een gezicht. Met de functie “createFacePositionMsg“ zetten we de coordinaten van het camera coördinaten stelsel om naar het ROS2 coördinaten stelsel. De coördinaten worden als meters gepubliceerd.



Figuur 5.13: Klasse diagram camera hld.



Figuur 5.14: Sequence diagram gezichts positie vinden .

5.2.5 ONTWERPBESLISSINGEN CAMERA HLD

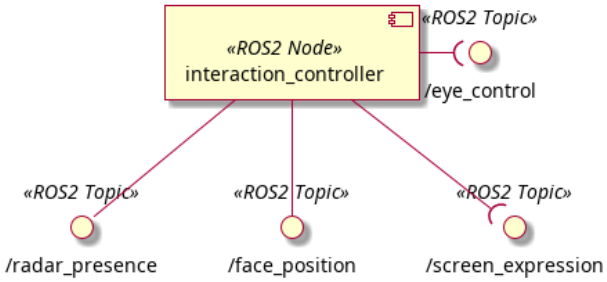
Gekozen om positie van gezicht in ROS2 coordinaten stelsel te publisheren i.p.v. camera coordinaaten stelsel. Dit heeft twee redenen. In de controller wil de ik positie relatief t.o.v. camera omrekenen naar

positie t.o.v. ogen. Het transformeren hiervan maakt het makkelijk als het in de input al in ROS2 coordinaten stelsel is. Ten tweede is coördinaten publiseren in ROS2 coördinaten stelsel de (verwachte) standaard binnen ROS.

Om de afstand van het gezicht relatief tot de camera te brekenen was initieel het idee om de breedte van het hoofd als referentie te gebruiken. Als de bounding box om het gezicht groter wordt dan weten we dat de gebruiker dichtbij staat. Op advies van de opdracht gever is er een betrouwbaardere referentie punt dan de breedte van de gezicht. Dit is namelijk de breedte van de iris, omdat dit voor elk geslacht, leeftijd en ras het zelfde is voor volwassenen. De breedte van de iris is 11.7 mm. Helaas is deze implementatie niet gelukt en zijn we terug gestapt naar de breedte van de gezicht.

5.3 INTERACTION CONTROLLER

Een controller maakt op basis van sensor inputs de keuze welke actuatoren op welke wijze aangestuurd moeten worden. In ons geval hebben we een controller genaamd “interaction_controller“ dat zorgt voor de interactie met een gebruiker. Op basis van radar en camera input laten we de oog displays een actie uitvoeren.



Figuur 5.15: Component diagram interaction controller.

Topic naam	Functie naam	Omschrijving
`/radar_presence`	void radarPresence-Callback(const interac-tion_controller::msg:: Presen-ceDetection:: SharedPtr pre-sence_msg)	Met behulp van een Presen-ceDetection bericht kan de controller bepalen of een per-soon binnen ons geïnteres-seerd gebied aanwezig is. Op dit moment kijken we al-leen of iemand “IN_RANGE“ of “OUT_OF_RANGE“ is. Op basis hier kan de controller be-palen of de ogen van de robot open of dicht moet.
`/face_position`	void facePositionCall-back(const geome-try_msgs::msg::PointStamped:: SharedPtr face_position)	Met behulp van deze callback functie kan de controller bepa-len waar een gezicht staat t.o.v. de ogen en deze die kant op laten kijken. De gezichtspositie komt binnen als een 3D coördi-naat t.o.v. een camera (frame).

Tabel 5.6: Provided interfaces ineteraction_controller

Listing 5.3: Definitie van PresenceDetection.msg

```
#PresenceDetection.msg

#constanst for target_state
uint8 TARGET_MOVING = 0
uint8 TARGET_STANDING = 1

uint8 target_state

#constants for presence_state
uint8 TARGET_OUT_OF_RANGE = 0
uint8 TARGET_IN_RANGE = 1

uint8 presence_state
```

5.3.1 GERELATEERDE REQUIREMENTS EN RICHTLIJNEN

INITIËLE REQUIREMENTS

De robot moet zijn ogen kunnen open en sluiten op basis van radar input en een persoon kunnen volgen met zijn ogen op basis van camera input.

RICHTLIJNEN

Richtlijn	Omschrijving
Programmeertaal	De software is geschreven in C++17.
Coding guidelines	Voor beide componenten wordt de ROS2 Jazzy coding style guide gebruikt.
Codekwaliteitscontrole	Code wordt gecontroleerd met ROS2 linter en cppcheck.
Compiler instellingen	De software wordt gecompileerd met: `-Wall -Wextra -Wpedantic`.
Afhankelijkheid	Transformeren van coördinaten stelsel is met de TF2 bliblio-theek van ROS2 gerealiseerd.

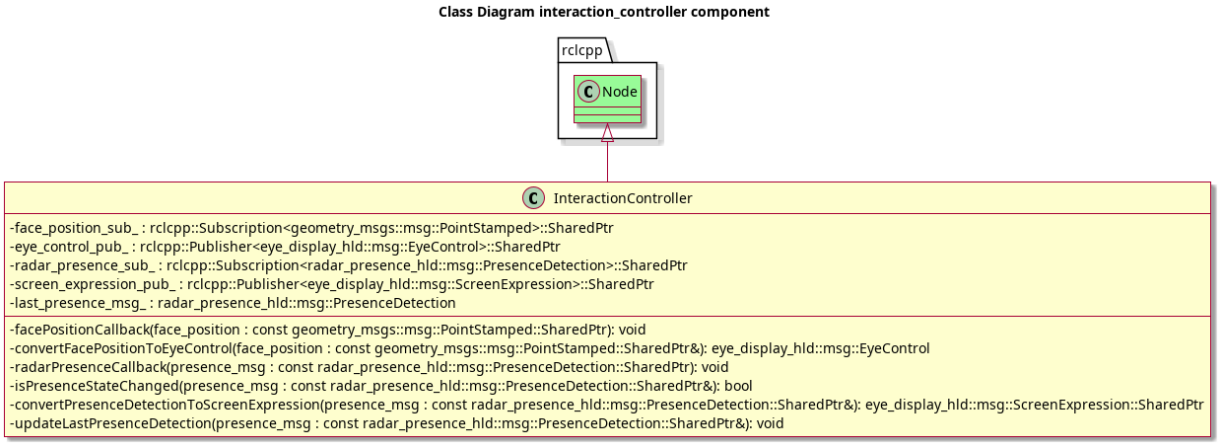
5.3.2 INTERACTIONCONTROLLER

De interaction_controller component bestaat maar uit 1 klasse genaamd ”InteractionController”. De klasse luistert naar /radar_presence topic met radar_presencen_sub_ en naar /face_position topic met face_postion_sub_.

Om ogen te openen of te sluiten publiceren we op de topic /screen_expression met screen_expression_pub_. Om de ogen naar een richting te laten kijken publiseren we op de topic /eye_control met eye_control_pub_. De constructor van deze klasse initialiseert last_presence_msg_ met de volgende waardes:

Listing 5.4: init last_presence_msg_

```
last_precence_msg_.presence_state = PresenceDetection::TARGET_OUT_OF_RANGE;
last_precence_msg_.target_state = PresenceDetection::TARGET_STANDING;
```

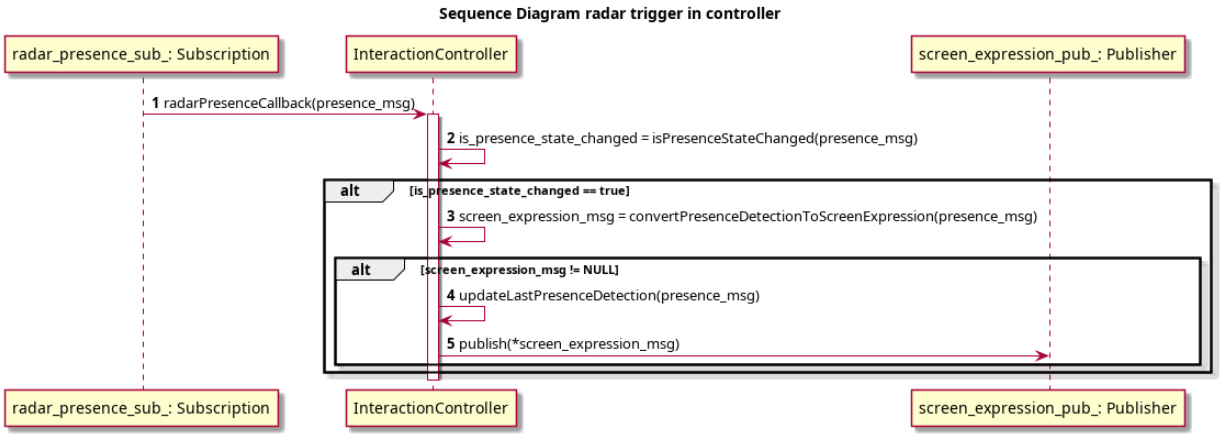


Figuur 5.16: Klasse diagram interaction controller.

In Figuur 5.17 wordt er weergeven hoe op basis van radar input de keuze gemaakt wordt of de ogen open of gesloten moet zijn. In de radarPresenceCallback functie controlleren we eerste of de binnen

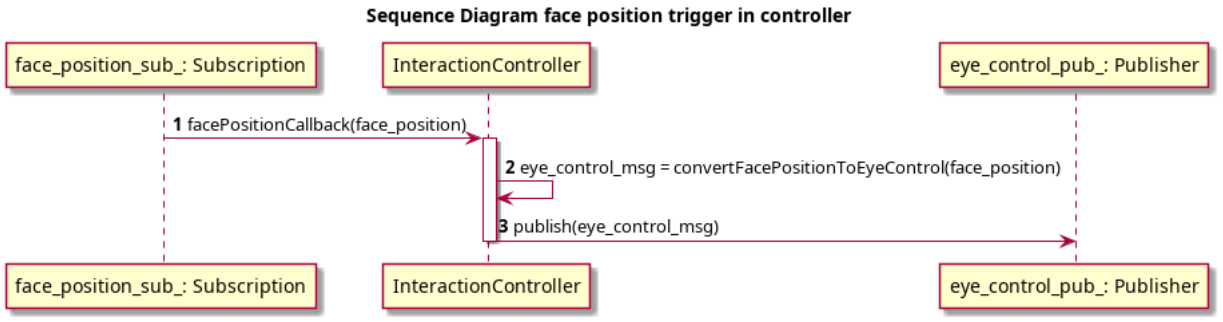
komen de bericht anders is dan laatst onthouden “PersenceDetecion“ bericht. Wanneer dit het geval is vertalen we dit bericht naar een ScreenExpression bericht. Dit gebeurd met de functie “convertPresenceDetectionToScreenExpression“. In de functie kijken we afhankelijk van de presence_state variable uit het “PresenceDetection“ bericht wat voor actie op de oog display uitgevoerd moet worden. Bij een “IN_RANGE“ state is de actie van “ScreenExpression“ dat de ogen open moeten. Bij een “OUT_OF_RANGE” state is de actie van “ScreenExpression“ dat de ogen moeten sluiten. Een ongedefinieerd presence_state zal lijden dat er geen actie uitgevoerd zal worden (nullptr).

Wanneer er bepaald is of ogen moeten openen of sluiten wordt eerst de last_presence_msg geupdate met inkomende “PresenceDetection“ bericht en vervolgens wordt dan naar de eye_display_hld verstuurd dat de ogen moeten openen of sluiten.



Figuur 5.17: Sequence diagram keuze bij radar input.

In Figuur 5.18 wordt er weergeven hoe een positie van een gezicht, dat door de camera_hld wordt bepaald, resulteert in een aansturing van de ogen. Het 3D punt wordt eerst getransformeerd naar een 3D punt t.o.v. van de midden van de ogen en vervolgens vertaalt naar een yaw en pitch hoek in radialen. Voor de afstand van de ogen naar het gezicht wordt de euclidische afstand berekent. Deze waarde wordt in centimeters opgestuurd.



Figuur 5.18: Sequence diagram aansturing ogen.

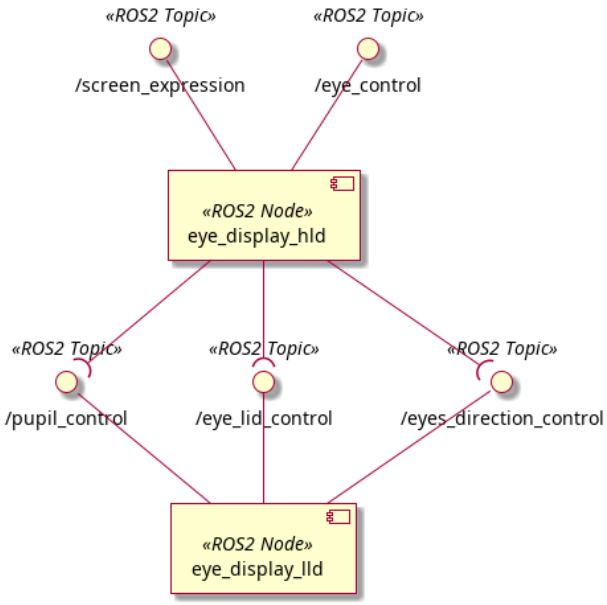
Letop! Het transformeren van een coördinatenstelsel wordt gerealiseerd met ROS TF2. De betekent dat de voorwaarde om een positie van een gezicht t.o.v. camera te vertalen naar de middel punt van de ogen, hiervan de relatie van TF frames beschikbaar moet zijn. We doen dit middels een externe static publisher node. Omdat we gebruik maken van TF2 zit de inhoud van facePositionCallback functie binnen een try-catch blok, omdat het mogelijk is om een exceptie te krijgen als geen transform frames tussen de camera en middelpunt van ogen beschikbaar is.

5.3.3 ONTWERPBESLISSINGEN INTERACTION CONTROLLER

Gekozen om de pitch en yaw hoeken in radialen te sturen i.p.v. graden. Binnen ROS2 is dit de aanbevolen “unit“ voor hoeken. Gekozen om aansturing van de ogen voor het volgen van een persoon te beredeneren vanuit de midden van de ogen, omdat de ogen toch de zelfde richting moeten opkijken.

5.4 EYE DISPLAY COMPONENT

In dit hoofdstuk wordt de high en low level driver van het eye component toegelicht.



Figuur 5.19: Component diagram eye display.

5.4.1 GERELATEERDE REQUIREMENTS EN RICHTLIJNEN

INITIELE REQUIREMENTS

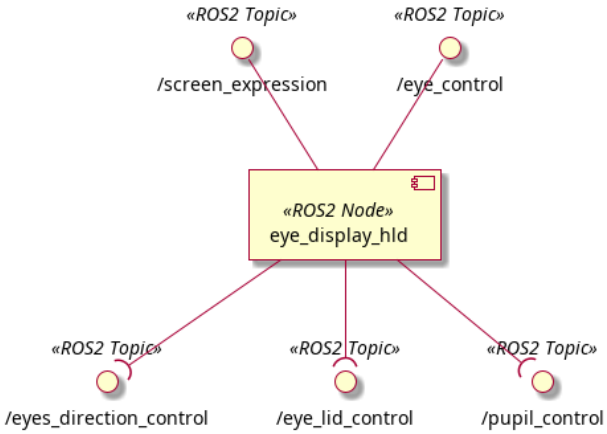
Voor de opdrachtgever is het gewenst om ogen te visualiseren op twee mini-display schermen. De ogen moeten een persoon kunnen volgen.

RICHTLIJNEN

Richtlijn	Omschrijving
Programmeertaal	De high level driver is geschreven in C++17. De low level driver is geschreven in javascript (ElectronJS)
Coding guidelines	Voor de high level driver wordt de ROS2 Jazzy coding style guide gebruikt.
Codekwaliteitscontrole	De high level driver wordt gecontroleerd met ROS2 linter en cppcheck.
Compiler instellingen	De high level driver wordt gecompileerd met: `-Wall -Wextra -Wpedantic`.
Afhankelijkheid	De low level driver wordt ontwikkelt met het Electron framework

5.4.2 EYE DISPLAY HLD

De high level eye display driver zorgt er voor dat commando's voor de ogen omgezet wordt naar bewegingen dat elk onderdeel van de oog moet uitvoeren.



Figuur 5.20: Component diagram eye display hld.

Topic naam	Functie naam	Omschrijving
/eye_control	void bothEyesCallback(const eye_display_hld::msg:: EyeControl:: SharedPtr eye_control_msg)	Met behulp van deze functie kan de high level driver bepalen in welke richting beide ogen moeten kijken en bepalen hoeveel de pupillen moet verwijden op basis van de afstand.
/screen_expression	void screenExpressionCallback(const eye_display_hld::msg:: ScreenExpression:: SharedPtr screen_expression_msg)	Met deze functie kan bepaald worden welke oog expressie op de display schermen weergeven wordt.

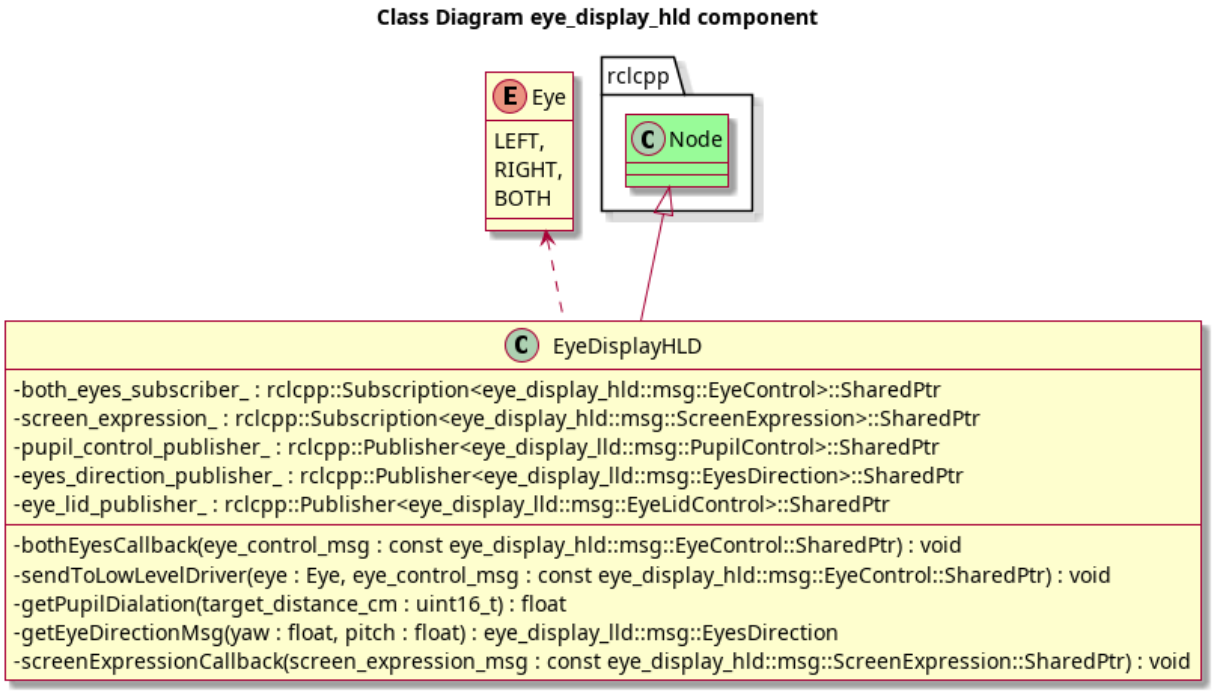
Tabel 5.9: Provided interfaces eye_display_hld

Listing 5.5: Definitie van EyeControl.msg

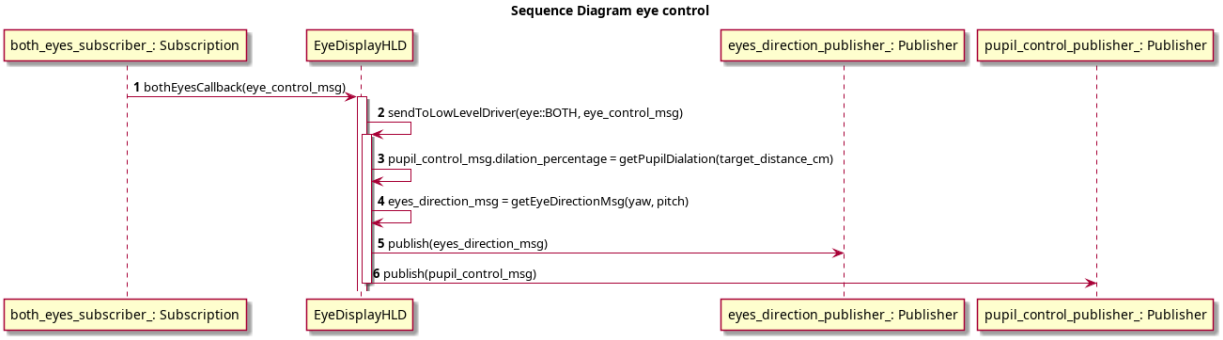
```
# EyeControl.msg
float32 yaw             # Horizontal angle in radian (positive = left, negative = right)
float32 pitch           # Vertical angle in radian (positive = up, negative = down)
uint16 target_distance_cm # Optional: distance in cm (e.g. for a target at a specific distance)
```

Listing 5.6: Definitie van ScreenExpression.msg

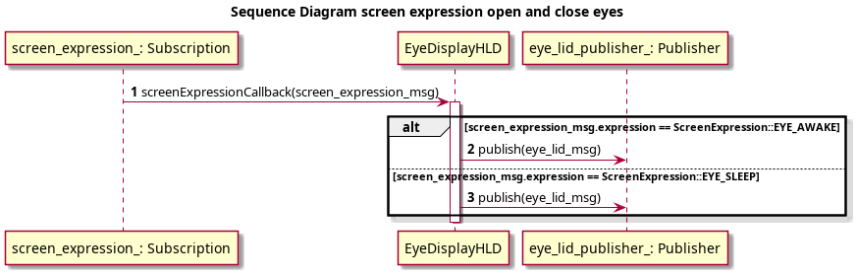
```
# ScreenExpression.msg
uint8 EYE_AWAKE      = 0
uint8 EYE_SLEEP      = 1
uint8 EYE_SLEEPY     = 2
uint8 EYE_GRIN       = 3
uint8 EYE_SAD        = 4
uint8 EYE_LOOK_RANDOM = 5
uint8 ANIMATE_LOVE   = 6
uint8 ANIMATE_STAR   = 7

uint8 action  # De expressie die moet worden weergegeven op het scherm
```



Figuur 5.21: Klasse diagram eye display hld.



Figuur 5.22: Sequence diagram aansturen ogen.

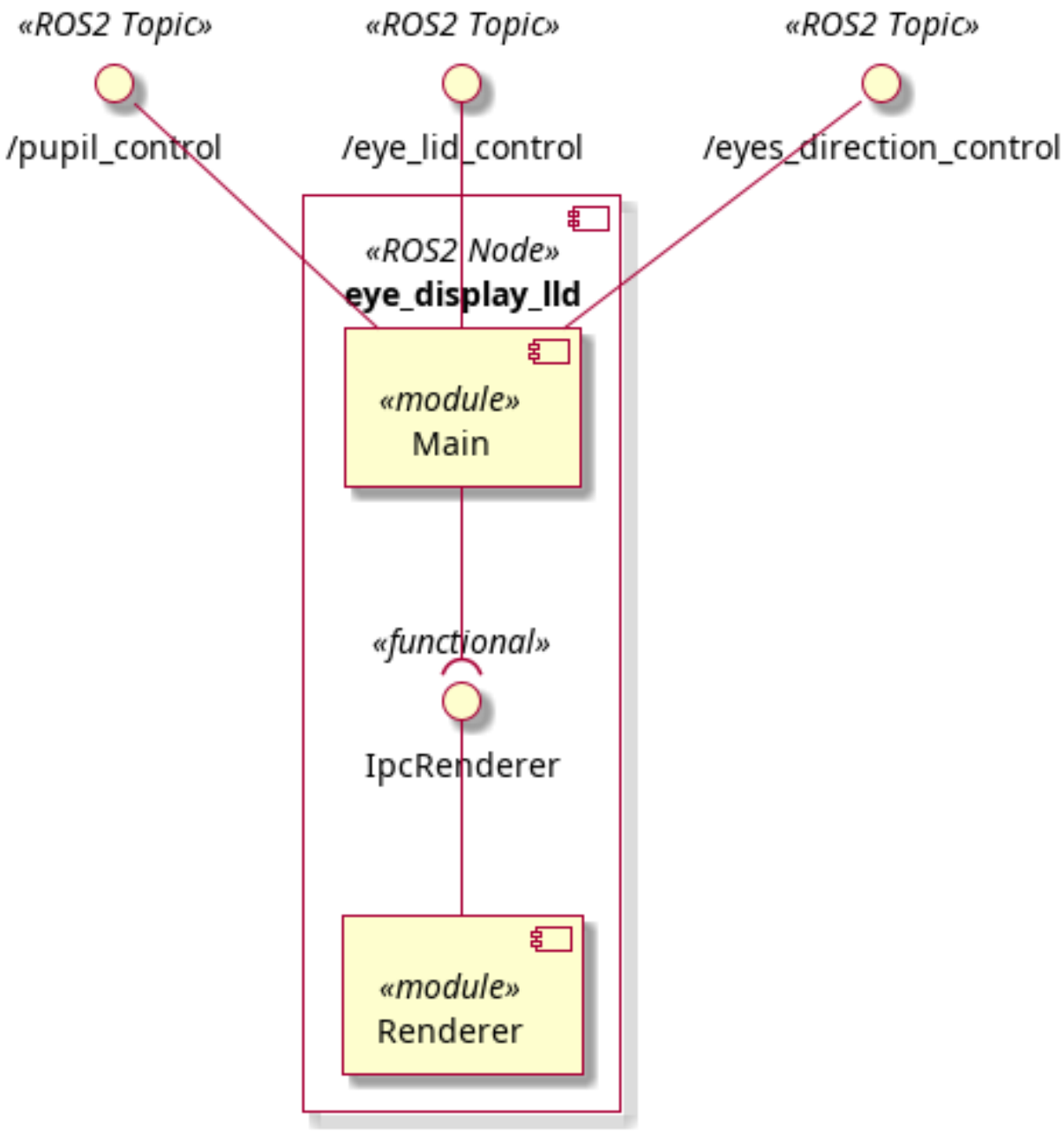


Figuur 5.23: Sequence diagram open en sluiten ogen.

5.4.3 EYE DISPLAY LLD

De low level eye display driver biedt de functionaliteit aan om de ogen te aansturen. Het component bestaat uit twee modules. Een module is een bestand waarin losse variabele en functies zijn gedefinieerd die alleen bereikbaar zijn binnen hun eigen module. Infeite is alles binnen een module in ons geval

“private“. De modules zijn twee aparte processen die draaien. De main module kan je zien als de backend/server van het component. Het maakt vensters aan voor de displays waar ogen op gevisualiseerd wordt en luistert naar binnen komende ROS berichten. De renderer module kan je zien als de frontend van het component. Een renderer module is gekoppeld aan een venster. De renderer module bevat functies dat direct een venster kan beïnvloeden. In Figuur 5.25 is een klasse diagram gemaakt dat de applicatie in Object georiënteerd stijl weergeeft.



Figuur 5.24: Component diagram eye display lld.

Om acties op een venster te kunnen uitvoeren moet de main module functies uit de renderer module kunnen aanroepen. We maken hiervoor gebruik van het Inter-process communication (IPC) functionaliteit van electron. Dit is in het component diagram weergeven met de interface “IpcRenderer“. De IPC communicatie hoe wij het gebruiken werkt als een pub-sub architectuur. De main verstuurt naar een bepaald kanaal een bericht. De renderer luister naar dit kanaal. In de onderstaande voorbeeld is een voorbeeld te zien hoe een ROS bericht van het /eyes_direction_control topic ervoor zorgt dat een functie uit de renderer module wordt aangeroepen.

Listing 5.7: main.js: Luisteren naar ROS bericht en doorsturen

```
node.createSubscription('eye_display_lld/msg/EyesDirection',
                        'eyes_direction_control', (msg) => {
  sendToRenderer('eyes_direction_control', msg);
});
```

We luisteren naar de topic “eye_direction_control“ topic. Wanneer we een bericht krijgen roepen we de functie sendToRenderer aan. We geven aan de functie aan dat het ROS bericht verstuurd moet worden naar het kanaal “eye_direction_control“.

Listing 5.8: renderer.js: Ontvangen bericht uit main en uitvoeren

```
ipcRenderer.on('eyes_direction_control', (event, eyes_direction) => {
console.log('Eyes direction yaw:', eyes_direction.yaw);
console.log('Eyes direction pitch:', eyes_direction.pitch);
moveEyes(eyes_direction.yaw, eyes_direction.pitch);
});
```

Wanneer we een bericht krijgen op het “eye_direction_control“ kanaal. Voeren we de functie moveEyes uit om de ogen naar een richting te laten kijken.

In het onderstaande tabel laten we de provided interfaces van eye_display_lld zien. We weergeven welke topic ervoor zorgt welke functie uit de render module aanroept.

Topic naam	Functie naam (renderer.js)	Omschrijving
/eye_direction_control	moveEyes(yaw, pitch)	Beide ogen worden hiermee naar een bepaalde kijkrichting getekend.
/eye_lid_control	moveLid(eye,top_lid_position, bottom_lid_position)	Met deze functie kan je bepalen van welke oog je de oogleden op welk positie wilt plaatsen. Een oog lid kan je zien als een slider dat je over oog kan schuiven. Een top_lid_position en bottom_lid_position met een waarde van 0 betekent dat de oog leden op hun begin positie staan en dus de ogen helemaal open is. Een waarde van 50 voor beide betekent dat ze allebei de helft van de oog innemen wat resulteert dat je een gesloten oog hebt.
/pupil_control	updatePupilDilation(dilation_percentage)	Met deze functie kan bepaald worden hoe groot de pupillen getekend moeten worden voor beide ogen. Het gaat om percentages relatief tot de grootte van de iris.

Tabel 5.10: Provided interfaces eye_display_lld

Listing 5.9: Definitie van EyeLidControl.msg

```
#EyeLidControl.msg

uint8 LEFT_EYE = 0
uint8 RIGHT_EYE = 1
uint8 BOTH_EYES = 2
```

```
uint8 eye_id # The id of the eye to control, should have one of the values above
float32 top_lid_position # The position of the upper lid, should be between 0 and 100.
float32 bottom_lid_position # The position of the lower lid, should be between 0 and 100.
```

Listing 5.10: Definitie van EyesDirection.msg

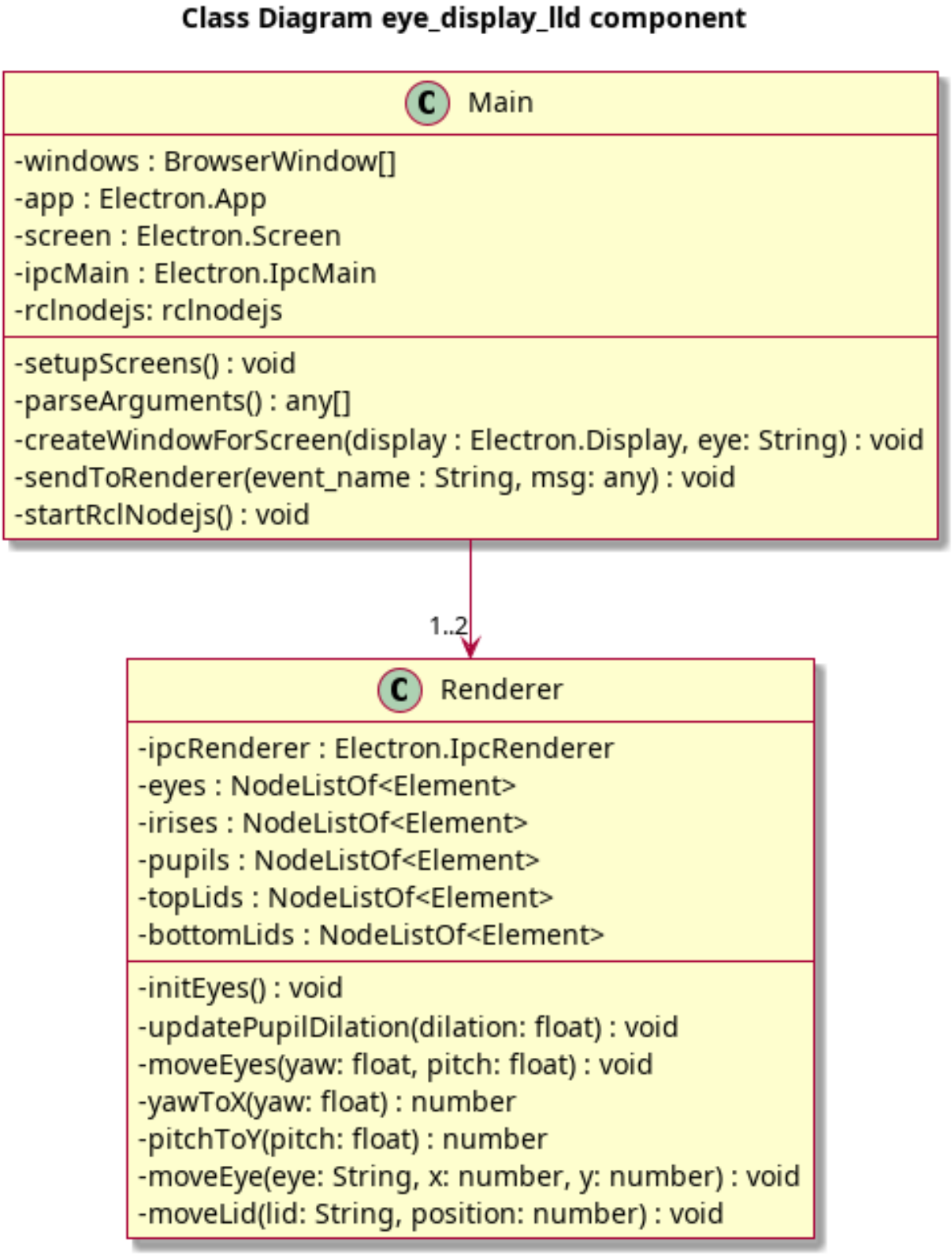
```
#EyesDirection.msg
float32 yaw             # Horizontal angle in degrees (positive = left, negative = right)
float32 pitch           # Vertical angle in degrees (positive = up, negative = down)
```

Listing 5.11: Definitie van PupilControl.msg

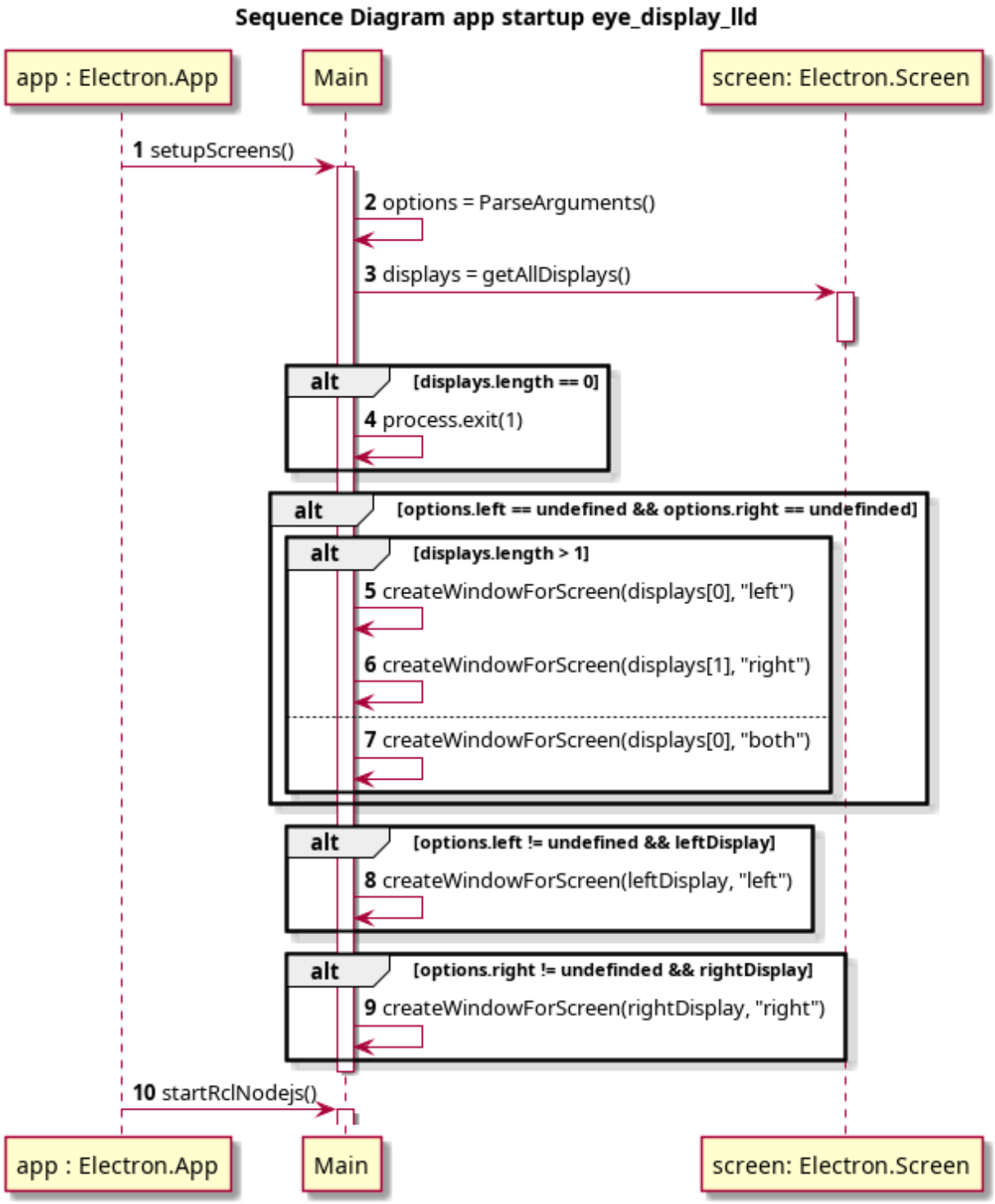
```
# PupilControl.msg
float32 dilation_percentage  # Pupil dilation percentage (0.0 to 100.0)
```

De werking van de sendToRender functie is te vinden in het sequence diagram 5.27. In 5.26 is te zien hoe bepaald wordt hoeveel venster er getekend moet worden met hoeveel ogen. We kijken of we eerst argumenten mee hebben gekregen bij het opstarten van het programma. Als argumenten kunnen we op geven op welk scherm welke oog getekend moet worden. Ook kunnen we met argumenten kiezen of alleen een linker of rechter oog getekend moet worden. Dit gebeurd in ParseArguments functie. Deze functie sluit de applicatie direct af als meer ogen configureert dan dat er schermen aanwezig zijn. Bij geen configuratie gaat de main module ervanuit dat je op het eerste scherm het linker oog wilt en op het tweede scherm de rechter oog.

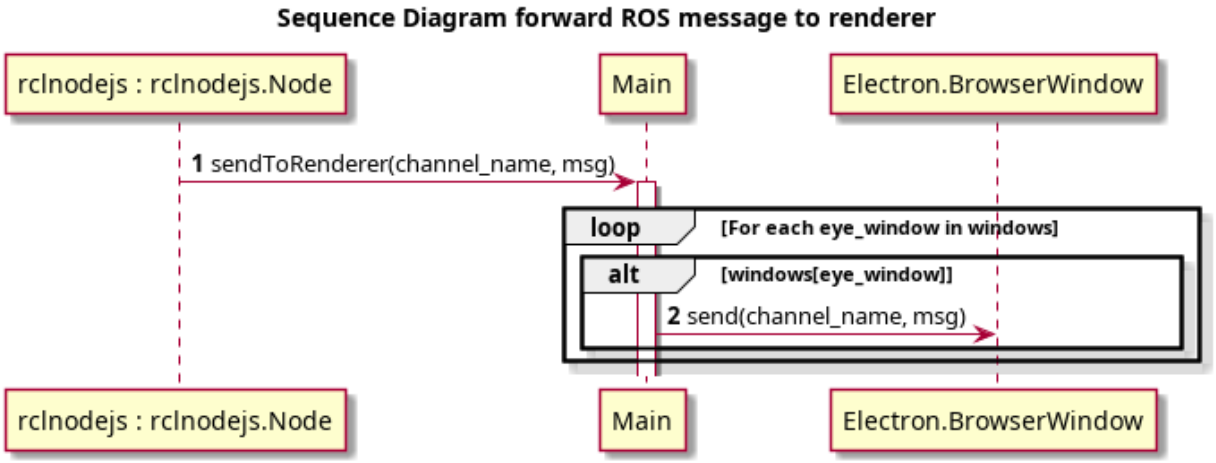
Het detecteren hoeveel schermen we hebben wordt gedaan met de getAllDisplays functie. Als er geen schermen zijn aangesloten sluit applicatie direct af. Wanneer er configuratie is opgegeven en er zijn meer dan 1 scherm aangesloten dan worden de ogen getekend zoals eerder beschreven. Anders worden beide ogen op een scherm getekend waarbij de rechter helft van de het beeldscherm de linker oog krijgt en de linker helft van het scherm de rechter oog.



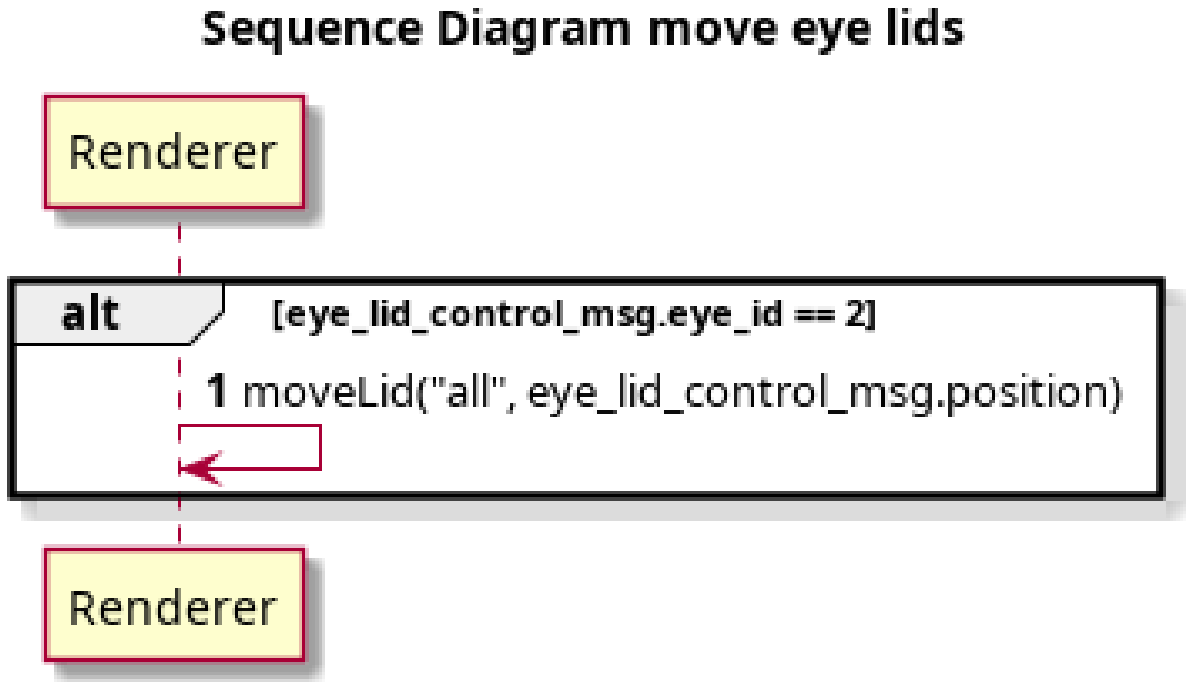
Figuur 5.25: Klasse diagram eye display lld.



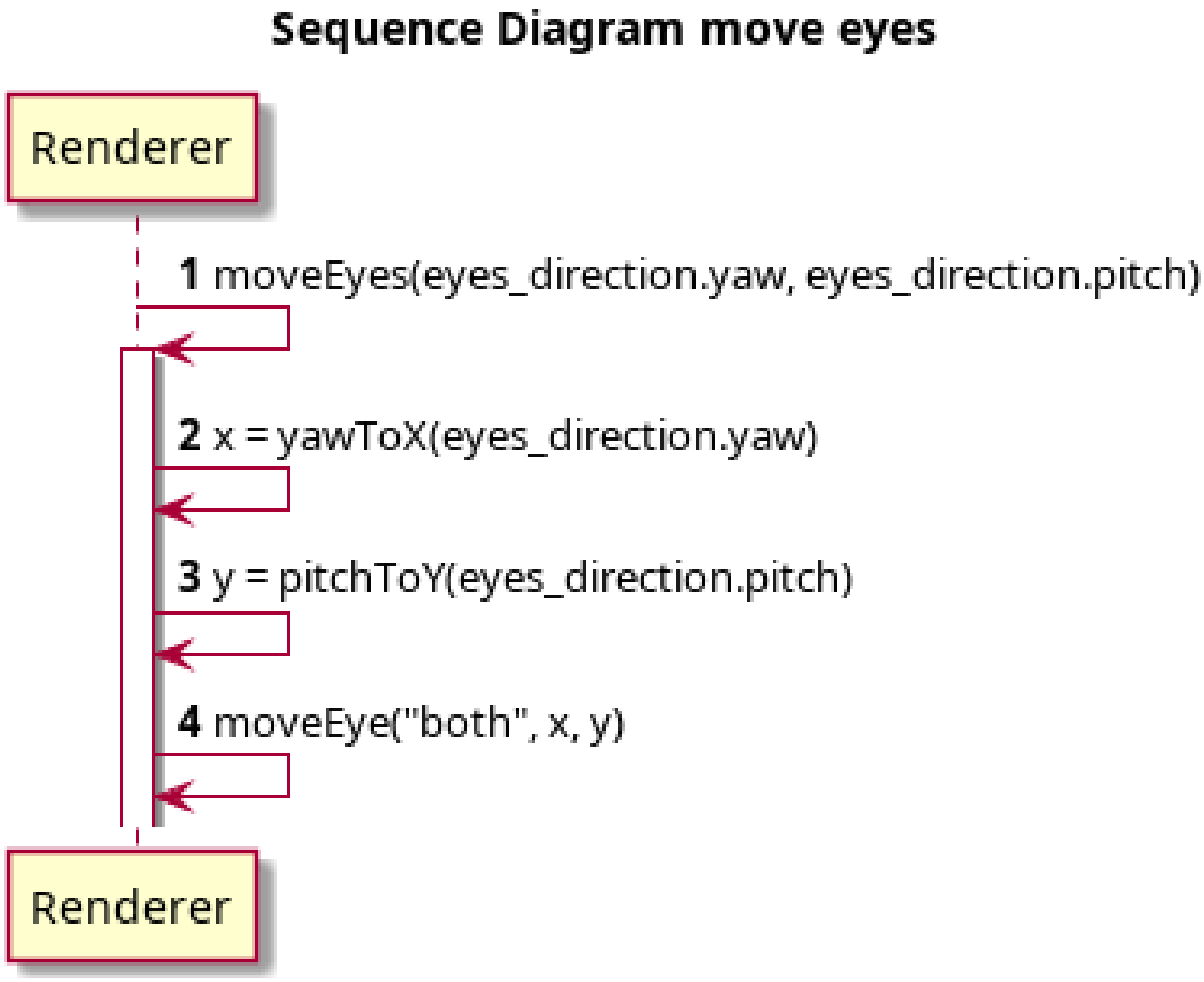
Figuur 5.26: Sequence diagram opstarten eye lld.



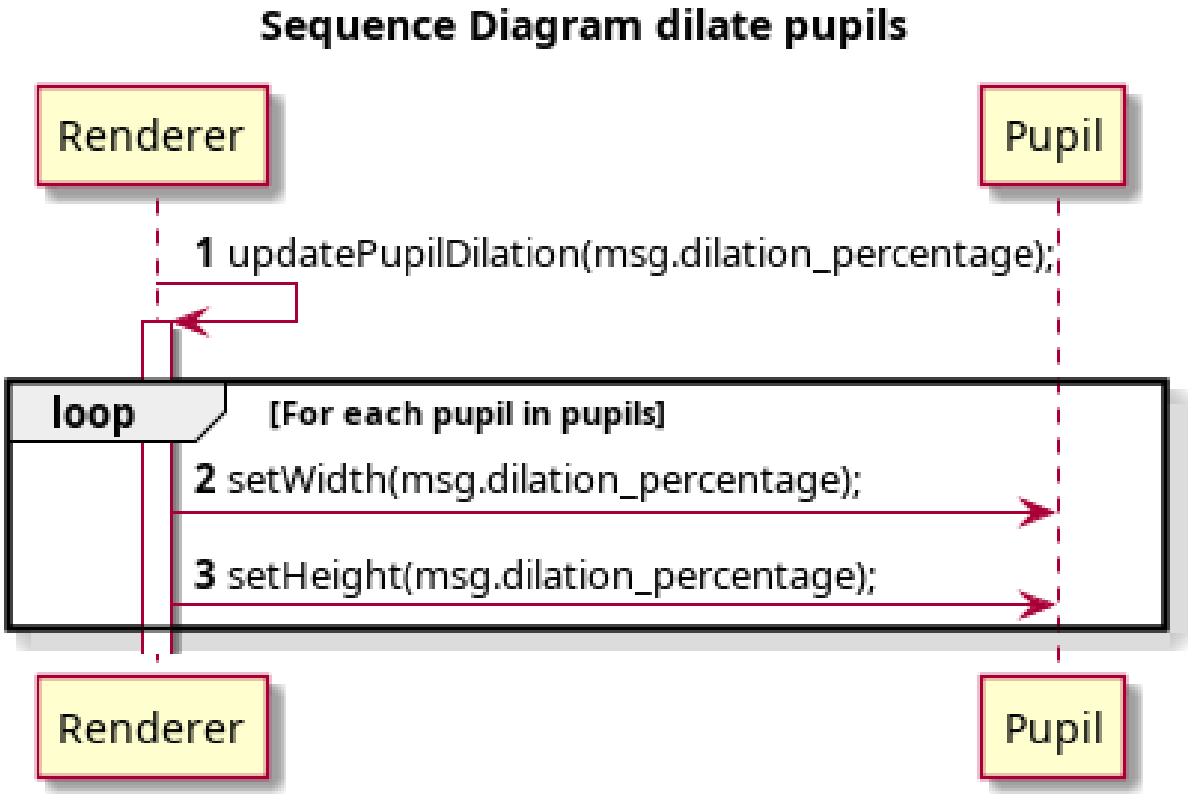
Figuur 5.27: Sequence diagram ROS berichten naar renderer module.



Figuur 5.28: Sequence diagram oogleden bewegen.



Figuur 5.29: Sequence diagram oog richting bewegen.



Figuur 5.30: Sequence diagram oog richting bewegen.

5.4.4 ONTWERPBESLISSINGEN EYE DISPLAY LLD

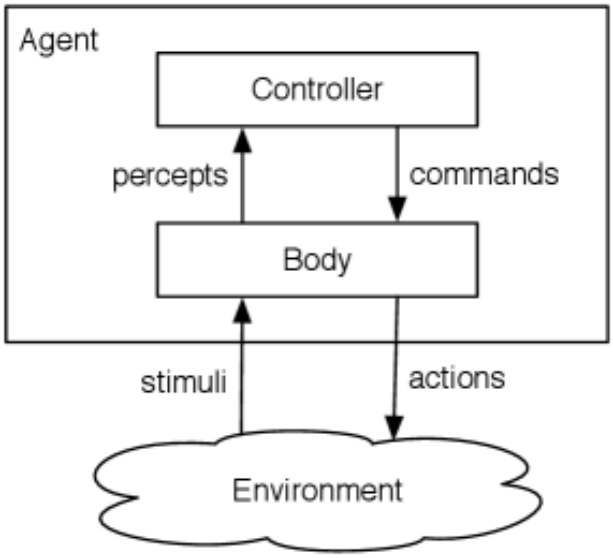
<Momenteel geen>

6 ACHTERGROND & BASISCONCEPTEN

In dit hoofdstuk worden achtergrondinformatie en basisconcepten toegelicht die nodig zijn om de rest van dit document te begrijpen. De uitleg is beknopt en richt zich op de essentie, zonder volledig de diepte in te gaan. Het doel is om voldoende context te bieden, zodat de gebruikte modellen en technologieën duidelijk zijn voor de lezer.

6.1 ALGEMEEN ROBOTMODEL

De robot voor dit project is volgens het robotmodel ontwikkeld. Met het robotmodel hebben we een duidelijke onderscheid met wat de robot kan (Body) en gaat doen (Controller).



Figuur 6.1: Een agent en zijn componenten (Poole & Mackworth, 2017).

Het robotmodel beschrijft een gesloten lus tussen de omgeving en de robot, waarbij de robot voortdurend zijn acties aanpast op basis van de veranderende waarnemingen uit de omgeving.

Hieronder worden de onderdelen van het robotmodel beschreven.

Environment (Omgeving)

Dit is de buitenwereld waar de robot zich in bevindt. De omgeving geeft stimuli aan de robot, zoals visuele signalen, geluid, temperatuur, of andere meetbare gegevens.

Body (Lichaam)

Dit is het fysieke deel van de robot dat in contact staat met de omgeving. Het ontvangt stimuli vanuit de omgeving en voert acties uit. Dit kunnen sensoren en actuatoren zijn die de fysieke interacties mogelijk maken, zoals camera's, motoren, en grijpers.

Percepts (Waarnemingen)

Dit zijn de informatieverwerkingen van de stimuli die de robot ontvangt via zijn sensoren. De robot "ziet", "hoort" of "voelt" deze waarnemingen en stuurt ze door naar de controller.

Controller (Regelaar)

Dit is het brein van de robot. Het ontvangt de waarnemingen (percepts), verwerkt deze informatie, en bepaalt welke acties de robot moet uitvoeren. De controller kan geprogrammeerde regels, algoritmen,

of machine learning modellen bevatten.

Commands (Opdrachten)

Dit zijn de instructies die de controller aan het lichaam geeft op basis van de verwerkte waarnemingen. Deze opdrachten bepalen welke acties de robot moet uitvoeren in de omgeving.

Actions (Acties)

Dit zijn de daadwerkelijke gedragingen of bewegingen van de robot in de omgeving, gebaseerd op de commando's van de controller. De acties beïnvloeden vervolgens de omgeving.

6.2 ELECTRON

Electron is een framework waarmee met webtechonologie (javascript,typescript,html,css) een native applicatie gebouwd kan worden. Tijdens dit project is dit gebruikt om de "low level eye display driver" te ontwikkelen. Met behulp van Electron kan makkelijk met webtechnologie een user interface gemaakt worden en heb je daarnaast toegang tot native dekstop functionaliteit. In ons geval het opvragen hoeveel HDMI schermen zijn aangesloten. Hiermee kunnen we bepalen waar elke oog getekent moet worden.

7 RESOURCES

BIBLIOGRAFIE

[Object Management Group (OMG), 2015] Object Management Group (OMG) (2015). OMG Unified Modeling Language, Version 2.5. OMG Document Number formal/2015-03-01 (http://www.omg.org/spec/UML/2.5).

[Poole and Mackworth, 2017] Poole, D. and Mackworth, A. (2017). Artificial intelligence: Foundations of computational agents - section 2.1. Accessed: 2025-02-13.