



5G RCID

5G Remote Controlled Indoor Drone

Sem van der Hoeven | Avans Hogeschool Breda



5G RCID

5G Remote Controlled Indoor Drone

Opdrachtnemer

Naam: Sem van der Hoeven
Studentnummer: 2162609
Organisatie: Avans Hogeschool Breda
Opleiding: Technische Informatica
E-mail: s.vanderhoeven1@student.avans.nl
Begeleider: A.H. van Dongen / H.A.J. van der Linden

Opdrachtgever

Naam: Edwin Dijkstra
Organisatie: Ericsson Telecommunicatie B.V.
E-mail: edwin.dijkstra@ericsson.com

16 juni 2023, Bergen op Zoom

Voorwoord

Dit verslag is een verantwoording van de totstandkoming van het meesterstuk van Sem van der Hoeven, een HBO student Technische Informatica. Het beschrijft een opdracht waarbij een drone is ontwikkeld die door 5G op afstand aangestuurd kan worden vanaf een computer. De opdracht is uitgevoerd in de 5G Hub onder opdracht van Ericsson.

Medewerkers van de 5G Hub en Ericsson kunnen dit verslag gebruiken om de werking van de drone te achterhalen en deze uit te breiden. Hiernaast is het ook bedoeld voor geïnteresseerden in drones, 5G of algemene geïnteresseerden in technische informatica. Enige voorkennis van programmeren of computers is vereist.

Ik wil allereerst mijn stagebegeleider, Edwin Dijkstra, bedanken voor de ondersteuning tijdens het project, de waardevolle adviezen, de kennis over 5G en de gezelligheid. Verder wil ik mijn docentbegeleiders Andries van Dongen en Hans van der Linden bedanken voor de adviezen en de begeleiding. Hiernaast wil ik Leticia Batista, Bas van Straalen, Richard Prins, Kaspar van Hulst, Robert Provoost, Ed van Hout en Iris Hertsenberg bedanken voor de gezelligheid in de 5G Hub.

Samenvatting

Ericsson heeft sinds 1920 de technologie en innovatie in Nederland aangedreven. Hier hoort ook de 5G Hub in Eindhoven bij. Dit is een innovatiecentrum voor 5G. Hier hangen verschillende 5G antennes. Ericsson wil deze graag op afstand kunnen inspecteren en hiermee willen ze ook gelijk een nieuwe demo om de mogelijkheden van 5G te laten zien. Dit kan helpen om meer klanten naar de 5G Hub te trekken.

Het doel van dit verslag is antwoord geven op de vraag *“Hoe kan een drone ontwikkeld worden die met 5G aangestuurd kan worden om zo op afstand autonoom inspecties uit te voeren?”*. Het antwoord op deze vraag wordt gegeven aan de hand van een analyse en onderzoek, waarna een ontwerp en implementatie gegeven worden om de drone te realiseren, samen met testen om aan te tonen dat het gerealiseerde product werkt. Ook wordt een grafische gebruikersinterface op de edge computer gerealiseerd zodat de drone op afstand bestuurd kan worden. Een edge computer staat dicht bij het 5G netwerk en communiceert via 5G met de drone. Voorafgaand aan het project is al een lijst met onderdelen besteld en daar wordt ook van uitgegaan.

Op de drone zit een Pixhawk 4 Mini flight controller die de motoren aanstuurt en zorgt dat de drone stabiel vliegt. Om autonoom te kunnen vliegen gebruikt deze de PX4 autopilot software. De versie hiervoor is een aangepaste beta-versie om met de flight computer te kunnen communiceren.

De flight computer is een Raspberry Pi 4B+ met 8 GB RAM. De taak van de flight computer is om alle sensoren op de drone uit te lezen, data over de besturing door te sturen naar de flight controller en de verbinding met de edge computer te onderhouden. Hierop draait Ubuntu 20 Server als besturingssysteem en ROS 2, een besturingssysteem voor robots, om met de drone te communiceren.

De flight controller heeft een schatting van zijn positie nodig om binnen op basis van snelheid bestuurd te kunnen worden. Dit is niet gelukt met de bestelde onderdelen. Als vervanging is geprobeerd dit te doen met een optical flow sensor. Dit is een sensor die met een naar beneden gerichte camera en een afstandssensor de positie van de drone meet. Dit is niet gelukt omdat de beta-versie van PX4 hier nog geen ondersteuning voor biedt. Uiteindelijk is gekozen om de drone buiten met GPS te laten vliegen om toch een bruikbaar product te realiseren. Hierdoor kan deze wel gebruikt worden om de 5G antennes die buiten hangen te inspecteren.

Om de drone te laten vliegen zijn verschillende ROS 2 modes (programma's) geschreven. De originele motoren en propellers zijn vervangen omdat deze niet voldoende waren om de drone te laten vliegen. De camera, flight computer en 5G modem op de drone zitten niet ideaal gemonteerd en het wordt aanbevolen om hier een bevestiging te ontwerpen en te 3D printen. Dit wordt ook aangeraden voor beschermkappen voor de propellers.

De flight computer heeft weinig resources over naast het aansturen van alle nodes. Hierdoor wordt aanbevolen onderzoek te doen naar een flight computer met een sterkere processor. Ook kan de camera niet met meer dan 10 FPS filmen, dus het wordt ook aanbevolen om hier een alternatief voor te vinden.

Inhoud

Samenvatting.....	4
Inhoud	5
1 Inleiding	7
2 Analyse.....	8
2.1 Context	8
2.2 Aanpak.....	9
2.2.1 Programma van eisen	9
2.3 Deelvragen.....	10
2.4 Methoden en technieken	10
2.4.1 Onderzoeksmethoden	11
2.4.2 Technieken	11
2.5 Conclusie	12
3 Flight controller.....	13
3.1 Autonoom vliegen	13
3.2 Autopilot software.....	15
3.2.1 ArduPilot.....	15
3.2.2 PX4.....	15
3.3 Communicatie met flight computer	16
3.3.1 Communicatie met ROS.....	16
3.4 Conclusie flight controller.....	17
4 Flight computer.....	18
4.1 Besturingssysteem.....	18
4.1.1 Raspberry Pi OS	18
4.1.2 Ubuntu.....	18
4.2 Uitlezen sensoren	18
4.2.1 ROS	19
4.2.2 ROS 2	19
4.3 Conclusie flight computer.....	20
5 Aansturing drone	21
5.1 Ontwerp.....	21
5.2 Configuratie	26
5.2.1 Configuratie flight computer	26
5.2.2 Configuratie flight controller	27
5.2.3 Verbinding tussen flight computer en flight controller	32
5.3 Implementatie	32
5.3.1 Aansturing flight controller.....	32
5.3.2 Uitlezen sensoren	42
5.4 Aansturing via 5G	45
5.5 Testen	46
5.5.1 Unit tests	47
5.5.2 Integratietests.....	50
5.6 Conclusie aansturing drone	54
6 API.....	56

6.1	Communicatieprotocol.....	56
6.2	Ontwerp.....	57
6.3	Implementatie	59
6.4	Testen	60
6.5	Conclusie API	66
7	Conclusies en aanbevelingen.....	68
7.1	Conclusies.....	68
7.2	Aanbevelingen	68
8	Bibliografie.....	70
	Bijlage A: ROS 2 nodes.....	74
	A.1: PositionChanger node.....	74
	A.2: PX4Controller node.....	81
	A.3: Heartbeat node.....	88
	A.4: Hoogtemeter node.....	90
	A.5: Multiflex node.....	92
	A.6: LIDAR node.....	94
	A.7: CameraController node.....	96
	A.8: TrackerPositioning node.....	98
	A.9: RelaisController node.....	100
	A.10: DroneStatus node	101
	A.11: FailSafe node	102
	A.12: ApiListener node	103
	A.13: GUI op edge computer.....	108
	Bijlage B: Code en output tests	111
	B.1: Camera unittest.....	111
	B.2: Failsafe unittest	113
	B.3: Heartbeat control mode unittest	115
	B.4: Integratietest failsafe en PX4Controller	117
	B.5: Integratietest PositionChanger, LIDAR en Failsafe.....	120
	B.6: Integratietest PositionChanger en objectdetectie	123
	B.7: ApiListener integratietest.....	126
	B.8: Output failsafe aanzetten door noodstop op GUI.....	128
	B.9: Output failsafe door te lage batterij.....	129
	B.10: Output wegvallen verbinding met API	129
	Bijlage C: Overzicht video's.....	130
	Register.....	131

1 Inleiding

Ericsson is al sinds 1920 actief in Nederland. Het heeft gedurende die tijd de technologie en innovatie aangedreven. Eén van die innovaties is de oprichting van de 5G Hub in Eindhoven, samen met VodafoneZiggo en Brainport Eindhoven. De 5G Hub is een innovatiecentrum voor 5G. 5G staat voor de vijfde generatie mobiele telecommunicatienetwerken. Het wordt gebruikt voor draadloos verzenden en ontvangen van data via radiogolven. In de 5G Hub staan meerdere demo's om het potentieel van 5G te laten zien. Om deze te laten werken, hangen er verschillende 5G-antennes in de ruimte.

Ericsson wil deze antennes graag inspecteren, zonder dat ze verplaatst moeten worden of dat iemand de antennes van dichtbij moet bekijken. Dit wil Ericsson doen met een 5G aangestuurde drone, waarmee ze ook gelijk een nieuwe demo kunnen realiseren om het potentieel van 5G laten zien. Hiermee kan Ericsson ervoor zorgen dat de 5G antennes optimaal blijven functioneren en tegelijk laten zien hoe 5G gebruikt kan worden om een drone aan te sturen.

Het doel van dit verslag is antwoord geven op de vraag *“Hoe kan een drone ontwikkeld worden die met 5G aangestuurd kan worden om zo op afstand autonoom inspecties uit te voeren?”*. Het antwoord op deze hoofdvraag wordt gegeven aan de hand van een analyse, onderzoek, ontwerp en realisatie van de drone en een API om de drone vanaf een andere computer aan te sturen. Voorafgaand aan het project is een lijst met onderdelen besteld. Hier wordt tijdens het project van uitgegaan. Er wordt niet actief gezocht naar andere onderdelen, tenzij dit nodig is om een probleem op te lossen.

Om de hoofdvraag van dit verslag te beantwoorden, beschrijft Hoofdstuk 2 de analyse van het project en de context van het probleem. Hoofdstuk 3 beschrijft de flight controller, hoe deze onderzocht is en wat de mogelijkheden zijn om deze aan te sturen. Hoofdstuk 4 beschrijft deze aspecten van de flight computer. Hierna wordt in Hoofdstuk 5 het ontwerp en de implementatie van de drone beschreven. In Hoofdstuk 6 wordt beschreven hoe de API ontwikkeld is en welke stappen hiervoor genomen zijn. In Hoofdstuk 7 worden de conclusies uit de daarvoor genoemde hoofdstukken beschreven en wordt een antwoord gegeven op de hoofdvraag. In dit hoofdstuk worden ook aanbevelingen gegeven over het vervolg van het project.

2 Analyse

Dit hoofdstuk beschrijft de analysefase van het project. De context van de opdracht en het probleem worden besproken in Paragraaf 2.1. Paragraaf 2.2 beschrijft de aanpak van het project. Paragraaf 2.3 beschrijft de deelvragen die opgesteld zijn om de hoofdvraag te beantwoorden. In Paragraaf 2.4 worden de gebruikte methoden en technieken behandeld. Als laatste wordt in Paragraaf 2.5 een conclusie gegeven.

2.1 Context

De opdracht wordt uitgevoerd voor Ericsson Nederland B.V. Ericsson is één van de leidende leveranciers van informatie- en communicatietechnologie (ICT) voor serviceproviders (Ericsson, 2023a). Het bedrijf focust zich vooral op connectiviteit en heeft een visie waarin de hele wereld met elkaar verbonden is (Ericsson, 2023b). Samen met Brainport Development, High Tech Campus en VodafoneZiggo heeft het bedrijf de 5G Hub opgericht. Dit is een innovatiecentrum voor 5G waar nieuwe technologieën worden onderzocht en getest. Dit project wordt dan ook voor de 5G Hub uitgevoerd.

In de 5G Hub hangen een aantal 5G antennes. Eén van deze antennes is de 5570 antenne van Ericsson. Hiervan zijn er drie te zien in Figuur 2.1. Dit is een antenne die normaal buiten hangt, maar voor demo doeleinden in de 5G Hub hangt. Ericsson wil deze visueel inspecteren om te controleren of deze goed aangesloten is en geen defecten bevat. De kabels, ook wel *feeders* genoemd, kunnen bij het installeren van de antenne verkeerd aangesloten worden of aan de verkeerde antenne aangesloten worden. Dit probleem wordt *swapped feeders* genoemd. Ook kan het voorkomen dat de connector van deze kabel niet goed is aangedraaid. Verder wil Ericsson graag controleren op oxidatie op de antenne of connectoren en of de kabels beschadigingen bevatten. Deze inspecties worden niet met een vast interval uitgevoerd. Ericsson houdt continu de kwaliteit van het netwerk in de gaten. Als er opgemerkt wordt dat de kwaliteit van het netwerk bij een bepaalde antenne verslechtert, wordt een inspectie uitgevoerd.



Figuur 2.1: 5570 5G antennes (Ericsson, 2023c)

Hiernaast wil Ericsson graag een nieuwe demo voor in de 5G Hub. Momenteel zijn er een aantal demo's te zien die verschillende aspecten van 5G demonstreren. Een voorbeeld dat goed aansluit op dit project is de 5G aangestuurde auto (5G Hub, 2023). Om de functionaliteit van 5G meer te demonstreren, wil Ericsson met de drone aantonen dat 5G geschikt is om een drone aan te sturen, waardoor uiteindelijk Beyond Visual Line of Sight (vliegen zonder dat de drone zichtbaar is voor de bestuurder) mogelijk wordt. Hiermee kan ook aangetoond worden dat 5G genoeg capaciteit bevat om video met hoge resolutie vanaf de drone naar een grondstation te streamen waarmee de drone bestuurd wordt. Hierbij zijn de eigenschappen van 5G belangrijk omdat het camerabeeld beschikbaar moet zijn als de drone buiten het zicht van de bestuurder vliegt en bijvoorbeeld een noodlanding moet maken.

Voorafgaand aan het project heeft Ericsson samen met Dutch Drone Center een lijst met onderdelen opgesteld en besteld om de volledige drone te bouwen. Deze zijn al aanwezig en zijn daarom het uitgangspunt voor verder werk aan de drone in dit project. Deze onderdelen zijn voldoende om een

drone met 6 motoren te bouwen. Er is gekozen voor een frame met 6 motoren omdat dit minder geluid genereert dan met 4 motoren. Naast het frame en de motoren zijn ook de batterij, flight computer (Raspberry Pi 4B+ 2GB), flight controller (Pixhawk 4 Mini), ESC's, propellers, hoogtemeter, objectdetectiesensoren waaronder een LIDAR, 5G modem en een hoge resolutie camera besteld.

2.2 Aanpak

Het project wordt stapsgewijs uitgevoerd volgens de watervalmethode. Dit betekent dat er eerst een globale analyse en een globaal onderzoek gedaan wordt om de eisen, probleemstelling en gewenste resultaten van het project duidelijk te maken. Tijdens dit onderzoek wordt ook onderzocht welke onderdelen gemaakt moeten worden om informatie te verzamelen over hoe deze het beste ontworpen en geïmplementeerd kunnen worden. Hierna wordt elk onderdeel apart ontworpen, geïmplementeerd en getest. Deze aparte onderdelen vormen dan samen het resultaat van het project en het antwoord op de hoofdvraag. Op deze manier wordt het systeem bottom-up opgebouwd.

2.2.1 Programma van eisen

Aan de hand hiervan is een MoSCoW analyse gemaakt om de eisen in kaart te brengen (Hoeven, Plan van Aanpak, 2023b). Deze zijn opgedeeld in must have, should have, could have en won't have eisen. Deze worden hieronder uitgelegd.

Must have eisen

De must have eisen zijn de minimale eisen waaraan voldaan moet worden om het project te laten slagen. Deze zijn als volgt:

- MH1:* De drone moet door een PixHawk 4 Mini flight controller bestuurd worden. Deze moet de ESC's (Electronic Speed Controllers) aansturen om de motoren te laten draaien.
- MH2:* De drone moet 6 motoren bevatten om de drone te kunnen laten vliegen. Dit moet gebeuren met 6 motoren omdat dit stiller is dan 4 motoren.
- MH3:* De drone moet een mate van objectdetectie bevatten waarmee het objecten in de buurt van de drone kan detecteren.
- MH4:* De drone moet een corrigerende actie uitvoeren als het te dicht bij een object komt, zodat het veilig kan blijven vliegen.
- MH5:* De drone moet positiebepaling hebben op basis van twee beacons op de drone zelf, en vier beacons in de ruimte waar de drone vliegt.
- MH6:* De drone moet een camera bevatten waarmee de 5G-antennes op afstand (1 meter) geïnspecteerd kunnen worden door foto's te maken.
- MH7:* De drone moet op afstand via 5G bestuurd kunnen worden door een externe computer.
- MH8:* De drone moet met 5G aangestuurd kunnen worden binnen het netwerk van de 5G Hub.
- MH9:* De drone moet in een ruimte van maximaal 50x40x7 meter (de afmetingen van de ruimte van de 5G Hub) kunnen vliegen.

Should have eisen

De should have eisen zijn eisen die belangrijk zijn voor het project, maar niet noodzakelijk zijn om het project te laten slagen. Deze zijn als volgt:

- SH1:* De drone moet positiebepaling bevatten en stabiel (zonder drift) kunnen vliegen op 1 cm nauwkeurigheid.
- SH2:* De drone moet data over de status ervan naar een API kunnen sturen zodat deze vanaf een ander apparaat bekeken kan worden.
- SH3:* De drone moet zelf een vooraf ingestelde route met een vastgesteld start- en stoppunt kunnen vliegen.
- SH4:* De drone moet op vooraf ingestelde posities kunnen stoppen en een foto kunnen maken.
- SH5:* De drone moet via een API handmatig bestuurd kunnen worden. Deze moet handmatig kunnen opstijgen, landen, rondvliegen en een foto kunnen maken.
- SH6:* De gemaakte foto's moeten verstuurd kunnen worden naar een API zodat deze vanaf een ander apparaat bekeken kunnen worden.

Could have eisen

De could have eisen zijn eisen die niet noodzakelijk zijn voor het project en als extra gezien kunnen worden, voor als er nog tijd over is aan het einde van het project. Deze zijn als volgt:

- CH1:* De drone moet een scherm bevatten waar gegevens over de drone (positie, snelheid van motoren) op weergegeven kunnen worden.
- CH2:* De gemaakte foto's moeten kunnen worden behandeld door een AI/ML-algoritme om verschillen te kunnen detecteren.

Won't have eisen

De won't have eisen zijn eisen waar de drone niet aan zal voldoen. Deze zijn als volgt:

- WH1:* De drone zal niet bediend kunnen worden door een apparaat wat niet met de 5G Hub verbonden is.

2.3 Deelvragen

Aan de hand van de hoofdvraag gedefinieerd in Hoofdstuk 1 zijn een aantal deelvragen opgesteld. Deze zijn hieronder opgesomd:

- D1 Hoe kunnen de Raspberry Pi flight computer en PixHawk 4 Mini flight controller gebruikt worden om de drone te laten vliegen?
- D2 Waar zit de intelligentie van de drone die de onderdelen samen laat werken?
- D3 Hoe kan de drone op afstand via 5G bestuurd worden?
- D4 Welke 5G-technologieën zijn nodig om de drone succesvol inspecties uit te laten voeren?

De antwoorden op deze deelvragen vormen het antwoord op de hoofdvraag. Deelvraag D1 wordt in Hoofdstuk 3 en Hoofdstuk 4 beantwoord. Deelvraag D2 wordt beantwoord met de informatie uit Hoofdstuk 5. Deelvraag D3 en D4 worden in Hoofdstuk 5 en 6 beantwoord. De volgende paragraaf beschrijft welke onderzoeksmethoden gebruikt worden om deze deelvragen te beantwoorden.

2.4 Methoden en technieken

Om de drone en de API te ontwikkelen en zo antwoord te kunnen geven op de hoofdvraag zijn een aantal methoden en technieken gedefinieerd om dit te bereiken. De technieken zijn globaal gedefinieerd in de analysefase. Welke hiervan specifiek gebruikt worden, wordt uitgelegd in de

volgende hoofdstukken. Dit is op deze manier beschreven omdat dit pas duidelijker is geworden tijdens de uitvoering.

2.4.1 Onderzoeksmethoden

Om een volledig onderzoek uit te voeren en alle deelvragen te beantwoorden wordt gebruik gemaakt van een aantal onderzoeksmethoden (HBO-i, 2022). Tijdens de analysefase is een *Problem analysis* uitgevoerd om duidelijk te maken welk probleem precies met deze opdracht opgelost wordt. Hiernaast is ook een *Requirements prioritization* uitgevoerd om de eisen voor het project duidelijk te maken en te verdelen. Hieruit is de in Paragraaf 2.2 genoemde MoSCoW-analyse opgesteld. Dit heeft betrekking op alle deelvragen. Na de analysefase is ook een *Pitch* aan de opdrachtgever gegeven om een idee te geven van waar het project over gaat en om feedback over de opdracht te krijgen.

Om deelvraag D1 te beantwoorden wordt allereerst gebruik gemaakt van een *Literature study*. Hierbij wordt m.b.v. literatuur en documentatie onderzocht hoe de flight controller en flight computer werken en hoe deze gebruikt kunnen worden. Er heeft ook vaak een *Brainstorm* plaatsgevonden met de bedrijfsbegeleider om te bespreken hoe de flight controller en flight computer gebruikt kunnen worden. Verder zijn voor de flight controller en flight computer *Unit tests* uitgevoerd om te controleren of ze als aparte onderdelen op zichzelf werken. Als laatst zijn *Integration tests* uitgevoerd om te kijken of de onderdelen in combinatie met elkaar werken.

Om antwoord te geven op deelvraag D2 wordt ook gebruik gemaakt van een *Literature study* waarbij de literatuur en documentatie worden gebruikt om inzicht te krijgen in hoe de onderdelen werken en hoe de flight computer deze samen kan laten werken. Per gemaakt onderdeel wordt ook een *Unit test* uitgevoerd om te controleren of dat onderdeel op zichzelf goed werkt. Als alle onderdelen op zichzelf werken wordt een *System test* uitgevoerd om te kijken of de flight computer op basis van de sensoren de drone correct aan kan sturen.

Om een antwoord te geven op deelvraag D3 wordt allereerst ook van een *Literature study* gebruik gemaakt. Hierbij wordt m.b.v. de literatuur onderzocht wat er allemaal nodig is om de drone via 5G te besturen. Er worden ook *Unit tests* uitgevoerd om te kijken of de API en de communicatie naar de drone apart werken. Hierna wordt een *System test* uitgevoerd om te controleren of het aansturen via 5G als geheel werkt. Verder wordt ook een *Ethical check* uitgevoerd om te zorgen dat de drone geen gevaarlijke situaties kan veroorzaken, door het inbouwen van een failsafe voor het wegvallen van communicatie.

Om als laatst een antwoord te geven op deelvraag D4 wordt een *Literature study* gebruikt om in de literatuur informatie te vinden over verschillende 5G technieken en hoe deze gebruikt kunnen worden om de drone inspecties uit te laten voeren. Ook wordt een *Expert interview* gehouden met de bedrijfsbegeleider om de voordelen van 5G in het opzicht van de drone in kaart te brengen.

2.4.2 Technieken

Om de drone te laten vliegen zijn een flight computer en een flight controller nodig. De flight controller zorgt ervoor dat de drone stabiel kan vliegen en stuurt alle motoren aan. Bij een gewone, handgestuurde drone wordt deze bestuurd door een afstandsbediening. In dit geval is de

afstandsbediening als het ware vervangen door de flight computer, te zien in Figuur 2.2. Deze stuurt commando's naar de flight controller over waar de drone naartoe moet vliegen.

De drone wordt aangestuurd door een PixHawk 4 Mini flight controller, te zien in Figuur 2.3. Deze maakt gebruik van *autopilot* software. Dit is een softwarepakket waarmee de drone aangestuurd kan worden via de flight controller. Welke softwarepakketten hiervoor beschikbaar zijn wordt in Hoofdstuk 3 beschreven. In Hoofdstuk 5 wordt beschreven welke hiervan gebruikt gaat worden en waarom.

Voor de flight computer wordt gebruik gemaakt van een besturingssysteem. Hier draait een versie van ROS (Robot Operating System) op waarmee de metingen van de sensoren samengevoegd kunnen worden. Voor ROS is gekozen omdat dit de beste optie is om verschillende sensoren en onderdelen aan elkaar te koppelen en zo een robot aan te sturen. De code om de drone aan te sturen en om de sensoren uit te lezen wordt geschreven in Python of C++ omdat dit de meest gebruikte talen voor ROS zijn. Welk besturingssysteem, welke versie van ROS en welke programmeertaal gebruikt gaat worden en waarom deze keuzes zijn gemaakt wordt besproken in Hoofdstuk 5. De hele drone met alle onderdelen is te zien in Figuur 5.2 in Hoofdstuk 5.

Gedurende het hele project is documentatie geschreven over de uitgevoerde taken en is dit verslag bijgewerkt. Dit is gedaan met Microsoft Word in combinatie met OneDrive. Hiervoor is gekozen omdat zo alle documentatie actueel blijft. Ook is alle documentatie zo op één plek beschikbaar.

Voor het bijhouden van de code wordt gebruik gemaakt van versiebeheer in Git. Alle ontwerpen worden gemaakt m.b.v. UML diagrammen. Het schrijven van de code wordt gedaan in Visual Studio Code. Deze keuze is gemaakt omdat deze ondersteuning biedt voor verschillende programmeertalen en een integratie biedt voor ROS. Welke andere tools gebruikt zijn voor het configureren, instellen en programmeren van de drone wordt in Hoofdstuk 5 en Hoofdstuk 6 uitgelegd.

2.5 Conclusie

De opdracht wordt uitgevoerd voor Ericsson Nederland B.V. voor de 5G Hub. Hier hangen een aantal 5G-antennes die Ericsson graag vanaf de grond wil inspecteren. Hiernaast wil Ericsson graag een nieuwe demo voor een 5G aangestuurde drone om aan klanten te laten zien in de 5G Hub.

Voorafgaand aan het project zijn een aantal onderdelen besteld om een volledige drone te bouwen. Van deze onderdelen wordt tijdens het project uitgegaan.

Het project wordt uitgevoerd volgens de watervalmethode. Om een volledig antwoord te geven op de hoofdvraag worden er een aantal onderzoeksmethoden gebruikt. Om de drone te laten vliegen zijn een flight controller en flight computer vereist. De flight controller voor dit project is de Pixhawk 4 Mini. Hoe deze onderzocht is en hoe deze kan communiceren met de flight computer wordt in het volgende hoofdstuk, Hoofdstuk 3, uitgelegd.



Figuur 2.2: Flight computer (Raspberry Pi, 2023)



Figuur 2.3: Flight controller (Pixhawk, 2023f)

3 Flight controller

Dit hoofdstuk beschrijft het onderzoek naar de Pixhawk 4 Mini flight controller. Het proces van onderzoek wordt beschreven samen met de verschillende opties om de flight controller aan te sturen. Ook wordt beschreven op welke manieren de flight controller met de flight computer verbonden kan worden. Paragraaf 3.1 legt uit hoe een drone autonoom vliegt. Paragraaf 3.2 legt uit welke software hiervoor gebruikt kan worden. In Paragraaf 3.3 wordt uitgelegd hoe de flight controller met de flight computer verbonden kan worden.

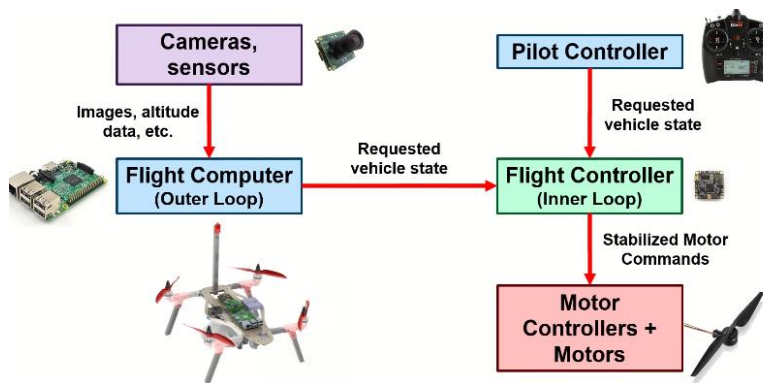
Het doel van een flight controller is om een drone stabiel te laten vliegen. Als flight controller voor de drone is de Pixhawk 4 Mini gekozen (Pixhawk, 2023f). Dit is een flight controller module die de ESC's (Electronic Speed Controllers) aanstuurt om de motoren met een bepaalde snelheid te laten draaien en zo de drone kan laten vliegen. Pixhawk is een set aan open source hardware specificaties en richtlijnen voor het ontwikkelen van drones. De flight controller is hierop gebaseerd, specifiek op de FMUv5 standaard (Pixhawk, 2022a). Deze heeft acht PWM (Pulse Width Modulation) uitgangen om in totaal 8 motoren aan te sturen. Naast een drukmeter en een EEPROM (een stukje geheugen) om instellingen op te slaan, zijn de belangrijkste onderdelen van de flight controller als volgt (Hoeven, Intelligentie voor een 5G aangestuurde drone, 2023a):

- FMU (Flight Management Unit). Dit is de microprocessor van de autopilot.
- Twee IMU's (Inertial Measurement Units). Dit zijn sensoren die de snelheid en positie van de drone kunnen meten.
- Een magnetometer om het magnetisch veld in drie assen rondom de drone te meten.
- Een ingebouwde GPS (Global Positioning System) om een schatting van de positie van de drone te maken.

3.1 Autonoom vliegen

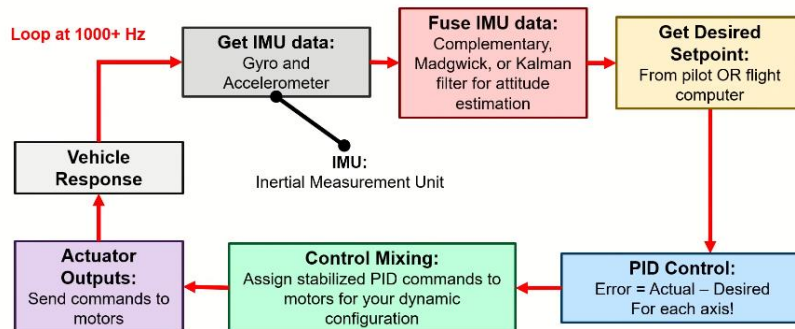
Een voertuig dat vliegt met behulp van propellers op de uiteinden van het voertuig heet een multicopter. Een multicopter wordt pas een drone als het autonoom kan vliegen (ArduPilot, 2022). Om de drone autonoom te laten vliegen is autopilot software nodig. Dit houdt in dat de informatie van de versnellingsmeter en de gyroscoop gecombineerd wordt met de data van de GPS en de barometer om te zorgen dat de flight controller niet alleen de oriëntatie, maar ook de positie van de drone begrijpt (Defining Drones, 2023).

Een drone heeft twee niveaus van besturing: de *inner loop* en de *outer loop*. Bij de inner loop wordt het voertuig gestabiliseerd op een gewenste hoek en snelheid. Deze wordt uitgevoerd op de flight controller. De outer loop wordt uitgevoerd op de flight computer en hierbij worden hoek- of snelheidscommando's gegenereerd om de drone van punt A naar punt B te krijgen. De outer loop en flight computer worden in Hoofdstuk 4 besproken. Hoe de informatiestroom tussen de inner en outer loop gaat is te zien in Figuur 3.2.



Figuur 3.2: Informatiestroom tussen de inner loop en outer loop (Rehm, 2021)

Bij de inner loop wordt de data uit de IMU verkregen en samengevoegd om de houding van de drone te bepalen. Hierna wordt de gewenste positie verkregen van de flight computer en wordt het verschil tussen de huidige positie en de gewenste positie berekend m.b.v. een PID-regelaar. Hierna wordt de drone naar de gewenste positie bewogen en begint het meetproces opnieuw. Dit proces is te zien in Figuur 3.1.



Figuur 3.1: Overzicht stappen inner loop (Rehm, 2021)

Het is moeilijk om de oriëntatie van de drone ten opzichte van een vast referentiekader, zoals de grond onder de drone, definitief te meten. Dit komt omdat de sensoren die gebruikt worden voor het meten van de houding van de drone een bepaalde mate van onnauwkeurigheid (ruis) bevatten (Hoeven, Intelligentie voor een 5G aangestuurde drone, 2023a). Deze schatting kan echter precies gemaakt worden door gebruik te maken van complementaire filters (Narkhede, Poddar, Walambe, Ghinea, & Kotecha, 2021). Hierbij worden de gegevens van meerdere sensoren gebruikt, omdat de data uit één sensor niet genoeg is om een goede precieze schatting te maken (White, 1991). Hiervoor wordt de data van de gyroscop, versnellingsmeter en magnetometer gebruikt.

Het samenvoegen van de data uit deze sensoren wordt voornamelijk gedaan met een Extended Kalman Filter (EKF) (Munguia & Grau, 2014). Dit is een uitbreiding op de Kalman Filter (KF). Een KF is een algoritme dat bestaat uit een reeks matrixvergelijkingen waarmee in korte tijd de echte waarde van de oriëntatie geschat kan worden wanneer de meetgegevens van de sensoren veel onnauwkeurigheden (ruis) bevatten. Een EKF is een niet-lineaire versie van de KF die iteratief toegepast kan worden op nieuwe sensordata en beter te gebruiken is bij data over meerdere assen dan andere soortgelijke algoritmen (Zanj, 2018).

3.2 Autopilot software

De flight controller ondersteunt twee verschillende autopilot software pakketten: PX4 en ArduPilot (Pixhawk, 2023f). Beide zijn open source en bieden ondersteuning voor meerdere soorten voertuigen, waaronder een multicopter met 6 propellers (Hoeven, Intelligentie voor een 5G aangestuurde drone, 2023a). Beide softwarepakketten worden in de volgende twee paragrafen apart besproken.

3.2.1 ArduPilot

ArduPilot is een open source autopilot systeem dat ondersteuning biedt voor meerdere soorten voertuigen, waaronder multicopters. Met ArduPilot kan de flight controller aangestuurd worden via een flight computer die de drone aanstuurt via een API. Als er geen flight computer gebruikt wordt, kan de drone ook aangestuurd worden via een Ground Control Station. Dit is een programma dat op een gewone computer draait waarmee de drone ingesteld en bestuurd kan worden. De flight computer of API communiceert met ArduPilot via MAVLink. Dit is een lichtgewicht communicatieprotocol om te communiceren met drones (MAVLink, 2022). De ArduPilot code draait op een OS (Operating System) die op de hardware van de flight controller draait. Eventuele andere sensoren kunnen ook op ArduPilot aangesloten worden via de flight controller.

3.2.2 PX4

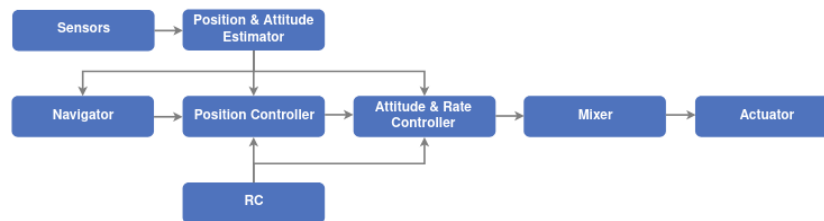
PX4 is een open source autopilot systeem dat (net zoals ArduPilot) ondersteuning biedt voor meerdere soorten voertuigen, waaronder multicopters. De open source hardware waar PX4 voor is gemaakt zijn de Pixhawk flight controllers. Hier is de Pixhawk 4 mini er één van.

Bij het gebruik van een flight computer communiceert deze met de flight controller via MAVLink. De communicatie met de Ground Control Station kan verzorgd worden door de flight computer of direct via de flight controller via een USB kabel. Verder is het bij het gebruik van PX4 met een flight computer gebruikelijk dat de flight computer andere sensoren uitleest en op basis hiervan data naar PX4 stuurt. PX4 draait ook op een OS dat op de flight controller draait. Dit OS bevat middleware om met de flight computer te communiceren en andere taken uit te voeren. Verder bevat het OS drivers om verschillende sensoren (zoals een kompas) direct op de flight controller uit te kunnen lezen (Willee & Sriram, PX4 System Architecture, 2022).

De PX4 software bestaat uit twee lagen. Dit zijn de flight stack: een flight control- en schattingssysteem en de middleware: een robotica laag die ondersteuning biedt voor meerdere soorten autonome robots. De middleware laag zorgt voor interne en externe communicatie met de flight controller en zorgt voor hardwareintegratie. Bij PX4 gaat de communicatie tussen de onderdelen van de flight controller via uORB berichten. Dit is een asynchrone berichten API die werkt met publish/subscribe berichten en communicatie tussen processen en threads mogelijk maakt (Willee, Oes, & Otu, uORB Messaging, 2022). Het communiceren met een flight computer wordt via het MAVLink protocol gedaan. Dit wordt verder uitgelegd in Hoofdstuk 4.

De flight stack is een verzameling van algoritmes voor het aansturen en navigeren van autonome voertuigen. In de flight stack zit een state machine die regelt in welke modus de flight controller zit. Bij het autonoom vliegen wordt eerst gekeken naar welke positie de drone moet vliegen. Hierna wordt berekend wat de houding en oriëntatie van de drone moet zijn en hoe de motoren aangestuurd moeten worden om dit te bereiken. Deze berekening wordt in PX4 gedaan met een EKF

algoritme genaamd EKF2 (Pixhawk, 2023g). Als laatst worden de motoren aangestuurd. Tijdens dit proces wordt de data van alle sensoren meegenomen. De opbouw van de flight stack is te zien in Figuur 3.3 (Willee & Chiap, PX4 Architectural Overview, 2022).



Figuur 3.3: Opbouw van PX4 flight stack

3.3 Communicatie met flight computer

Om de flight controller fysiek aan de flight computer te verbinden, moet gebruik gemaakt worden van de UART GPIO pinnen op de Raspberry Pi. Deze kunnen verbonden worden met de *TELEM1* (USART2) of *UART/I2C B* (UART4) poort op de flight controller (Willee, Fjare, & Poon, Pixhawk 4 Mini: Serial Port mapping, 2022). Deze zijn in het rood omcirkeld in Figuur 3.4. Op de Raspberry Pi moeten de GPIO pinnen 2 of 4 (5V), 6 (GND), 8 (TX) en 10 (RX) gebruikt worden.



Figuur 3.4: beschikbare poorten om de flight computer te verbinden

De communicatie tussen de flight controller en flight computer gaat zowel bij PX4 als ArduPilot via het MAVLink protocol. Bij beide autopilot pakketten moeten een aantal parameters ingesteld worden om de seriële poort, het protocol en de snelheid van de communicatie in te stellen (Hoeven, Intelligentie voor een 5G aangestuurde drone, 2023a).

3.3.1 Communicatie met ROS

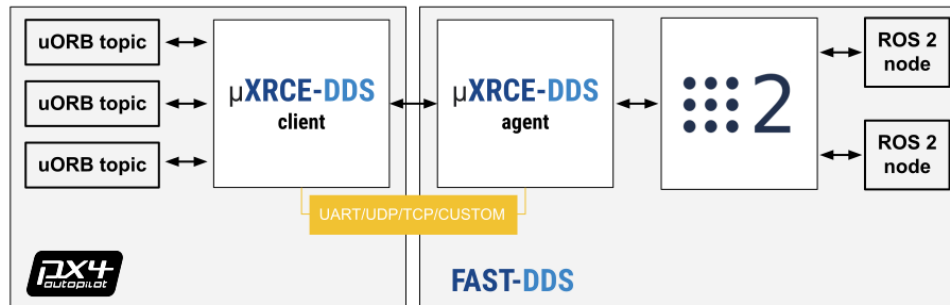
Beide autopilot pakketten zijn te bedienen via ROS (Robot Operating System) en hebben een eigen manier om hiermee te communiceren. De werking van ROS wordt verder uitgelegd in Paragraaf 4.2. PX4 ondersteunt zowel ROS 1 als ROS 2. Het kan via ROS 1 aangestuurd worden door gebruik te maken van MAVROS. Dit is een ROS package die MAVLink communicatie tussen ROS nodes mogelijk maakt. Het ontwikkelteam van PX4 raadt echter aan om dit niet te gebruiken en over te stappen naar ROS 2 (Willee, Roche, Higgason, Marques, & Hwang, 2023):

ROS 2 is the “latest and greatest” version of ROS. The PX4 development team recommend that all users upgrade to ROS 2!

De communicatie tussen ROS 2 en PX4 gaat via de PX4-ROS 2 bridge. Dit is een interface die een directe verbinding aanbiedt tussen uORB berichten van PX4 en ROS 2 DDS berichten en types. DDS is de berichtenarchitectuur die bij ROS 2 wordt gebruikt. Dit zorgt ervoor dat ROS 2 nodes direct real-time toegang hebben tot de data en functies van PX4.

De middleware die de communicatie tussen de uORB topics van PX4 en ROS 2 mogelijk maakt is XRCE-DDS. Dit is een middleware die ervoor zorgt dat de uORB berichten op een flight computer met ROS 2 kunnen worden gepubliceerd en dat de flight computer hierop kan abonneren door deze te

vertalen naar ROS 2 topics. Dit zorgt voor een snelle en betrouwbare integratie tussen PX4 en ROS 2 (Willee, XRCE-DDS (PX4-FastDDS Bridge), 2023). PX4 gebruikt een implementatie van XRCE-DDS die gebruik maakt van eProxima Micro XRCE-DDS. Dit is een XRCE-DDS implementatie voor apparaten met weinig resources (eProxima, 2018). Hoe de communicatie tussen ROS 2 en micro XRCE-DDS zoals hierboven beschreven gaat is te zien in Figuur 3.5.



Figuur 3.5: Communicatie tussen PX4 en ROS 2 met micro XRCE-DDS

Deze middleware is echter nog niet beschikbaar in de stabiele versie van PX4 (v1.13) en alleen nog maar in de v1.14 beta versie. Om de PX4-ROS 2 Bridge te gebruiken op PX4 versie v1.13, wordt de microRTPS Bridge middleware gebruikt (Pixhawk, 2023a). Deze zorgt er ook voor dat uORB berichten omgezet worden naar ROS 2 topics en andersom. Dit gaat door een Real-Time Public Subscribe (RTPS) interface aan PX4 toe te voegen, waardoor uORB berichten ook real-time omgezet kunnen worden naar ROS 2 topics en andersom (Pixhawk, 2023b).

ArduPilot kan aangestuurd worden met ROS 1 via MAVROS. Deze package kan ROS topics naar MAVLink berichten omzetten en andersom, waardoor voertuigen die ArduPilot draaien kunnen communiceren met ROS. MAVROS voor ROS 2 is nog in pre-alpha ontwikkeling en is (tijdens het schrijven van dit verslag) nog afhankelijk van de ROS 1 MAVROS packages (MAVLink, 2021).

3.4 Conclusie flight controller

Het doel van een flight controller is om een drone stabiel te laten vliegen. De flight controller voor dit project is de Pixhawk 4 Mini. Deze stuurt de ESC's aan die de motoren aansturen. Een multicopter (een voertuig dat vliegt met meerdere propellers) wordt pas een drone als het autonoom kan vliegen. Hiervoor is autopilot software nodig.

Voor het autonoom vliegen heeft een drone twee niveaus van besturing: de inner loop en de outer loop. Bij de inner loop wordt het voertuig gestabiliseerd op een gewenste hoek en snelheid. Deze wordt op de flight controller uitgevoerd. De outer loop wordt op de flight computer uitgevoerd en hierbij worden commando's gegenereerd om de drone naar een bepaald punt te bewegen.

De flight controller ondersteunt twee open source autopilot softwarepakketten: PX4 en ArduPilot. De flight controller kan met de flight computer communiceren via het MAVLink protocol. De autopilot pakketten kunnen beide door de flight computer aangestuurd worden via ROS. ArduPilot biedt alleen ondersteuning voor ROS 1 via MAVROS en PX4 biedt ondersteuning voor ROS 1 via MAVROS en ROS 2 via XRCE-DDS. Hoe de flight computer onderzocht is en hoe deze de sensoren op de drone uit kan lezen wordt beschreven in het volgende hoofdstuk, Hoofdstuk 4.

4 Flight computer

Dit hoofdstuk beschrijft het onderzoek naar de Raspberry Pi flight computer. Het proces van onderzoek wordt beschreven samen met de verschillende opties om de flight computer aan te sturen. Als eerst beschrijft Paragraaf 4.1 welke besturingssystemen beschikbaar zijn voor de flight computer. Daarna beschrijft Paragraaf 4.2 hoe deze de sensoren uit kan lezen.

De taak van de flight computer is om de flight controller aan te sturen en te zorgen dat deze alleen bezig hoeft te zijn met het laten vliegen van de drone. Voor dit project wordt de flight computer ook gebruikt omdat er nog geen kant-en-klare oplossingen bestaan om een drone via 5G aan te sturen. De functies van de flight computer zijn (Hoeven, Intelligentie voor een 5G aangestuurde drone, 2023a):

- Sensoren (LIDAR, hoogtemeter, camera, beacons) op de drone uitlezen.
- Data over de positie naar de flight controller sturen.
- Data over de besturing naar de flight controller sturen.
- Communicatie met edge computer onderhouden.

4.1 Besturingssysteem

Om deze functies uit te voeren moet een OS (Operating System) gekozen worden dat op de flight computer uitgevoerd wordt. Er zijn verschillende Besturingssystemen die geschikt zijn voor de Raspberry Pi 4B+. De meest gebruikte zijn Raspberry Pi OS en Ubuntu. Deze zijn allebei gebaseerd op Debian Linux.

4.1.1 Raspberry Pi OS

Raspberry Pi OS is een besturingssysteem dat speciaal ontwikkeld en geoptimaliseerd is om op verschillende modellen van de Raspberry Pi te werken. Dit besturingssysteem heeft een versie met een GUI (*desktop*), een versie met GUI en vooraf geïnstalleerde programma's (*Desktop full*) en een versie zonder GUI (*Lite*). Deze versie heeft het minste geheugen en de minste opslag nodig, en verbruikt ook de minste resources. Raspberry Pi OS komt met een vooraf geïnstalleerd programma om instellingen specifiek voor de Raspberry Pi te configureren, zoals de GPIO of kloksnelheid.

4.1.2 Ubuntu

Ubuntu is één van de meest gebruikte Linux distributies ter wereld (Lunsing, 2018). Het is gemaakt om op verschillende soorten apparaten te draaien, waaronder de Raspberry Pi. Ubuntu heeft ook drie versies: *Desktop*, *Server* en *Core*. *Desktop* en *Server* zijn vergelijkbaar met de *Desktop* en *Lite* versies van Raspberry Pi OS.

De *Core*-versie van Ubuntu werkt anders dan de *Desktop* en *Server* versies. Deze versie heeft een ander pakketbeheersysteem genaamd Snap. Hierbij draaien alle applicaties in hun eigen containers en niet op het besturingssysteem zelf, waardoor conflicten tussen applicaties worden verminderd (Canonical, 2023). Deze versie van Ubuntu kan niet direct bediend worden, maar moet via een Ubuntu One account worden aangestuurd.

4.2 Uitlezen sensoren

De functie van de flight computer is om alle sensoren die aan de drone verbonden zitten uit te lezen en deze data door te sturen naar de flight controller. Omdat het gaat om meerdere sensoren, is er

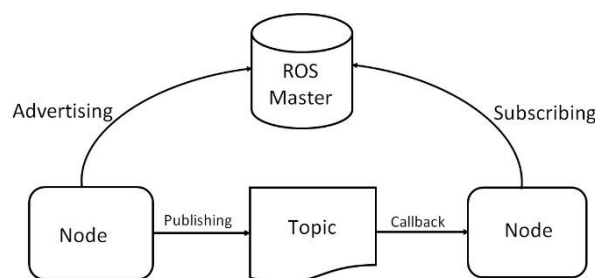
een systeem nodig om alle data van de sensoren uit te lezen en te verwerken. Een systeem wat hier geschikt voor is, is ROS (Robot Operating System). Hiervan zijn twee versies: ROS 1 en ROS 2.

4.2.1 ROS

ROS is een open source development kit voor robotica applicaties. Het biedt een standaard softwareplatform voor allerlei verschillende doeleinden. De processen in ROS zijn ingedeeld in een graafarchitectuur. Hierin zijn individuele processen gedefinieerd in nodes. Deze nodes kunnen sensordata, besturingsdata en andere berichten naar elkaar sturen en van elkaar ontvangen (Ademovic, 2015). ROS nodes kunnen geschreven worden in C++, Python of Lisp.

De nodes communiceren met elkaar via berichten die heen en weer worden gestuurd via zogenaamde topics. Dit versturen en ontvangen van berichten werkt via een publish/subscribe protocol, waarbij een node zich kan abonneren op een bepaalde topic. Als een andere node dan iets op deze topic publiceert, ontvangt de abonnerende node dit bericht (Tawil, 2017).

ROS 1 is de eerste versie van ROS en biedt de meeste ondersteuning voor verschillende soorten sensoren en software. Alle informatie over de verbindingen en communicatie tussen nodes wordt in ROS 1 geregeld door de ROS Master. Nodes kunnen zich hierbij registreren en aangeven welke berichten zij willen ontvangen en op welke topics zij willen publiceren/abonneren. De ROS Master houdt dan bij welke nodes actief zijn en welke topics beschikbaar zijn. Een overzicht hiervan is te zien in Figuur 4.1 (Wilcher, 2019). Het aanbevolen besturingssysteem voor ROS is Ubuntu Linux. Het is niet aanbevolen om op Raspberry Pi OS uitgevoerd te worden. Dit komt omdat de Linux versie waar Raspberry Pi OS op gebaseerd is niet wordt ondersteund door ROS (Robotics Backend, 2023).



Figuur 4.1: Overzicht ROS Master met nodes en berichten

4.2.2 ROS 2

Het doel van ROS 2 is om verder door te bouwen op alle positieve aspecten van ROS 1 en alle negatieve aspecten te verbeteren (Macenski, Foote, Gerkey, Lalancette, & Woodall, 2022). De grootste veranderingen zijn een nieuw communicatiesysteem die ondersteuning biedt voor DDS (Data Distribution Service) waardoor de ROS Master niet meer nodig is, een real-time laag, een modulaire architectuur en meer ondersteuning voor verschillende programmeertalen.

De nadelen van het gebruik van ROS 2 zijn de grootte van de community: het bestaat minder lang dan ROS 1 en er zijn dus minder voorbeelden beschikbaar. Ook is ROS 2 niet backward compatible met ROS 1 code.

ROS 2 kan op de Raspberry Pi op twee manieren gebruikt worden: met Ubuntu Linux of via Docker met Raspberry Pi OS (Alexander, 2022). ROS 2 kan zonder enige aanpassing op Ubuntu Linux uitgevoerd worden. Op deze manier heeft het Tier 1 ondersteuning. Dit houdt in dat deze versie

regelmatig getest wordt door het ontwikkelteam van ROS 2 (Arguedas, Ragnarok, Thomas, Nash, & Biggs, 2022). ROS 2 op Raspberry Pi OS heeft Tier 3 ondersteuning. Dit betekent dat deze niet meer bijgehouden wordt door de ontwikkelaars van ROS 2.

Om wel Tier 1 ondersteuning te krijgen op Raspberry Pi OS kan Docker gebruikt worden. Dit is een platform waarmee applicaties in containers gedraaid kunnen worden, zodat ze niet direct op het besturingssysteem draaien. Hiermee kunnen Ubuntu containers op Raspberry Pi OS gedraaid worden. Hier zitten echter enkele limieten aan. Zo heeft Docker geen high-level interface voor het bereiken van low-level hardware zoals GPIO. Om wel met GPIO te kunnen werken vanuit Docker, wordt vaak de `--privileged` optie gebruikt (Docker, 2023). Dit zorgt ervoor dat er veel meer van een OS blootgesteld wordt aan de container dan nodig is (Noury, 2021).

4.3 Conclusie flight computer

De flight computer zorgt ervoor dat de flight controller alleen bezig hoeft te zijn met het laten vliegen van de drone. Deze leest sensoren op de drone uit, stuurt data over de positie en besturing naar de flight controller en onderhoudt communicatie met de edge computer. Om dit te doen heeft het een besturingssysteem nodig. Hiervoor zijn de meest gebruikte besturingssystemen Raspberry Pi OS en Ubuntu. Deze hebben allebei verschillende versies.

De sensoren op de drone kunnen uitgelezen worden via ROS. Dit is een open source development kit voor robotica applicaties waarbij processen zijn ingedeeld in een graafarchitectuur, met de individuele processen gedefinieerd als nodes. Hiervan zijn twee versies beschikbaar: ROS 1 en ROS 2. ROS 1 is de oudere versie met meer ondersteuning en ROS 2 is de nieuwere versie met veel verbeteringen.

Op de flight computer kan ROS 1 uitgevoerd worden door Ubuntu te gebruiken. ROS 2 kan uitgevoerd worden op Ubuntu of op Raspberry Pi OS via Docker. Bij het gebruik van Docker moeten wel een aantal aanpassingen gemaakt worden op GPIO te kunnen gebruiken.

Hoe de aansturing van de drone is ontworpen, geïmplementeerd en getest wordt uitgelegd in het volgende hoofdstuk: Hoofdstuk 5.

5 Aansturing drone

Dit hoofdstuk beschrijft het ontwerp, de implementatie en het testen van de drone. De gemaakte keuzes naar aanleiding van de vorige hoofdstukken zullen worden uitgelegd. Ook wordt uitgelegd waarom bepaalde stappen zijn genomen in het ontwerp- en implementatieproces. Paragraaf 5.1 beschrijft het ontwerp voor het aansturen van de drone. Paragraaf 5.2 beschrijft hoe de flight controller en flight computer geconfigureerd zijn. Paragraaf 5.3 beschrijft hoe de aansturing geïmplementeerd is. In Paragraaf 5.4 wordt beschreven hoe de drone via 5G aangestuurd wordt. Als laatst beschrijft Paragraaf 5.5 hoe de aansturing getest is.

Op basis van de informatie uit Hoofdstuk 3 en Hoofdstuk 4 is de keuze gemaakt om ROS 2 te gebruiken voor het uitlezen van de sensoren en het samenvoegen van de sensordata. Deze keuze is gemaakt aan de hand van de verbeteringen genoemd in Paragraaf 4.2.2. Doordat ROS 2 geen gebruik meer maakt van de ROS Master, is er geen Single Point of Failure meer en is de kans kleiner dat de ROS installatie vastloopt als er iets fout gaat. Als versie is ROS 2 Humble gekozen. Deze keuze is gemaakt omdat dit de meest recente versie van ROS 2 is. Hierdoor is deze versie het meest bijgewerkt en bevat deze de meest recente bugfixes. Echter, zoals beschreven in Paragraaf 5.2.1, is uiteindelijk gekozen voor ROS 2 Foxy op Ubuntu 20. Deze keuze is gemaakt naar aanleiding van advies van PX4 ontwikkelaars en omdat dit betere resultaten gaf dan een nieuwere versie van ROS 2 en Ubuntu.

Omdat ArduPilot afhankelijk is van MAVROS en deze, zoals beschreven is in Paragraaf 3.3.1, nog in pre-alpha ontwikkeling is voor ROS 2, is gekozen om de flight controller aan te sturen met PX4. PX4 kan communiceren met ROS 2 via de PX4-ROS 2 Bridge via microRTPS of XRCE-DDS. Eerst was besloten om voor microRTPS te kiezen omdat hierbij de meest stabiele versie van PX4 gebruikt kan worden. Echter, zoals beschreven in Paragraaf 5.2.3, is uiteindelijk gekozen voor de nieuwste beta-versie van PX4 (v1.14) in combinatie met XRCE-DDS omdat deze veel verbeteringen bevat ten opzichte van microRTPS. Dit wordt in Paragraaf 5.2.3 verder uitgelegd.

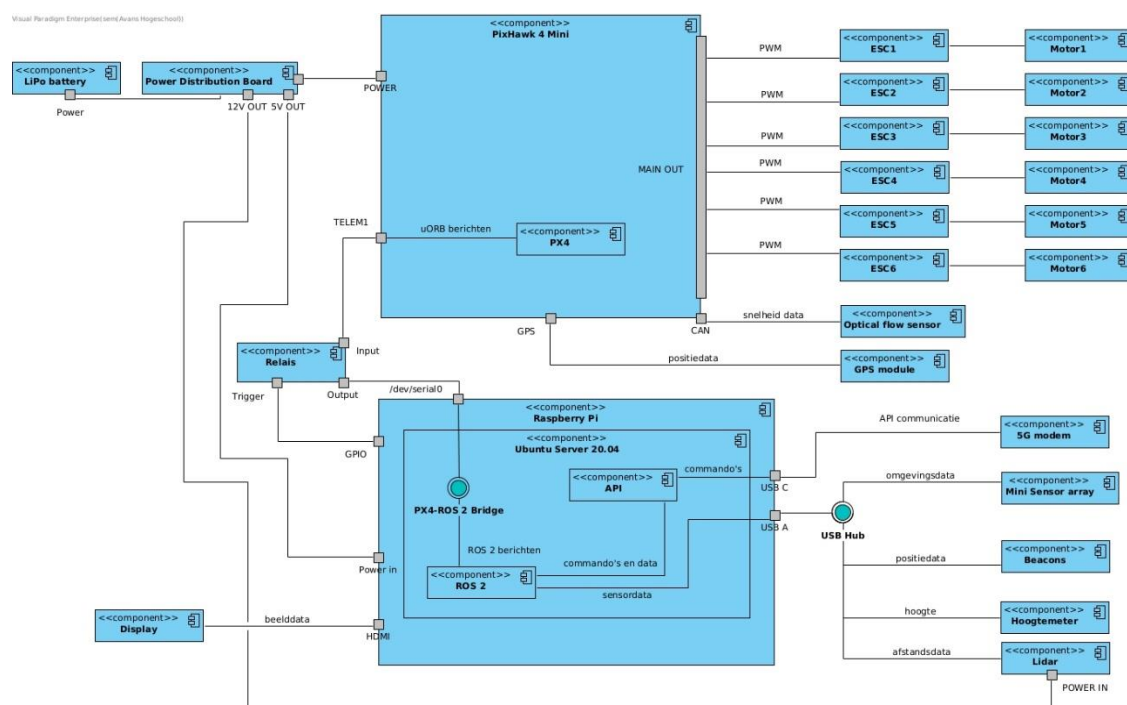
Om de flight computer aan te sturen is gekozen om Ubuntu Server op de Raspberry Pi te gebruiken. Deze keuze is gemaakt omdat op deze manier ROS 2 direct op het besturingssysteem kan draaien. De Server variant is gekozen omdat deze weinig resources gebruikt ten opzichte van de Desktop versie en deze kan direct benaderd worden zonder dat een Ubuntu One account nodig is zoals bij de Core versie. Ook is niet voor Raspberry Pi OS gekozen omdat hierbij Docker gebruikt moet worden om ROS 2 uit te voeren. Zoals besproken in Paragraaf 4.2.2, moeten er extra stappen genomen worden om de GPIO aan te kunnen sturen via Docker. Dit is niet wenselijk omdat de flight controller via GPIO aan de flight computer verbonden wordt.

5.1 Ontwerp

De flight controller wordt zoals beschreven in Paragraaf 3.3 via GPIO verbonden met de flight computer. De andere sensoren op de drone worden via USB aangesloten op de Raspberry Pi. De LIDAR heeft naast een USB verbinding ook 12V spanning nodig. Een schematisch overzicht van hoe de verschillende hardwarecomponenten van de drone met elkaar verbonden zijn is te zien in Figuur 5.1. Hierin is te zien dat de LIDAR, flight computer en flight controller worden gevoed door het PDB (Power Distribution Board) die aan de batterij verbonden zit. De flight controller stuurt de ESC's, en daarmee de motoren, aan via de MAIN OUT poort.

Op de flight controller draait PX4 en deze is via de TELEM1 poort verbonden met de UART GPIO pinnen van de flight computer. Hiertussen zit een relais die de communicatie pas inschakelt als de flight computer aanstaat. Dit is gedaan omdat de flight controller eerder opgestart is dan de flight computer als de batterij ingeschakeld wordt. Dit komt omdat ze allebei onafhankelijk door het PDB gevoed worden. Dit is een probleem omdat de flight controller al gelijk begint met sturen van data als deze opgestart is en de UART pinnen hiermee van spanning voorziet. Dit zorgt ervoor dat de flight computer niet goed opstart (Ansic, 2022). Om dit probleem op te lossen is gekozen om de communicatie pas toe te laten als de flight computer ingeschakeld is door deze het relais te laten schakelen. Dit relais heeft twee poorten waarmee de RX en TX communicatielijnen van de UART verbinding apart geschakeld kunnen worden. Er is gekozen voor een relais omdat deze simpelweg al beschikbaar was en op deze manier niets nieuws besteld hoefde te worden. Ook kan deze makkelijk door de flight computer aangestuurd worden. Het relais is verbonden aan GPIO pinnen 17 en 27, omdat deze fysiek dicht in de buurt van de andere aangesloten GPIO pinnen zitten en zo de bekabeling van de drone netjes blijft.

Op de flight computer draait Ubuntu Server 20.04, waarop ROS 2 en de API om de drone te besturen draaien. De berichten voor het aansturen van de drone komen via de API binnen en worden via de PX4-ROS 2 Bridge heen en weer gestuurd tussen de flight controller en flight computer. De flight computer leest de andere sensoren uit via de USB hub en de data hiervan wordt ook via ROS 2 verwerkt. De verbinding tussen de API en de edge computer gaat via de 5G modem die aan de USB C poort van de flight computer is aangesloten. Als laatste wordt het display via de HDMI poort aan de flight computer verbonden.



Figuur 5.1: Schematisch ontwerp hardware drone

In het schematisch ontwerp zijn ook een optical flow sensor en GPS module te zien. Deze zijn later in het project aangeschaft en worden besproken in Paragraaf 5.2.2.

De drone met een overzicht van alle onderdelen is te zien in Figuur 5.2. Hierin is bij nummer 2 een beacon voor het positioneren te zien. Deze is uiteindelijk vervangen door een GPS.

Alle blauwe interfaces in de packages zijn de services die de nodes aanbieden. Hierbij geeft een generalisatie aan dat de node die service aanbiedt. Alle roze interfaces zijn de topics waarop nodes kunnen abonneren en publiceren. Een aggregatie geeft aan dat een node op die topic publiceert. Een associatie tussen een node en een topic betekent dat die node zich op die topic abonneert.

De *ApiListener* node zorgt voor de communicatie met de API en de edge computer om bij te dragen aan eis *MH7*, *MH8* en *SH5*. Deze gebruikt de service *drone/picture* van de *CameraController* node, die de camera bedient. Deze node stuurt de videostream direct naar de edge computer om zo bij te dragen aan eis *MH6* en *SH6*.

De *ApiListener* gebruikt de *FollowRoute* service van de *RouteFollower* node om een route in te geven. Dit draagt bij aan eis *SH3* en *SH4*. Tijdens het vliegen van de route krijgt de *ApiListener* informatie over het proces door te abonneren op de *drone/route_status* topic. Deze informatie wordt dan weer naar de edge computer gestuurd om bij te dragen aan eis *SH2*. De *RouteListener* gebruikt de *drone/move_to_position* service van de *PositionChanger* node om de positie van de drone aan te passen. Voor het handmatig besturen van de drone gebruikt de *ApiListener* ook deze service. Verder gebruikt de *ApiListener* ook de *drone/ready* service van de *PositionChanger* om de drone te armen en de motoren te laten draaien voordat de drone gaat vliegen.

De *PositionChanger* gebruikt de *drone/set_trajectory* en *drone/set_attitude* services van de *PX4Controller* om de drone te laten bewegen, die de commando's doorstuurt naar PX4 op de flight controller. Deze publiceert de status van de drone op de *drone/arm_status* en *drone/control_mode* topics die respectievelijk aangeven of de drone gearmd is en of de drone op basis van houding, snelheid of positie bestuurd wordt.

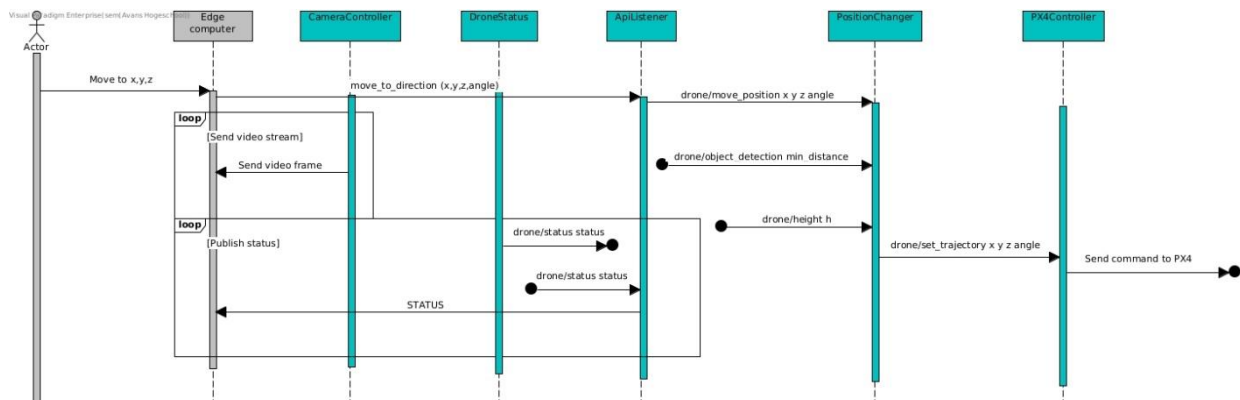
De *DroneStatus* node is op deze topics geabonneerd en maakt deze data beschikbaar voor weergave door het te publiceren op de *drone/status* topic. Zo draagt deze bij aan eis *SH2*. Hier voegt deze ook de status van de te volgen route bij die wordt ontvangen via de *drone/route_status* topic. Op de *drone/status* node is de *DisplayManager* node geabonneerd en deze laat de informatie op het scherm op de drone zien om bij te dragen aan eis *CH1*. De *RelaisController* node zet het relais om voor de communicatie met de flight controller. Deze biedt ook een service aan waarmee nodes de relais in- of uit kunnen schakelen.

De *PositionChanger* node is ook geabonneerd op de *drone/object_detection* topic, waar de *LidarReader* en *MultiflexReader* de objectdetectiedata op publiceren. Dit draagt bij aan eis *MH3* en *MH4*. Verder is deze geabonneerd op de *drone/position* topic waar de *TrackerPositioning* node de informatie van de beacons op publiceert. Dit draagt bij aan eis *MH5* en *SH1*. Als laatste publiceert de *HeightReader* node data over de hoogte onder de drone op het *drone/height* topic, waar de *PositionChanger* node ook op geabonneerd is.

De *FailSafe* node regelt de algemene failsafe als er iets mis is met de drone en deze een noodlanding moet maken. Deze biedt een *drone/enable_failsafe* service aan waarmee andere nodes de failsafe aan kunnen zetten. De service kan aangeroepen worden met een bericht waarin staat wat de reden voor de failsafe is. Als deze service aangeroepen wordt, publiceert de node dit op de *drone/failsafe* topic. Hierop is de *PX4Controller* geabonneerd en die zorgt dat de drone een noodlanding maakt als de failsafe aangezet wordt. De *PositionChanger* gebruikt deze service als de verbinding met de LIDAR wegvalt. De *ApiListener* gebruikt deze service als de verbinding met de edge computer wegvalt. Deze

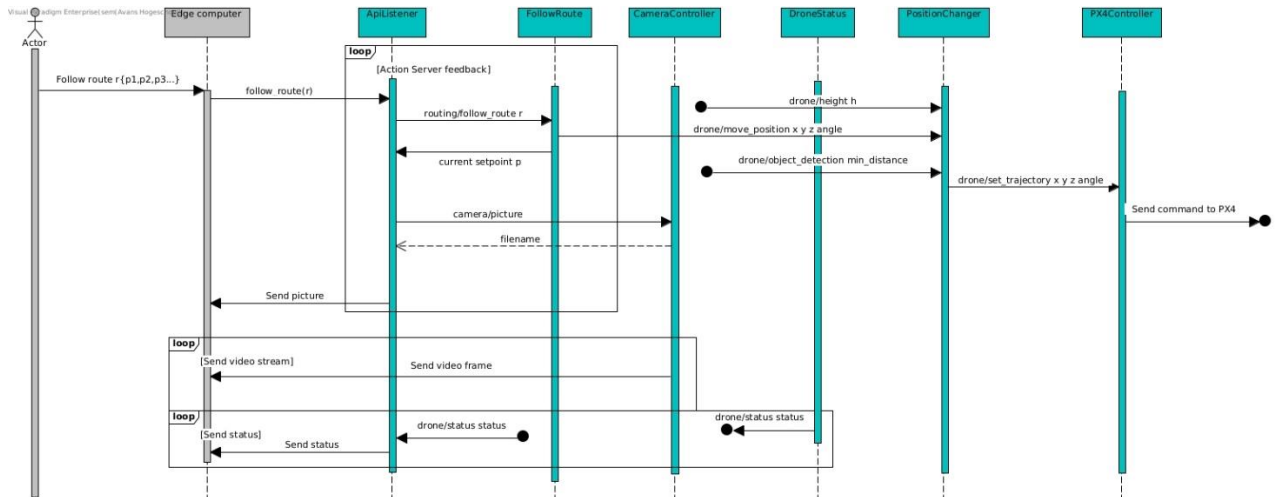
is ook op de *drone/failsafe* topic geabonneerd om aan de edge computer door te geven dat de failsafe is ingeschakeld, als de verbinding met de LIDAR is weggefallen.

Om de communicatie tussen de nodes duidelijk te maken, is in Figuur 5.4 een sequence diagram opgenomen die de communicatie bij het ontvangen van een bericht laat zien. Hierbij stuurt de gebruiker (*Actor*) een gewenste richting via de edge computer naar de drone. Dit wordt ontvangen door de *ApiListener* en die stuurt dit door via de *drone/move_position* service van de *PositionChanger*. Deze krijgt hierna via de *drone/object_detection* en *drone/height* topics respectievelijk de huidige objectdetectiedata en hoogte binnen. Hierna wordt de gewenste houding om naar die richting te bewegen naar de *PX4Controller* gestuurd via de *drone/set_trajectory* service. Deze stuurt hierna het commando voor de nieuwe setpoint naar PX4. Ook is te zien dat gedurende het hele proces de *CameraController* de video naar de edge computer stuurt en dat de *DroneStatus* de status van de drone naar de *ApiListener* stuurt, die dit weer naar de edge computer stuurt.



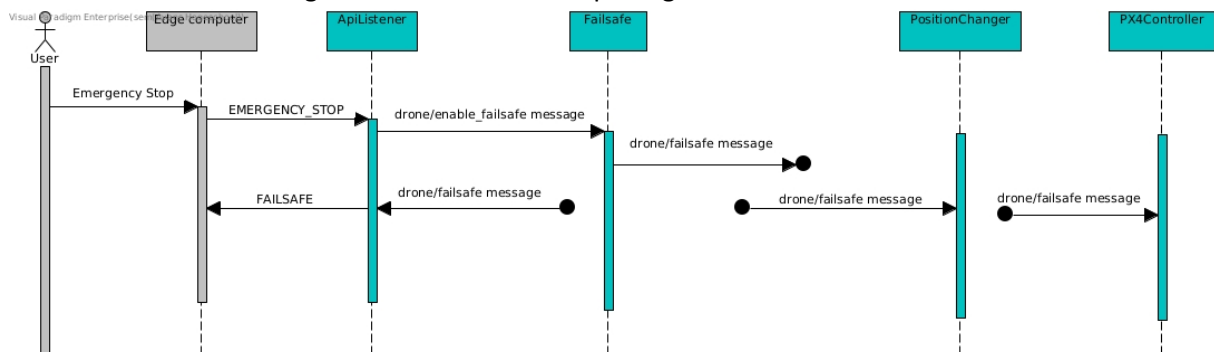
Figuur 5.4: Sequence diagram voor het ontvangen van een richting

De communicatie bij het volgen van een route is iets uitgebreider. Dit is weergegeven in Figuur 5.5. Hierbij stuurt de gebruiker een te volgen route met punten waarop een foto gemaakt moet worden naar de API. Dit wordt weer ontvangen door de *ApiListener* en deze stuurt de route via de Action Server van de *FollowRoute* node door. De *ApiListener* krijgt hierdoor continu feedback over het verloop van de route. De *FollowRoute* node stuurt telkens nieuwe posities naar de *PositionChanger*, die hierna de gewenste trajectory setpoints naar de *PX4Controller* stuurt zoals ook hierboven beschreven. Als de drone op een punt in de route komt waar een foto gemaakt moet worden, roept de *ApiListener* de *drone/picture* service van de *CameraController* node aan. Die reageert hierop met de bestandsnaam van de gemaakte foto. Gedurende dit proces wordt de videostream en status op dezelfde manier als bij Figuur 5.4 naar de edge computer gestuurd.



Figuur 5.5: Sequence diagram voor het ontvangen van een route

Een sequence diagram voor het aanzetten van de failsafe door op de noodstop te drukken is te zien in Figuur 5.6. Hierin is te zien dat als de gebruiker op de noodstop drukt, dit doorgestuurd wordt naar de *ApiListener*. Deze roept de *drone/enable_failsafe* aan met het bericht dat op de noodstop gedrukt is. Deze publiceert dit op de *drone/failsafe* topic waar de *PositionChanger* en *PX4Controller* op geabonneerd zijn. De *ApiListener* is hier ook op geabonneerd en stuurt een bericht naar de edge computer als hier een bericht op binnen komt. Op de edge computer wordt nu een melding gegeven met dat de failsafe is aangezet omdat de noodstop is ingedrukt.



Figuur 5.6: Sequence diagram aanzetten failsafe door noodstop

5.2 Configuratie

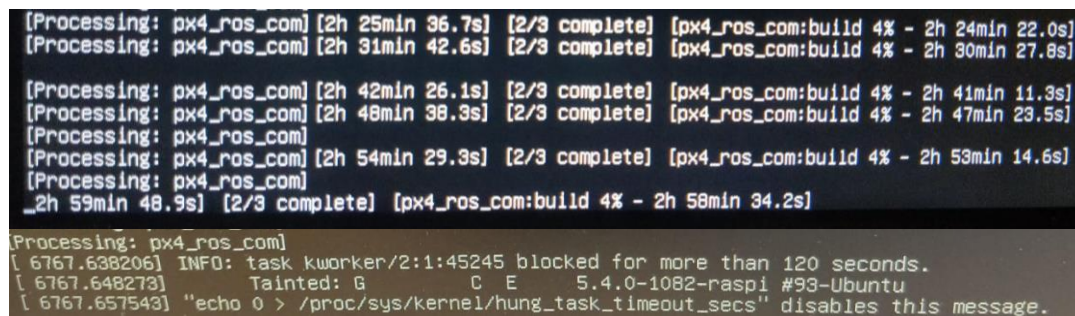
Om de flight computer en flight controller met elkaar te laten communiceren moeten deze eerst allebei geconfigureerd worden. De stappen die genomen zijn om deze configuratie uit te voeren worden in deze paragraaf benoemd.

5.2.1 Configuratie flight computer

Als eerst is Ubuntu Server 22.04 op de flight computer geïnstalleerd. Hierna is microRTPS geïnstalleerd volgens de handleiding van PX4 (Pixhawk, 2023a). Dit is een versie van RTPS die met PX4 gebruikt kan worden. Tijdens de installatie van Fast-DDS verscheen een error m.b.t. OpenSSL. Fast-DDS is een versie van DDS die met RTPS gebruikt kan worden. OpenSSL is een verzameling programma's voor cryptografie en beveiligde communicatie. Het probleem was dat de flight computer OpenSSL 3.0.2 gebruikte en Fast-DDS OpenSSL 1.0.2. Dit is opgelost door deze versie apart te compileren hierbij de parameter `-DOPENSSL_ROOT_DIR=/usr/local/ssl/*` mee te geven (Bilbao,

2017). Hierna is FastRTPS-Gen geïnstalleerd en kon ROS 2 geconfigureerd worden. FastRTPS-Gen is een programma om code te generen voor de koppeling tussen Fast-DDS en microRTPS.

ROS 2 Humble is geïnstalleerd en de PX4-ROS 2 Bridge is geïnstalleerd. Deze bestaat uit de packages `px4_msgs` voor de berichtdefinities tussen ROS 2 en PX4 en `px4_ros_com` voor het aansturen van PX4. De installatie van de PX4-ROS 2 Bridge gaf veel problemen. Er stonden fouten in de berichtdefinities van de `px4_msgs` ROS 2 package die handmatig opgelost moesten worden. Het bouwen en installeren van de `px4_ros_com` package duurde erg lang omdat de flight computer niet genoeg RAM had om deze package normaal te kunnen compileren. Dit is te zien in Figuur 5.7.



```
[Processing: px4_ros_com] [2h 25min 36.7s] [2/3 complete] [px4_ros_com:build 4% - 2h 24min 22.0s]
[Processing: px4_ros_com] [2h 31min 42.6s] [2/3 complete] [px4_ros_com:build 4% - 2h 30min 27.8s]

[Processing: px4_ros_com] [2h 42min 26.1s] [2/3 complete] [px4_ros_com:build 4% - 2h 41min 11.3s]
[Processing: px4_ros_com] [2h 48min 38.3s] [2/3 complete] [px4_ros_com:build 4% - 2h 47min 23.5s]
[Processing: px4_ros_com]
[Processing: px4_ros_com] [2h 54min 29.3s] [2/3 complete] [px4_ros_com:build 4% - 2h 53min 14.6s]
[Processing: px4_ros_com]
[Processing: px4_ros_com] [2h 59min 48.9s] [2/3 complete] [px4_ros_com:build 4% - 2h 58min 34.2s]

[Processing: px4_ros_com]
[ 6767.638206] INFO: task kworker/2:1:45245 blocked for more than 120 seconds.
[ 6767.648273] Tainted: G          C E          5.4.0-1082-raspi #93-Ubuntu
[ 6767.657543] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
```

Figuur 5.7: Het compileren van de `px4_ros_com` package duurt lang en de flight computer heeft niet genoeg RAM.

Om dit probleem op te lossen is geprobeerd om de configuratie en installatie van ROS 2 op een sterkere laptop over te nemen. Hierbij is het gelukt om de package te bouwen, maar bij het overzetten naar de flight computer lukte het niet om deze uit te voeren, omdat de installatie niet hetzelfde was. Toen het niet lukte om deze package op de Raspberry Pi te bouwen op Ubuntu Server 22.04, is geprobeerd om deze te bouwen op Ubuntu 20.04 met ROS 2 Foxy. Dit is gedaan omdat deze configuratie in de handleiding van PX4 ook gebruikt wordt en dit werd aangeraden door een ontwikkelaar van PX4 (Hoeven, Cannot arm drone with companion computer: “Arming denied! manual control lost”, 2023). Dit gaf echter hetzelfde probleem waarbij de installatie erg lang duurde en op een gegeven moment door het besturingssysteem werd gestopt omdat er geen RAM-geheugen meer vrij was.

Om de package toch te kunnen bouwen en de PX4-ROS 2 bridge te kunnen gebruiken, is een nieuwe Raspberry Pi flight computer besteld met 8 GB RAM in plaats van de 2 GB die de flight computer eerst had. Hiermee is eerst geprobeerd de package te bouwen volgens de handleiding met Ubuntu 20.04 en ROS 2 Foxy en dit is gelukt. Hierna is geprobeerd om de package te bouwen met Ubuntu 22.04 en de laatste versie van ROS 2. Hierbij werd echter weer een foutmelding gegeven bij het installeren van ROS 2 en bij het compileren van OpenSSL. Naar aanleiding hiervan is de keuze gemaakt om bij Ubuntu 20.04 en ROS 2 Foxy te blijven. Er is niet gekozen om Ubuntu 22 te gebruiken in combinatie met ROS 2 Foxy, omdat ROS 2 Foxy alleen (Tier 1) ondersteuning biedt voor Ubuntu 20.04 (Arguedas, Ragnarok, Thomas, Nash, & Biggs, 2022).

5.2.2 Configuratie flight controller

De flight controller is ingesteld met QGroundControl. De laatste versie hiervan gaf een foutmelding bij het uitvoeren en hierdoor is versie 4.1.7 gebruikt (Hoeven, Starting QGroundControl ApplImage gives symbol lookup error, 2023c). Dit is een *Daily* versie en niet de laatste stabiele versie. Er is voor een versie nieuwer dan de stabiele versie gekozen omdat PX4 vanaf v1.13.3 ondersteuning biedt voor meerdere batterijen d.m.v. `BATx_*` parameters. De stabiele versie van QGroundControl heeft dit niet en gaf hierdoor een foutmelding bij het instellen van het voltage voor de ESC's. De *Daily* versie biedt

hier wel ondersteuning voor. Middels de handleiding van de ESC's (SkyWalker, 2015) is de flight controller ingesteld met de waarden in Tabel 5.1.

Tabel 5.1: Waarden batterijinstellingen flight controller

Parameter	Waarde
Amount of cells	4
Empty voltage per cell	3.50 V
Full voltage per cell	4.05 V
Voltage divider	10.35605049
Amps per volt	6.5625 A

Om de vorm van het frame van de drone in te stellen is de parameter `SYS_AUTOSTART` op 6001 gezet (Pixhawk, 2022b). Om te zorgen dat de flight controller kan vliegen zonder afstandsbediening is de parameter `COM_RC_IN_MODE` op 1 gezet (Pixhawk, 2021). De interne sensoren van de flight controller zijn gekalibreerd en de ESC's zijn ook automatisch gekalibreerd.

Voor het plaatsen van de ESC's was eerst een positie onder de motoren gekozen. Om zeker te zijn dat deze plaatsing geen negatief effect heeft op de werking van de motoren, is op een forum gevraagd of dit een probleem zou worden (Hoeven, Can I mount ESCs underneath the rotors of my hexacopter drone like this? Will it impact flight performance?, 2023). Hierop werd geantwoord met het feit dat de motoren en propellers te klein zijn om de drone efficiënt te laten vliegen. Naar aanleiding hiervan is dit verder onderzocht, en hieruit is gebleken dat met de originele motoren en propellers de drone ongeveer 5,4 minuten vliegtijd kan halen (Hoeven, 5G RCID: Voorstel veranderen motoren, 2023).

Voor het frame van de drone worden Tarot 4006 620kV motoren met grotere 1355 propellers aangeraden. Deze kunnen een drone van 2,7 kg 14,5 minuten in de lucht houden (Tarot, 2023). Dit is gunstig, aangezien de RCID ongeveer 2,2 kg weegt (Hoeven, 5G RCID: Voorstel veranderen motoren, 2023). Deze nieuwe motoren en propellers zijn besteld, samen met nieuwe motorsteunen om deze op het frame te monteren. Hiermee is voldaan aan eis *MH2*. De nieuwe onderdelen en de propellers naast de oude motoren en propellers zijn te zien in Figuur 5.8. Voor de zekerheid is een extra motor besteld voor het geval dat er één defect raakt. De nieuwe propellers zijn erg groot en steken vrij ver buiten het frame uit. Tijdens dit project was er onvoldoende kennis en tijd om beschermkappen voor de propellers te 3D-printen zodat de drone geen beschadigingen veroorzaakt, mocht het ergens te dicht bij in de buurt komen. Het wordt dus aangeraden om deze in het vervolg van dit project te ontwerpen en te printen.



Figuur 5.8: Nieuwe onderdelen en vergelijking nieuwe en oude propellers

De flight controller publiceert niet standaard berichten over de status van de batterij en de belasting van de processor. Als deze data wel uitgelezen moet worden, moet de firmware van de flight controller aangepast worden om deze data toe te laten (Pixhawk, 2023a). Dit is gedaan door topics toe te voegen voor de `battery_status` en `cpu_load` berichten die door de flight controller worden verstuurd. Op deze manier konden deze waarden uitgelezen worden op de flight computer.

Omdat de drone binnen in de 5G Hub gaat vliegen, is het gebruik van GPS niet accuraat voor het bepalen van de positie. Hierom wordt het positiesysteem gebaseerd op beacons gebruikt. Omdat geen gebruik gemaakt wordt van GPS, kan de gewenste positie van de drone niet simpelweg naar de flight controller gestuurd worden. Hiervoor moet gebruik gemaakt worden van attitude setpoints. Dit houdt in dat niet de gewenste positie in de ruimte naar de flight controller gestuurd wordt, maar alleen een gewenste houding en snelheid van de motoren (Pixhawk, 2023c). Dit is te zien in video 1 en video 2 uit Bijlage C: Overzicht video's. Echter is gebleken dat de besturing hiermee niet erg accuraat is en worden de functies van de flight controller voor het stabiel aansturen van de drone via een PID-regeling bijna niet gebruikt.

Om alle functies van de flight controller te kunnen gebruiken en de drone aan te kunnen sturen op basis van een gewenste snelheid heeft de flight controller een *schatting* van de positie nodig (Pixhawk, 2023c). Er is getest of een voldoende schatting van de positie verkregen kan worden door toch een GPS te bestellen en deze op de flight controller aan te sluiten. Dit gaf echter wisselende resultaten. De drone kan bestuurd worden op basis van snelheid, maar alleen als de GPS-verbinding voldoende is. Dit is soms het geval als de drone binnen vliegt, maar is niet stabiel genoeg om betrouwbaar mee te vliegen.

Hierna is gekeken naar het verkrijgen van een schatting van de positie van de drone op basis van *optical flow*. Hiermee kan de snelheid van de drone (en een schatting van de positie) bepaald worden door het gebruik van een naar beneden gerichte camera en afstandssensor aan de onderkant van de drone (Pixhawk, 2023e). Volgens de documentatie zijn de twee best werkende opties de *Ark Flow* sensor (te zien in Figuur 5.9) en een module gebaseerd op de PMW3901 sensor (Pixhawk, 2023e). De *Ark Flow* module is alleen in Amerika beschikbaar en misschien niet voor het einde van het project geleverd, daarom is deze niet gekozen.

Voor een module op basis van de PWM3901 zijn er verschillende opties die op SPI, UART of CAN werken (Pixhawk, 2022c). Omdat de flight controller geen SPI ingang heeft, is niet voor een module die op SPI werkt gekozen. Hieruit blijven de *Hex HereFlow* (te zien in Figuur 5.10) en de *Thone ThoneFlow* (te zien in Figuur 5.11) modules over. De *HereFlow* sensor bevat naast de PWM3901 ook een afstandssensor, in tegenstelling tot de *ThoneFlow*, die alleen de PWM3901 bevat. Op basis hiervan is de *HereFlow* module gekozen, omdat dit een completere oplossing is en hierbij geen aparte afstandssensor toegevoegd moet worden.



Figuur 5.9: Ark Flow sensor



Figuur 5.10: Hex HereFlow sensor



Figuur 5.11: Thone ThoneFlow sensor

Om de sensor aan te sluiten is de parameter `EKF2_OF_CTRL` op 1 gezet. Hierbij zijn ook de parameters `EKF2_OF_POS_X`, `EKF2_OF_POS_Y`, `EKF2_OF_POS_Z` aangepast naar de positie van de sensor ten opzichte van de flight computer. Na het instellen van deze parameters werd de sensor nog niet herkend door de flight controller. Figuur 5.12 weergeeft dat de optical flow module in de flight controller niet aan staat als het commando uitgevoerd wordt om te kijken of optical flow berichten gestuurd worden.

```
nsh: sysinit: fopen failed: No such file or directory
NuttShell (NSH) NuttX-11.0.0
nsh> listener estimator_optical_flow_vel

TOPIC: estimator_optical_flow_vel 4 instances

Instance 0:

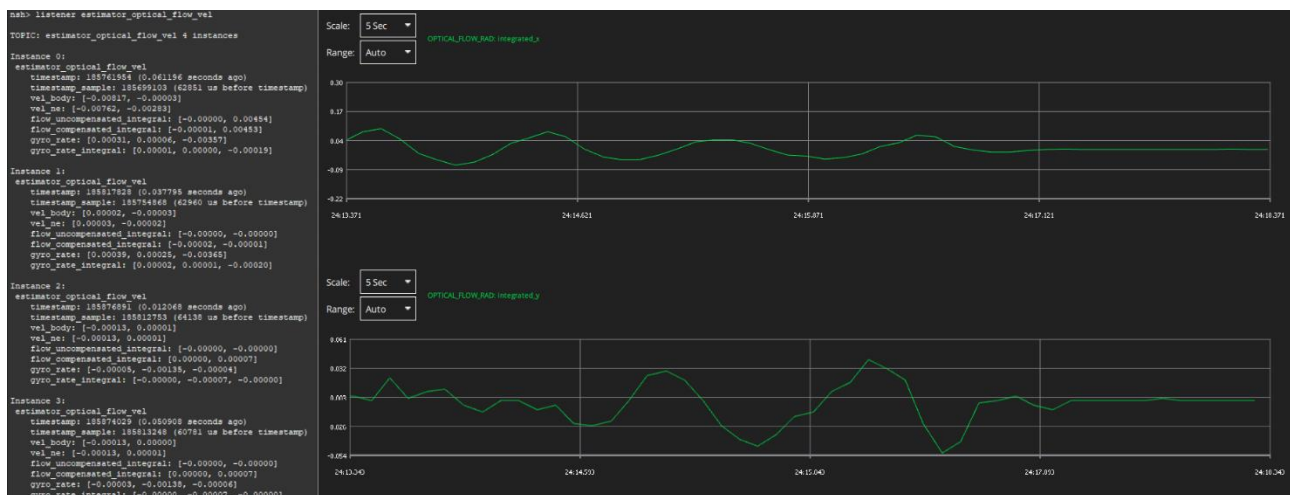
Instance 1:

Instance 2:

Instance 3:
```

Figuur 5.12: optical flow module in flight controller niet werkend

De sensor werkt via CAN en de flight controller heeft een aantal parameters om bepaalde CAN modules in of uit te schakelen. Hierbij horen ook de parameters `UAVCAN_SUB_FLOW` en `UAVCAN_SUB_RNG` (Pixhawk, 2023h). Over deze parameters staat niets in de PX4 documentatie, maar deze zijn om te testen toch op 1 gezet. Dit bleek de oplossing te zijn, want hierna werd de optical flow module uitgevoerd en kon de data hiervan uitgelezen worden. Dit is te zien in Figuur 5.13. Hierin is links te zien dat de module aan staat en rechts een grafiek met bovenin de x-waarden en onderin de y-waarden die de sensor meet als de drone heen en weer bewogen wordt.



Figuur 5.13: optical flow module staat aan en data kan uitgelezen worden

Nadat de flight controller correct ingesteld was om de optical flow sensor uit te lezen, is begonnen met testen van hoe de drone met deze sensor vliegt. Bij het armen werd echter een foutmelding gegeven over dat de schatting van de snelheid niet juist was. Dit is te zien in Figuur 5.14.



Figuur 5.14: foutmeldingen over de snelheid bij het armen

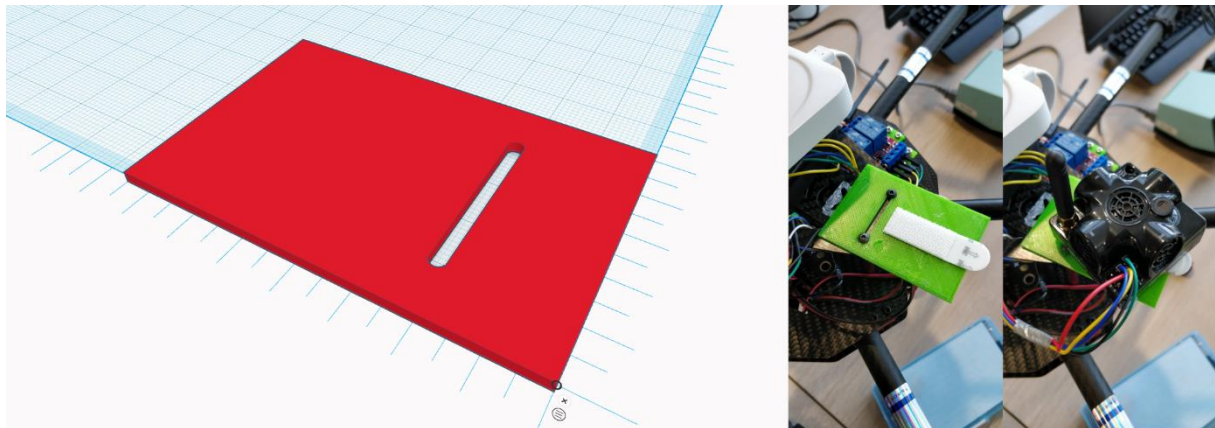
Toen dit opgezocht werd is gevonden dat de v1.14-beta versie van PX4 die voor dit project gebruikt wordt nog één probleem heeft dat ervoor zorgt dat de versie niet officieel uitgegeven kan worden: bij het armen van de drone wordt altijd een foutmelding gegeven over de snelheid bij het gebruik van een optical flow sensor (Klimaj, 2023). Er wordt actief aan dit probleem gewerkt, maar het is niet duidelijk wanneer een oplossing beschikbaar zal zijn en of dit op tijd zal zijn voor het einde van dit

project. Om de drone verder te ontwikkelen na dit project wordt aanbevolen om de ontwikkeling van PX4 in de gaten te houden en de optical flow sensor te gebruiken om binnen te vliegen zodra deze werkt met PX4.

Als laatste poging om de drone binnen te laten vliegen is verder onderzoek gedaan en gekeken naar een set van sensoren genaamd MarvelMind Indoor Navigation System (MarvelMind Robotics, 2021). Dit is een beacon positioning systeem die werkt met een aantal stationaire beacons in de ruimte en een mobiele beacon op de drone. Deze communiceren dan via een centrale modem. De beacons communiceren op basis van radio en bieden positiebepaling op ± 2 cm nauwkeurig. Deze zijn onderzocht omdat ze een out-of-the-box integratie met PX4 bieden waarbij de mobiele beacon op de GPS-poort van de flight controller aangesloten kan worden. Hierbij lijkt het voor de flight controller alsof deze gewone GPS coördinaten ontvangt, maar dit is eigenlijk de lokale positie in de ruimte omgeven door de beacons (MarvelMind Robotics, 2022).

Deze set beacons kan op verschillende manieren ingesteld worden. Voor de drone is een Non-Inverse Architecture (NIA) opstelling het beste waarbij de mobiele beacon data verstuurt en de stationaire beacons data ontvangen. Deze opstelling is gekozen omdat dit de gebruikelijke opstelling is voor drones (MarvelMind Robotics, 2023). De werking van de beacons is te zien in video 3 uit Bijlage C: Overzicht video's. Bij het configureren van de beacons werd het duidelijk dat een licentie nodig was om de PX4 integratie te gebruiken. Deze licentie moet aangeschaft worden, waarna een mail met gegevens over de beacon op de drone naar MarvelMind Robotics gestuurd moet worden. Hierna stuurt Marvelmind Robotics een mail terug met een activatiecode om de licentie te mogen gebruiken (MarvelMind Robotics, 2023).

Om de mobiele beacon op de drone te bevestigen is een plateau ontworpen en geprint met een 3D-printer. Het ontwerp en het uiteindelijke onderdeel met de beacon bevestigd zijn te zien in Figuur 5.15. De beacon is gedraaid bevestigd om zo de sensoren niet te blokkeren met de flight computer.



Figuur 5.15: 3D-print onderdeel om beacon te bevestigen

Uiteindelijk zijn de beacons geconfigureerd en aangesloten op de flight controller. Hierbij is het echter niet gelukt om een accurate schatting van de positie te genereren. Dit komt omdat, als de beacons in verschillende hoeken van de 5G Hub hangen, er te veel obstructies zijn voor een accurate positiebepaling van de drone. De flight controller gaf nog steeds foutmeldingen over de snelheid zoals te zien in Figuur 5.14. Omdat dit gebeurde terwijl er nog maar weinig tijd over was voor het project, is gekozen om alleen buiten te vliegen met GPS. Deze keuze is gemaakt om zo toch nog een bruikbare demo op te leveren en een vliegende drone te realiseren. Hierdoor kan niet worden

voldaan aan eis *MH5*, doordat GPS gebruikt wordt voor de positiebepaling en zo de twee beacons niet meer nodig zijn. Ook kan niet volledig worden voldaan aan eis *MH9* omdat de drone buiten zal vliegen.

5.2.3 Verbinding tussen flight computer en flight controller

De connectie tussen de flight controller en flight computer is gemaakt zoals beschreven in Paragraaf 3.3. Hierna is geprobeerd om eerst handmatig via MAVProxy te communiceren met de drone. Dit is een Python library om met MAVLink apparaten te communiceren. De communicatie is gelukt en er kan verbinding gemaakt worden met de flight controller door het commando `mavproxy.py --master=/dev/serial0 --baudrate=57600` te gebruiken (The Drone Dojo, 2021). Dit betekent dat de verbinding tussen de flight controller en flight computer werkt.

Hierbij is het gelukt de verschillende modes van de flight controller op te vragen, maar bij het inschakelen van de motoren verscheen de foutmelding `COMPONENT_ARM_DISARM:`

`TEMPORARILY_REJECTED. AP: Arming denied! manual control lost.` Hierover is een vraag gesteld op het PX4 forum (Hoeven, Cannot arm drone with companion computer: “Arming denied! manual control lost”, 2023). Hierop heeft een ontwikkelaar van PX4 geantwoord. Deze ontwikkelaar gaf aan dat de parameter `COM_RC_IN_MODE` op 4 gezet moest worden om te kunnen vliegen zonder afstandsbediening, maar als dit gedaan werd verscheen een foutmelding “*value must be within 0 and 2*”.

Op dit moment gebruikte de flight controller nog versie v1.13.3 van PX4. Op het PX4 forum werd aangeraden om de v1.14 beta versie te gebruiken, omdat de PX4 ROS 2 bridge hierin erg verbeterd is. Dit advies is opgevolgd en de PX4 versie op de flight controller is veranderd naar de beta versie. Tot nu toe werd QGroundControl op een Linux computer uitgevoerd. Om te kijken of dit probleem niet aan het besturingssysteem lag, is het programma ook op een Windows computer geïnstalleerd. Hierbij kon de parameter zonder problemen veranderd worden.

Verder is de parameter `COM_RCL_EXCEPT` op 5 gezet om ervoor te zorgen dat geen afstandsbediening gebruikt hoeft te worden in *Offboard mode* (bestuurd door een flight computer) (Pixhawk, 2023d). Hierna is de PX4-ROS 2 bridge geïnstalleerd met XRCE-DDS. De baudrate is aangepast naar 921600 omdat met de lagere baudrate van 57600 een *RTT Timesync too high* foutmelding verkregen werd. Hiermee kon de flight controller uiteindelijk bestuurd worden via de ROS 2 bridge. Hiermee is voldaan aan eis *MH1*.

5.3 Implementatie

Om de code veilig te kunnen testen zonder dat de drone gevaarlijke situaties veroorzaakt is een simulator ingesteld waarmee de drone gesimuleerd kan worden. Hiermee wordt de flight controller op een drone gesimuleerd en kunnen de ROS 2 nodes getest worden zonder dat er met een echte drone gevlogen hoeft te worden. Deze simulator kan samen met de firmware voor de flight controller gedownload worden. Deze is op een Ubuntu 20 virtual machine gezet om de PX4-ROS2 bridge ook te kunnen simuleren.

5.3.1 Aansturing flight controller

Om de motoren in te kunnen schakelen in *Offboard mode* moet de flight controller eerst een goede verbinding hebben met de flight computer. Hiervoor moet de flight computer een aantal *OffboardControlMode* berichten sturen waarin staat welke mate van besturing de flight computer

biedt. Deze mate van besturing kan zijn op basis van positie, snelheid, acceleratie, houding of aansturing van individuele motoren. Deze berichten moeten met een frequentie van minimaal 2 Hz gestuurd worden om een gezonde verbinding te representeren voordat de motoren ingeschakeld kunnen worden (Pixhawk, 2023c). De drone bevat besturing op basis van positie, snelheid of houding. Deze functionaliteit is opgenomen in een ROS 2 Heartbeat node. De volledige code voor deze node is te zien in Bijlage A.3: Heartbeat node. Deze node stuurt *OffboardControlMode* berichten met de huidige mate van besturing met een frequentie van 100 Hz. Het versturen van deze berichten is te zien in Codefragment 5.1. Hierbij worden de individuele maten van besturing aangepast naar de waarde die de node binnen krijgt.

```

1  /**
2  * @brief Publish offboard control mode messages as a heartbeat.
3  *      Only the attitude is enabled, because that is how the drone will be controlled.
4  */
5  void send_heartbeat()
6  {
7      auto msg = px4_msgs::msg::OffboardControlMode();
8      msg.position = (this->control_mode >> CONTROL_POSITION_POS) & 1 ? true : false;
9      msg.velocity = (this->control_mode >> CONTROL_VELOCITY_POS) & 1 ? true : false;
10     msg.acceleration = (this->control_mode >> CONTROL_ACCELERATION_POS) & 1 ? true : false;
11     msg.attitude = (this->control_mode >> CONTROL_ATTITUDE_POS) & 1 ? true : false;
12     msg.body_rate = (this->control_mode >> CONTROL_BODY_RATE_POS) & 1 ? true : false;
13     msg.actuator = (this->control_mode >> CONTROL_ACTUATOR_POS) & 1 ? true : false;
14     // get timestamp and publish message
15     msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
16     offboard_control_mode_publisher->publish(msg);
17 }

```

Codefragment 5.1: versturen van *OffboardControlMode* berichten

De mate van besturing kan aangepast worden door de *SetVehicleControl* service zoals te zien in Figuur 5.3. Als deze service aangeroepen wordt, wordt de nieuwe mate van besturing gepubliceerd op de *drone/control_mode* topic. Het afvangen van requests naar deze service en het publiceren op de topic is te zien in Codefragment 5.2. Als een request wordt ontvangen wordt eerst gecontroleerd of deze de juiste mate van besturing bevat (regel 40 - 43).

```

1  /**
2  * @brief Publish the current control mode of the drone onto the 'drone/control_mode' topic
3  *
4  */
5  void publish_new_control_mode()
6  {
7      auto msg = drone_services::msg::DroneControlMode();
8      if (this->control_mode & (1 << CONTROL_ATTITUDE_POS))
9      {
10         msg.control_mode = CONTROL_MODE_ATTITUDE;
11         RCLCPP_INFO(this->get_logger(), "set control mode to attitude");
12     }
13     else if (this->control_mode & (1 << CONTROL_VELOCITY_POS))
14     {
15         msg.control_mode = CONTROL_MODE_VELOCITY;
16         RCLCPP_INFO(this->get_logger(), "set control mode to velocity");
17     }
18     else if (this->control_mode & (1 << CONTROL_POSITION_POS))
19     {
20         msg.control_mode = CONTROL_MODE_POSITION;
21         RCLCPP_INFO(this->get_logger(), "set control mode to position");
22     }
23     RCLCPP_INFO(this->get_logger(), "publishing new control mode %d", msg.control_mode);
24     drone_control_mode_publisher->publish(msg);
25 }
26 /**
27 * @brief Handle a request to change the vehicle control mode
28 *
29 * @param request_header the header of the service request
30 * @param request the request that was sent to the service
31 * @param response the response that the service sends back to the client.
32 */
33 void handle_vehicle_control_change(

```

```

34     const std::shared_ptr<rmw_request_id_t> request_header,
35     const std::shared_ptr<drone_services::srv::SetVehicleControl::Request> request,
36     const std::shared_ptr<drone_services::srv::SetVehicleControl::Response> response)
37     {
38         if (request->control < 0 || request->control > (1 << CONTROL_POSITION_POS))
39         {
40             response->success = false;
41         }
42         else
43         {
44             this->control_mode = request->control;
45             RCLCPP_INFO(this->get_logger(), "set control mode to %d", this->control_mode);
46             publish_new_control_mode();
47             response->success = true;
48         }
49     }

```

Codefragment 5.2: afvangen requests SetVehicleControl en publiceren nieuwe mate van besturing

1 Naast dat de Heartbeat node de verbinding met de flight controller onderhoudt, zorgt de PX4Controller node ervoor dat de flight controller aangestuurd kan worden. De volledige code hiervoor is te zien in Bijlage A.2: PX4Controller node. Deze heeft een service voor het aanpassen van de houding van de drone via attitude setpoints als de control mode van de drone op `CONTROL_MODE_ATTITUDE` staat. Deze functionaliteit is te zien in /**

```

2  * @brief handles reception of attitude setpoints through service
3  *
4  * @param request_header the header of the request
5  * @param request the request containing the new attitude setpoint
6  * @param response the response indicating if the setpoint was changed successfully
7  */
8  void handle_attitude_setpoint(
9      const std::shared_ptr<rmw_request_id_t> request_header,
10     const std::shared_ptr<drone_services::srv::SetAttitude::Request> request,
11     const std::shared_ptr<drone_services::srv::SetAttitude::Response> response)
12     {
13         if (this->failsafe_enabled)
14         {
15             RCLCPP_INFO(this->get_logger(), "Failsafe enabled, ignoring attitude setpoint");
16             response->success = false;
17             return;
18         }
19         if (armed)
20         {
21             if (request->yaw == 0 && request->pitch == 0 && request->roll == 0 && request->thrust == 0)
22             {
23                 last_setpoint[0] = degrees_to_radians(request->yaw);
24                 last_setpoint[1] = degrees_to_radians(request->pitch);
25                 last_setpoint[2] = degrees_to_radians(request->roll);
26                 last_thrust = request->thrust;
27                 RCLCPP_INFO(this->get_logger(), "STOPPING MOTORS");
28             }
29             else
30             {
31                 last_setpoint[0] += degrees_to_radians(request->yaw);
32                 last_setpoint[1] += degrees_to_radians(request->pitch);
33                 last_setpoint[2] += degrees_to_radians(request->roll);
34                 last_thrust += request->thrust;
35                 if (last_thrust > 1)
36                     last_thrust = 1;
37                 else if (last_thrust < 0)
38                     last_thrust = 0;
39             }
40
41             RCLCPP_INFO(this->get_logger(), "got values: yaw:%f pitch:%f roll:%f thrust:%f", request->yaw, request->pitch, request->roll, request->thrust);
42             RCLCPP_INFO(this->get_logger(), "New setpoint: yaw:%f pitch:%f roll:%f thrust:%f", last_setpoint[0], last_setpoint[1], last_setpoint[2], last_thrust);
43
44             response->success = true;
45         }
46         else
47         {
48             last_thrust = 0;
49             last_setpoint[0] = 0;
50             last_setpoint[1] = 0;
51             last_setpoint[2] = 0;
52             response->success = false;
53         }
54     }

```

Codefragment 5.3. Hierin wordt de huidige attitude van de drone aangepast met de waarden uit de nieuwe setpoint (regels 32 - 39). Als de failsafe geactiveerd is, wordt de request afgewezen (regels 14 - 19). Als een setpoint wordt ontvangen zonder een verandering van houding en waarbij de motoren niet draaien, wordt dit afgehandeld als een noodstop (regels 22 - 29). Als een setpoint wordt

ontvangen terwijl de drone nog niet gearmd is, wordt dit weergegeven door `false` terug te geven (regels 47 - 54).

```

55  /**
56   * @brief handles reception of attitude setpoints through service
57   *
58   * @param request_header the header of the request
59   * @param request the request containing the new attitude setpoint
60   * @param response the response indicating if the setpoint was changed successfully
61   */
62  void handle_attitude_setpoint(
63      const std::shared_ptr<rmw_request_id_t> request_header,
64      const std::shared_ptr<drone_services::srv::SetAttitude::Request> request,
65      const std::shared_ptr<drone_services::srv::SetAttitude::Response> response)
66  {
67      if (this->failsafe_enabled)
68      {
69          RCLCPP_INFO(this->get_logger(), "Failsafe enabled, ignoring attitude setpoint");
70          response->success = false;
71          return;
72      }
73      if (armed)
74      {
75          if (request->yaw == 0 && request->pitch == 0 && request->roll == 0 && request->thrust == 0)
76          {
77              last_setpoint[0] = degrees_to_radians(request->yaw);
78              last_setpoint[1] = degrees_to_radians(request->pitch);
79              last_setpoint[2] = degrees_to_radians(request->roll);
80              last_thrust = request->thrust;
81              RCLCPP_INFO(this->get_logger(), "STOPPING MOTORS");
82          }
83          else
84          {
85              last_setpoint[0] += degrees_to_radians(request->yaw);
86              last_setpoint[1] += degrees_to_radians(request->pitch);
87              last_setpoint[2] += degrees_to_radians(request->roll);
88              last_thrust += request->thrust;
89              if (last_thrust > 1)
90                  last_thrust = 1;
91              else if (last_thrust < 0)
92                  last_thrust = 0;
93          }
94
95          RCLCPP_INFO(this->get_logger(), "got values: yaw:%f pitch:%f roll:%f thrust:%f", request->yaw, request->pitch, request->roll, request->thrust);
96          RCLCPP_INFO(this->get_logger(), "New setpoint: yaw:%f pitch:%f roll:%f thrust:%f", last_setpoint[0], last_setpoint[1], last_setpoint[2], last_thrust);
97
98          response->success = true;
99      }
100     else
101     {
102         last_thrust = 0;
103         last_setpoint[0] = 0;
104         last_setpoint[1] = 0;
105         last_setpoint[2] = 0;
106         response->success = false;
107     }
108 }

```

Codefragment 5.3: methode ontvangen attitude setpoints

```

1  Het ontvangen van trajectory setpoints is te zien in /**
2   * @brief handles reception of trajectory setpoints through service
3   *
4   * @param request_header the header of the request
5   * @param request the request containing the new trajectory setpoint (velocity or position)
6   * @param response the response indicating if the setpoint was changed successfully
7   */
8  void handle_trajectory_setpoint(
9      const std::shared_ptr<rmw_request_id_t> request_header,
10     const std::shared_ptr<drone_services::srv::SetTrajectory::Request> request,
11     const std::shared_ptr<drone_services::srv::SetTrajectory::Response> response)
12  {
13      if (this->failsafe_enabled)
14      {
15          RCLCPP_INFO(this->get_logger(), "Failsafe enabled, ignoring trajectory setpoint");
16          response->success = false;
17          return;
18      }
19      if (!(request->control_mode == CONTROL_MODE_VELOCITY || request->control_mode == CONTROL_MODE_POSITION))
20      {
21          RCLCPP_INFO(this->get_logger(), "Got invalid trajectory control mode: %d", request->control_mode);
22          response->success = false;
23      }
24      else
25      {
26          RCLCPP_INFO(this->get_logger(), "Got new trajectory setpoint with control mode: %d", request->control_mode);
27          if (request->control_mode == CONTROL_MODE_VELOCITY)
28          {
29              for (int i = 0; i < 3; i++)
30              {
31                  velocity[i] = request->values[i];
32              }
33              RCLCPP_INFO(this->get_logger(), "Got new velocity setpoint. %f %f %f", velocity[0], velocity[1], velocity[2]);
34          }
35          else if (request->control_mode == CONTROL_MODE_POSITION)

```

```

36         {
37             position[0] += request->values[0];
38             position[1] += request->values[1];
39             position[2] -= request->values[2]; // height is negative
40
41             RCLCPP_INFO(this->get_logger(), "Got new position setpoint: %f %f %f", position[0], position[1], position[2]);
42         }
43
44         last_angle = request->yaw;
45         new_setpoint = true;
46         RCLCPP_INFO(this->get_logger(), "Yaw: %f", last_angle);
47         response->success = true;
48     }
49 }

```

Codefragment 5.4. Hierbij wordt eerst gekeken of de huidige `control_mode` juist is. Als dit niet zo is, wordt `false` teruggestuurd (regel 19 – 23). Als deze wel juist is, wordt eerst gekeken of het gaat om een setpoint voor de snelheid of de positie. Als de failsafe al geactiveerd is, wordt een request afgewezen (regels 13 - 18). Het aanpassen van de snelheid is absoluut: de snelheid wordt aangepast naar de ontvangen waarde (regel 27 - 34). Het aanpassen van de positie is relatief: de ontvangen waarden worden bij de huidige positie opgeteld (regel 35 - 42). Als laatst wordt de hoek aangepast naar de ontvangen waarde en wordt `true` teruggestuurd (regel 44 - 47). Bij het versturen van de setpoints wordt voor de positie gebruik gemaakt van poolcoördinaten. Hiermee kan gemakkelijk de gewenste relatieve positie berekend worden en kan hierbij ook de snelheid berekend worden.

```

50 /**
51  * @brief handles reception of trajectory setpoints through service
52  *
53  * @param request_header the header of the request
54  * @param request the request containing the new trajectory setpoint (velocity or position)
55  * @param response the response indicating if the setpoint was changed successfully
56  */
57 void handle_trajectory_setpoint(
58     const std::shared_ptr<rmw_request_id_t> request_header,
59     const std::shared_ptr<drone_services::srv::SetTrajectory::Request> request,
60     const std::shared_ptr<drone_services::srv::SetTrajectory::Response> response)
61 {
62     if (this->failsafe_enabled)
63     {
64         RCLCPP_INFO(this->get_logger(), "Failsafe enabled, ignoring trajectory setpoint");
65         response->success = false;
66         return;
67     }
68     if (!(request->control_mode == CONTROL_MODE_VELOCITY || request->control_mode == CONTROL_MODE_POSITION))
69     {
70         RCLCPP_INFO(this->get_logger(), "Got invalid trajectory control mode: %d", request->control_mode);
71         response->success = false;
72     }
73     else
74     {
75         RCLCPP_INFO(this->get_logger(), "Got new trajectory setpoint with control mode: %d", request->control_mode);
76         if (request->control_mode == CONTROL_MODE_VELOCITY)
77         {
78             for (int i = 0; i < 3; i++)
79             {
80                 velocity[i] = request->values[i];
81             }
82             RCLCPP_INFO(this->get_logger(), "Got new velocity setpoint. %f %f %f", velocity[0], velocity[1], velocity[2]);
83         }
84         else if (request->control_mode == CONTROL_MODE_POSITION)
85         {
86             position[0] += request->values[0];
87             position[1] += request->values[1];
88             position[2] -= request->values[2]; // height is negative
89
90             RCLCPP_INFO(this->get_logger(), "Got new position setpoint: %f %f %f", position[0], position[1], position[2]);
91         }
92
93         last_angle = request->yaw;
94         new_setpoint = true;
95         RCLCPP_INFO(this->get_logger(), "Yaw: %f", last_angle);
96         response->success = true;
97     }
98 }

```

Codefragment 5.4: ontvangen trajectory setpoints

De PX4Controller biedt ook functionaliteit om de drone te armen (Codefragment 5.5) of te disarmen (Codefragment 5.6). Bij beide fragmenten wordt eerst gekeken of de drone al gearmd is en of het request voor het armen of disarmen juist is. Hierna wordt de drone gearmd of gedisarmd middels een MAVLink commando. Bij het armen wordt de drone eerst in *Offboard mode* gezet door hiervoor ook een MAVLink commando te sturen.

```

1  /**
2   * @brief arms the drone when a call to the arm service is made
3   *
4   * @param request_header the header for the service request
5   * @param request the request (empty)
6   * @param response the response (empty)
7   */
8  void handle_arm_request(
9      const std::shared_ptr<rmw_request_id_t> request_header,
10     const std::shared_ptr<drone_services::srv::ArmDrone::Request> request,
11     const std::shared_ptr<drone_services::srv::ArmDrone::Response> response)
12  {
13     RCLCPP_INFO(this->get_logger(), "Got arm request...");
14     if (!armed)
15     {
16         this->publish_vehicle_command(px4_msgs::msg::VehicleCommand::VEHICLE_CMD_DO_SET_MODE, 1, 6);
17         RCLCPP_INFO(this->get_logger(), "Set to offboard mode");
18         // arm the drone
19         this->publish_vehicle_command(px4_msgs::msg::VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM, 1.0, 0);
20         RCLCPP_INFO(this->get_logger(), "Arm command sent");
21         this->last_thrust = -0.1; // spin motors at 10% thrust
22         armed = true;
23         auto msg = drone_services::msg::DroneArmStatus();
24         msg.armed = true;
25         arm_status_publisher->publish(msg);
26         response->success = true;
27     }
28     else
29     {
30         RCLCPP_ERROR(this->get_logger(), "Arm request received but the drone is already armed!");
31         response->success = false;
32     }
33 }

```

Codefragment 5.5: armen van de drone

```

1  /**
2   * @brief disarms the drone when a call to the disarm service is made
3   *
4   * @param request_header the header for the service request
5   * @param request the request (empty)
6   * @param response the response (empty)
7   */
8  void handle_disarm_request(
9      const std::shared_ptr<rmw_request_id_t> request_header,
10     const std::shared_ptr<drone_services::srv::DisarmDrone::Request> request,
11     const std::shared_ptr<drone_services::srv::DisarmDrone::Response> response)
12  {
13     RCLCPP_INFO(this->get_logger(), "Got disarm request...");
14
15     if (armed)
16     {
17         armed = false;
18         user_in_control = false;
19         publish_vehicle_command(px4_msgs::msg::VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM, 0.0, 0);
20
21         auto msg = drone_services::msg::DroneArmStatus();
22         msg.armed = false;
23         arm_status_publisher->publish(msg);
24
25         response->success = true;
26     }
27     else
28     {
29         RCLCPP_ERROR(this->get_logger(), "Got disarm request but drone was not armed!");
30         response->success = false;
31     }
32 }

```

Codefragment 5.6: disarmen van de drone

De PX4Controller is geabonneerd op de `/drone/failsafe` topic die aangeeft wanneer de failsafe ingeschakeld is. Als een bericht op deze topic wordt ontvangen, kunnen er geen setpoints meer afgehandeld worden en maakt de drone een noodlanding. Dit is te zien in Codefragment 5.8. Voor het ontvangen van de failsafe berichten en in Codefragment 5.8 voor het versturen van de setpoints. Bij het versturen van de setpoints wordt ook met poolcoördinaten de nieuwe positie berekend met inachtneming van de huidige yaw van de drone (regel 18 - 27).

```

1  /**
2   * @brief Callback function for receiving failsafe messages
3   *
4   * @param msg the message indicating that the failsafe was enabled
5   */
6  void on_failsafe_received(const drone_services::msg::FailsafeMsg::SharedPtr msg)
7  {
8     if (msg->enabled)
9     {
10         this->failsafe_enabled = true;
11     }

```

Codefragment 5.7: Ontvangen failsafe berichten

```

1  /**
2   * @brief Send setpoints to PX4.
3   *
4   */
5   void send_setpoint()
6   {
7       if (failsafe_enabled)
8       {
9           // immediately land if failsafe enabled
10          RCLCPP_INFO(this->get_logger(), "Failsafe enabled, landing...");
11          this->current_control_mode = CONTROL_MODE_VELOCITY;
12          this->velocity[0] = 0;
13          this->velocity[1] = 0;
14          this->velocity[2] = -0.1; // move down slowly
15          send_velocity_setpoint();
16          return;
17      }
18      // the spiral, in polar coordinates (theta, rho), is given by
19      // theta = theta_0 + omega*t
20      // rho = rho_0 + K*theta
21      float theta = theta_0 + omega * 0.1;
22      float rho = rho_0 + K * theta;
23
24      // from polar to cartesian coordinates
25      des_x = rho * cos(theta);
26      des_y = rho * sin(theta);
27      des_z = position[2]; // the z position can be set to the received height
28
29      if (!user_in_control)
30      {
31          send_attitude_setpoint();
32      }
33      else
34      {
35          if (current_control_mode == CONTROL_MODE_ATTITUDE)
36          {
37              send_attitude_setpoint();
38          }
39          else
40          {
41              if (!new_setpoint)
42              {
43                  return;
44              }
45              if (current_control_mode == CONTROL_MODE_VELOCITY)
46              {
47                  send_velocity_setpoint();
48              }
49              else if (current_control_mode == CONTROL_MODE_POSITION)
50              {
51                  send_position_setpoint();
52              }
53              new_setpoint = false;
54          }
55      }
56  }

```

Codefragment 5.8: versturen van setpoints naar de flight controller

Tussen de commando's vanuit de API en de aansturing van de flight controller zit een node genaamd PositionChanger. De volledige code hiervoor is te zien in Bijlage A.1: PositionChanger node. Deze ontvangt de commando's voor de besturing van de *ApiListener* via de MovePosition service (zie Figuur 5.3) die aangeven of de drone naar voren, achter, links of rechts moet bewegen en vertaalt dit naar x-,y- en z-waarden met inachtneming van de hoek van de drone. Deze node ontvangt ook de data van de LIDAR over de afstand van de drone tot verschillende objecten en zorgt dat de drone van een object af beweegt als deze te dicht in de buurt komt. De methode voor het ontvangen van MovePosition commando's is te zien in Codefragment 5.9. Hierbij wordt eerst gekeken of de yaw (draaiing om de z-as van de drone) correct is (niet meer dan 360° links of rechts) (regel 14 - 19) en wordt daarna de snelheid in de x- en y- richting berekend met de doorgegeven yaw (regel 22). Het aanpassen van de yaw is relatief aan de huidige yaw van de drone. Een bericht met een yaw van +30 betekent dat de drone 30 graden naar rechts zal draaien.

```

1  /**
2   * @brief Callback function for when the /drone/move_position service is called
3   *
4   * @param request_header the header of the request
5   * @param request the request containing the new parameters to move to
6   * @param response the response to send back. true if the request was successful, false otherwise
7   */
8   void handle_move_position_request(
9       const std::shared_ptr<rmw_request_id_t> request_header,
10      const std::shared_ptr<drone_services::srv::MovePosition::Request> request,
11      const std::shared_ptr<drone_services::srv::MovePosition::Response> response)
12  {
13      if (!this->failsafe_enabled)
14      {
15          RCLCPP_INFO(this->get_logger(), "Incoming request\nfront_back: %f\nleft_right: %f\nup_down: %f\nangle: %f", request->front_back, request->left_right, request->up_down, request->angle);
16          if (request->angle > 360 || request->angle < -360)
17          {
18              RCLCPP_ERROR(this->get_logger(), "Angle is not in range [-360, 360]");
19              response->success = false;
20              return;
21          }
22          this->current_yaw += request->angle * (M_PI / 180.0); // get the angle in radians
23          this->current_speed_z = request->up_down;
24          get_x_y_with_rotation(request->front_back, request->left_right, this->current_yaw, &this->current_speed_x, &this->current_speed_y);
25          RCLCPP_INFO(this->get_logger(), "Calculated speed x: %f, y: %f", this->current_speed_x, this->current_speed_y);
26          send_trajectory_message();
27          response->success = true;
28      } else {
29          response->success = false;
30      }
31  }

```

Codefragment 5.9: methode voor ontvangen berichten MovePosition service

Het berekenen van de nieuwe snelheid wordt gedaan met de `get_x_y_with_rotation` functie en deze is te zien in Codefragment 5.10. Dit wordt gedaan omdat de x- en y-waarden van de snelheid bij een *trajectory setpoint* onafhankelijk zijn van de yaw. Hiermee hoeft de gebruiker geen rekening met de huidige yaw van de drone te houden, en kan deze simpelweg berichten voor bijvoorbeeld “naar voren bewegen” sturen.

```

1  /**
2   * @brief Get the new x and y coordinates after a rotation of yaw radians.
3   * Uses the standard rotation matrix: https://en.wikipedia.org/wiki/Rotation\_matrix
4   *
5   * @param x the original x coordinate
6   * @param y the original y coordinate
7   * @param yaw the angle to rotate by in radians
8   * @param x_out the resulting x coordinate
9   * @param y_out the resulting y coordinate
10  */
11  static void get_x_y_with_rotation(float x, float y, float yaw, float *x_res, float *y_res)
12  {
13      *x_res = x * std::cos(yaw) - y * std::sin(yaw);
14      *y_res = x * std::sin(yaw) + y * std::cos(yaw);
15  }

```

Codefragment 5.10: berekenen snelheid met bepaalde hoek

De PositionChanger is geabonneerd op de VehicleOdometry publisher vanuit de flight controller. Deze publiceert data over de huidige houding en positie van de flight controller. De PositionChanger gebruikt deze data om de yaw van de drone te ontvangen als deze nog niet is opgestegen. Dit wordt gedaan omdat het van tevoren niet duidelijk is met welke yaw de drone gedraaid is. Deze yaw wordt dan als startpunt gezien en alle draaiingen zijn relatief aan dit startpunt. Deze functionaliteit is te zien in Codefragment 5.11. Hierbij ontvangt de `handle_odometry_message` functie de data uit de flight controller. Deze haalt de quaternion uit het bericht en berekent de yaw met de `get_yaw_angle` functie. Een quaternion is een manier om 3D vectoren weer te geven. Dit gebeurt niet met een pitch, yaw en roll, maar met drie imaginaire getallen en één reaal getal (Hughes, 2017). In de flight controller gebeurt het aanpassen van de houding via *attitude setpoints* ook via quaternions (Pixhawk, 2023j).

```

1 struct Quaternion
2 {
3     float w, x, y, z;
4 };
5
6 /**
7  * @brief Callback function for receiving the odometry data from the drone.
8  * This is used to get the current yaw angle before the drone takes off,
9  * to make sure it doesn't want to spin to a certain angle while taking off.
10  *
11  * @param message the message containing the odometry data
12  */
13 void handle_odometry_message(const px4_msgs::msg::VehicleOdometry::SharedPtr message)
14 {
15     Quaternion q = {message->q[0], message->q[1], message->q[2], message->q[3]};
16     this->current_yaw = get_yaw_angle(q);
17 }
18
19 /**
20  * @brief Get the yaw angle from a specified normalized quaternion.
21  * Uses the theory from https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles
22  *
23  * @param q the quaternion that holds the rotation
24  * @return the yaw angle in radians
25  */
26 static float get_yaw_angle(Quaternion q)
27 {
28     float siny_cosp = 2 * (q.w * q.z + q.x * q.y);
29     float cosy_cosp = 1 - 2 * (q.y * q.y + q.z * q.z);
30     return std::atan2(siny_cosp, cosy_cosp);
31 }

```

Codefragment 5.11: ontvangen VehicleOdometry berichten uit flight controller en omzetten naar yaw

Zoals te zien in Figuur 5.3, is de PositionChanger ook geabonneerd op de `/drone/object_detection` topic waar de LIDAR de afstandsdata op publiceert. Deze data gebruikt de PositionChanger om ervoor te zorgen dat de drone nergens tegen aan vliegt. De methode om deze data te ontvangen is te zien in Codefragment 5.12. Hierbij worden de waarden van iedere sensor van de LIDAR opgeslagen in een array op de positie van die sensor. Zo weet de PositionChanger aan welke kant van de drone het object zit en welke kant de drone op moet bewegen om het object te vermijden.

```

1 #define LIDAR_SENSOR_FR 0 // front right sensor
2 #define LIDAR_SENSOR_FL 1 // front left sensor
3 #define LIDAR_SENSOR_RL 2 // rear left sensor
4 #define LIDAR_SENSOR_RR 3 // rear right sensor
5
6 /**
7  * @brief Callback function for receiving new LIDAR data
8  *
9  * @param message the message containing the LIDAR data
10  */
11 void handle_lidar_message(const drone_services::msg::LidarReading::SharedPtr message)
12 {
13     if (message->sensor_1 > 0)
14     {
15         this->lidar_sensor_values[LIDAR_SENSOR_FR] = message->sensor_1;
16     }
17     if (message->sensor_2 > 0)
18     {
19         this->lidar_sensor_values[LIDAR_SENSOR_FL] = message->sensor_2;
20     }
21     if (message->sensor_3 > 0)
22     {
23         this->lidar_sensor_values[LIDAR_SENSOR_RL] = message->sensor_3;
24     }
25     if (message->sensor_4 > 0)
26     {
27         this->lidar_sensor_values[LIDAR_SENSOR_RR] = message->sensor_4;
28     }
29     for (int i = 0; i < 4; i++)
30     {
31         this->lidar_imu_readings[i] = message->imu_data[i];
32     }
33     check_move_direction_allowed();
34 }

```

Codefragment 5.12: methode ontvangen LIDAR data

De PositionChanger controleert ook of de LIDAR regelmatig berichten stuurt. Deze heeft een timer die elke seconde checkt of de LIDAR sinds de vorige keer dat er gecheckt werd een nieuw bericht heeft gestuurd. Als dit niet zo is, wordt de failsafe geactiveerd. Dit is te zien in Codefragment 5.13.

```

1 /**
2  * @brief Checks if the LIDAR is still sending messages. If not, enable failsafe
3  */

```

```

4      */
5      void check_lidar_health()
6      {
7          if (!this->received_lidar_message)
8          {
9              RCLCPP_WARN(this->get_logger(), "Lidar not sending messages, enabling failsafe");
10             enable_failsafe("No healthy connection to LIDAR!");
11         }
12         this->received_lidar_message = false;
13     }
14     /**
15     * @brief Enables the failsafe with the specified message
16     *
17     * @param message the message indicating the cause of the failsafe
18     */
19     void enable_failsafe(std::string message)
20     {
21         this->failsafe_enabled = true;
22         this->failsafe_request->message = message;
23         auto failsafe_response = this->failsafe_client->async_send_request(this->failsafe_request,
24             std::bind(&PositionChanger::failsafe_message_callback, this, std::placeholders::_1));
25     }

```

Codefragment 5.13: controleren verbinding LIDAR en aanzetten failsafe

Nadat de data van de LIDAR is ontvangen, wordt berekend of de een object zich te dicht bij de drone bevindt (minder dan 1 meter) en of de drone in een bepaalde richting moet bewegen om van het object af te bewegen. Deze functionaliteit is te zien in Codefragment 5.14. Hierbij wordt eerst gekeken of een object te dicht bij de drone in de buurt is (regel 11 - 14). Hierna wordt berekend hoe snel de drone in een bepaalde richting moet bewegen om van het object weg te vliegen (regel 16 - 24). Als dit berekend is, wordt de verandering van de snelheid toegepast (regel 26).

```

1  #define MOVE_DIRECTION_FRONT 0 // position in array for moving forward
2  #define MOVE_DIRECTION_LEFT 1 // position in array for moving left
3  #define MOVE_DIRECTION_BACK 2 // position in array for moving backward
4  #define MOVE_DIRECTION_RIGHT 3 // position in array for moving right
5  /**
6   * @brief checks for every direction if an object is too close and if we can move in that direction.
7   * If the object is too close to the drone, calculate the amount we need to move away from it
8   */
9  void check_move_direction_allowed()
10 {
11     this->move_direction_allowed[MOVE_DIRECTION_FRONT] = this->lidar_sensor_values[LIDAR_SENSOR_FR] > MIN_DISTANCE && this->lidar_sensor_values[LIDAR_SENSOR_FL] > MIN_DISTANCE;
12     this->move_direction_allowed[MOVE_DIRECTION_LEFT] = this->lidar_sensor_values[LIDAR_SENSOR_FL] > MIN_DISTANCE && this->lidar_sensor_values[LIDAR_SENSOR_RL] > MIN_DISTANCE;
13     this->move_direction_allowed[MOVE_DIRECTION_BACK] = this->lidar_sensor_values[LIDAR_SENSOR_RL] > MIN_DISTANCE && this->lidar_sensor_values[LIDAR_SENSOR_RR] > MIN_DISTANCE;
14     this->move_direction_allowed[MOVE_DIRECTION_RIGHT] = this->lidar_sensor_values[LIDAR_SENSOR_RR] > MIN_DISTANCE && this->lidar_sensor_values[LIDAR_SENSOR_FR] > MIN_DISTANCE;
15
16     // calculate the amount we need to move away from the object to be at the minimum distance
17     collision_prevention_weights[MOVE_DIRECTION_FRONT] = this->move_direction_allowed[MOVE_DIRECTION_FRONT] ? 0 : -(MIN_DISTANCE -
18         std::min(this->lidar_sensor_values[LIDAR_SENSOR_FR], this->lidar_sensor_values[LIDAR_SENSOR_FL]));
19     collision_prevention_weights[MOVE_DIRECTION_LEFT] = this->move_direction_allowed[MOVE_DIRECTION_LEFT] ? 0 : (MIN_DISTANCE -
20         std::min(this->lidar_sensor_values[LIDAR_SENSOR_FL], this->lidar_sensor_values[LIDAR_SENSOR_RL]));
21     collision_prevention_weights[MOVE_DIRECTION_BACK] = this->move_direction_allowed[MOVE_DIRECTION_BACK] ? 0 : (MIN_DISTANCE -
22         std::min(this->lidar_sensor_values[LIDAR_SENSOR_RL], this->lidar_sensor_values[LIDAR_SENSOR_RR]));
23     collision_prevention_weights[MOVE_DIRECTION_RIGHT] = this->move_direction_allowed[MOVE_DIRECTION_RIGHT] ? 0 : -(MIN_DISTANCE -
24         std::min(this->lidar_sensor_values[LIDAR_SENSOR_RR], this->lidar_sensor_values[LIDAR_SENSOR_FR]));
25     apply_collision_weights();
26 }
27

```

Codefragment 5.14: berekenen of de drone van een object weg moet bewegen

Het toepassen van de snelheid om van een object weg te bewegen wordt gedaan in de `apply_collision_weights` methode te zien in Codefragment 5.15. Hierin wordt gekeken of de drone zich te dicht bij een object bevindt en wordt de snelheid aangepast waar nodig. Dit draagt bij aan eis MH4.

```

1  /**
2   * @brief applies the collision prevention weights to the current x and y speed if the drone is too close to an object.
3   * It moves the drone away from the object until it is far enough away
4   */
5  void apply_collision_weights()
6  {
7      if (this->current_speed_x > 0) // if moving forward
8

```

```

9      {
10         if (!this->move_direction_allowed[MOVE_DIRECTION_FRONT])
11         {
12             this->current_speed_x = collision_prevention_weights[MOVE_DIRECTION_FRONT];
13         }
14     }
15     else // moving backward
16     {
17         if (!this->move_direction_allowed[MOVE_DIRECTION_BACK])
18         {
19             this->current_speed_x = collision_prevention_weights[MOVE_DIRECTION_BACK];
20         }
21     }
22     if (this->current_speed_y > 0) // moving right
23     {
24         if (!this->move_direction_allowed[MOVE_DIRECTION_RIGHT])
25         {
26             this->current_speed_y = collision_prevention_weights[MOVE_DIRECTION_RIGHT];
27         }
28     }
29 }
30 else // moving left
31 {
32     if (!this->move_direction_allowed[MOVE_DIRECTION_LEFT])
33     {
34         this->current_speed_y = collision_prevention_weights[MOVE_DIRECTION_LEFT];
35     }
36 }
37 }

```

Codefragment 5.15: toepassen van snelheid om van een object weg te bewegen

Bij het versturen van een bericht naar de PX4Controller wordt ook eerst gekeken of de drone zich niet te dicht bij een object bevindt door de `check_move_direction_allowed` methode aan te roepen voordat de waarden verstuurd worden. Dit is te zien in Codefragment 5.16. De werking van de besturing is te zien in video 4 in Bijlage C: Overzicht video's. In de video worden telkens requests gestuurd (terminal onderin rechts) met andere `front_back`, `left_right`, `up_down` en `angle` parameters. Er is te zien dat de drone bijvoorbeeld met de parameters 1 0 0 20 met 1 m/s naar voren beweegt en met een hoek van 20° naar rechts draait.

```

1  define DEFAULT_CONTROL_MODE 16 // default velocity control bitmask
2  // converts bitmask control mode to control mode used by PX4Controller
3  #define PX4_CONTROLLER_CONTROL_MODE(x) ((x) == 4 ? 1 : ((x) == 16 ? 2 : ((x) == 32 ? 3 : -1)))
4  /**
5   * @brief sends the current x, y and z speed and yaw to the trajectory service
6   *
7   */
8  void send_trajectory_message()
9  {
10     check_move_direction_allowed();
11     this->trajectory_request->values[0] = this->current_speed_x;
12     this->trajectory_request->values[1] = this->current_speed_y;
13     this->trajectory_request->values[2] = this->current_speed_z;
14     this->trajectory_request->yaw = this->current_yaw;
15     this->trajectory_request->control_mode = PX4_CONTROLLER_CONTROL_MODE(DEFAULT_CONTROL_MODE);
16     RCLCPP_INFO(this->get_logger(), "Sending trajectory message\nx: %f\ny: %f\nz: %f\nyaw: %f", this->current_speed_x, this->current_speed_y, this->current_speed_z, this->current_yaw);
17     auto trajectory_response = this->trajectory_client->async_send_request(this->trajectory_request,
18     std::bind(&PositionChanger::trajectory_message_callback, this, std::placeholders::_1));

```

Codefragment 5.16: versturen van setpoint naar PX4Controller

De failsafe node is te zien in Bijlage A.11: FailSafe node. Hierin is te zien dat de node een service aanbiedt voor het aanzetten van de failsafe (regel 12, 13 en 16 - 23) en op de `/drone/failsafe` topic publiceert als deze aangezet wordt (regel 14 en 25 - 30).

Met de huidige implementatie kan de drone alleen handmatig bestuurd worden. Dit wordt gedaan door setpoints voor een richting en snelheid naar de drone te sturen. Dit kan echter vrij eenvoudig omgebouwd worden naar autonome besturing. Hiervoor hoeft alleen een systeem ingebouwd te worden waarmee deze setpoints achter elkaar gestuurd kunnen worden. Dit kan gedaan worden door de RouteFollower node in Figuur 5.3 te implementeren. Dit wordt dan ook aanbevolen om na dit project te doen.

5.3.2 Uitlezen sensoren

Hoogtemeter

Voor het landen van de drone wordt de hoogtemeter aan de onderkant van de drone gebruikt. Dit is een TeraRanger Evo Mini, te zien in Figuur 5.16. Hiervoor is een *height_reader* node gemaakt (zie Bijlage A.4: Hoogtemeter node) die de data van de sensor uitleest en deze op de *drone/height* topic publiceert. De sensor meet de hoogte onder de drone in vier kwadranten. Deze waarden worden samen met de kleinste waarde gepubliceerd om zo de kleinste hoogte onder de drone te verkrijgen.



Figuur 5.16: TeraRanger Evo Mini

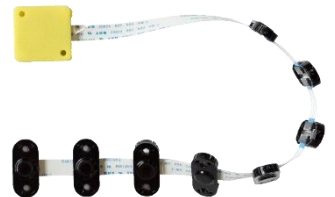
LIDAR en objectdetectiesensoren

Om te zorgen dat de drone nergens tegenaan vliegt, zijn er ook nodes gemaakt om de LIDAR (TeraRanger Tower Evo, te zien in Figuur 5.17) en objectdetectiesensoren (TeraRanger Multiflex, te zien in Figuur 5.18) uit te lezen. De code voor de objectdetectiesensoren is te zien in Bijlage A.5: Multiflex node. Hierin worden de 6 afstandssensoren van de sensor uitgelezen en op de *drone/object_detection* topic gepubliceerd. Helaas is de objectdetectiesensor defect geraakt aan het eind van het project en een nieuwe zou misschien niet voor het eind van het project bezorgd zijn. Hierom is de keuze gemaakt om alleen de LIDAR te gebruiken.



Figuur 5.17: TeraRanger Tower Evo

De code voor de LIDAR is te zien in Bijlage A.6: LIDAR node. Hierbij wordt de data uit de 4 afstandssensoren van de LIDAR verzameld samen met de data uit de ingebouwde IMU van de LIDAR over de houding van de sensor. Deze data wordt dan ook op de *drone/object_detection* topic gepubliceerd. Met de implementatie van deze onderdelen wordt bijgedragen aan eis MH3 en MH4.



Figuur 5.18: TeraRanger Multiflex

De hoogtemeter en LIDAR zijn via USB verbonden aan de flight computer met een eigen poortnaam. Deze poortnaam wordt bij het opstarten van de flight controller toegewezen, en is niet voor elke sensor altijd hetzelfde. Normaal kan dit opgelost worden door de *product id* en *vendor id* van de USB apparaten te vergelijken. De beide sensoren zijn echter gebaseerd op dezelfde STM32 microcontroller waardoor deze waarden hetzelfde zijn (0483:5740). Om dit op te lossen is een Linux script geschreven die de apparaten zoekt op basis van het serienummer van het apparaat. Dit script werkt door de serienummers (00000000001A voor de LIDAR en 369A378F3439 voor de hoogtemeter) van de apparaten op de USB-poorten te vergelijken en de betreffende USB-poort in een bestand te schrijven. Het script is te zien in Codefragment 5.17.

```

1  #!/bin/bash
2  lidarserial=0000000001A
3  heightserial=369A378F3439
4
5  find_by_id(){
6      v=${1%:*}; p=${1#*:} # split vid:pid into 2 vars
7      v=${v%[!0]*}; p=${p%[!0]*} # strip leading zeros
8      grep -il "^PRODUCT=$v/$p" /sys/bus/usb/devices/*:/uevent |
9      sed s,uevent,, |
10     xargs -r grep -r '^DEVNAME=' --include uevent
11 }
12 ids=$(find_by_id 0483:5740)
13 echo $ids
14
15 for device in $ids
16 do
17     id=$(echo $device | cut -d "=" -f 2)
18     serial=$(/bin/udevadm info --name=/dev/$id | grep SERIAL_SHORT | cut -d = -f 2)
19     echo "device \"$id\" has serial number \"$serial"
20     if [ $serial = $lidarserial ]
21     then
22         echo "LIDAR on port" $id
23         echo $id > /home/ubuntu/drone_conf/lidar.conf
24     elif [ $serial = $heightserial ]
25     then
26         echo "height sensor on port" $id
27         echo $id > /home/ubuntu/drone_conf/height.conf
28     else
29         echo "unknown serial number" $serial
30     fi
31 done

```

Codefragment 5.17: script voor USB-poorten hoogtemeter en LIDAR

Het resultaat van dit script wordt dan gebruikt door de ROS 2 nodes van de sensoren door de seriële poort als parameter uit te lezen (regel 18-20 en 30 in Bijlage A.4: Hoogtemeter node en regel 20-23 en 30 in Bijlage A.6: LIDAR node). Deze parameters worden aangepast in de scripts voor het starten van de nodes, te zien in Codefragment 5.18 voor de hoogtemeter en Codefragment 5.19 voor de LIDAR. Hierbij wordt de output van de node weggeschreven naar een logbestand (regel 5 bij beide fragmenten)

```

1  #!/bin/bash
2  . /home/ubuntu/source_ros2.sh # source ros2 installation
3  SERIAL=$(cat /home/ubuntu/drone_conf/height.conf)
4  echo "serial port is "$SERIAL
5  ros2 run height height_reader --ros-args -p height_serial_port:=/dev/${SERIAL} | tee
   /home/ubuntu/drone_log/height.log

```

Codefragment 5.18: opstart script voor hoogtemeter

```

1  #!/bin/bash
2  . /home/ubuntu/source_ros2.sh # source ros2 installation
3  SERIAL=$(cat /home/ubuntu/drone_conf/lidar.conf)
4  echo "serial port is "$SERIAL
5  ros2 run object_detection lidar_reader --ros-args -p lidar_serial_port:=/dev/${SERIAL} | tee
   /home/ubuntu/drone_log/lidar.log

```

Codefragment 5.19: opstart script voor LIDAR

Aansturen camera

De functionaliteit voor het maken van foto's zit in de *CameraController* node. De code hiervoor is te zien in Bijlage A.7: CameraController node. Deze heeft een */drone/picture* service die een optionele bestandsnaam ontvangt voor de te nemen foto. Als deze service aangeroepen wordt, wordt een Linux commando uitgevoerd om een foto met hoge resolutie te maken en op te slaan onder de ingegeven bestandsnaam (regel 26 - 28). Als geen bestandsnaam meegegeven wordt, wordt de huidige datum en tijd als bestandsnaam gezet (regel 20 - 24). Hierna wordt de bestandsnaam van de gemaakte foto teruggestuurd. Het implementeren van de camera draagt bij aan eis *MH6*.

Uitlezen beacons

Voor het uitlezen van de beacons en het bepalen van de positie wordt de *TrackerPositioning* node gebruikt. Hiermee wordt bijgedragen aan eis *MH5*. De code hiervoor is te zien in Bijlage A.8: *TrackerPositioning* node. Deze node wordt twee keer opgestart, één keer voor elke beacon op de drone. Om te bepalen welke beacon het is, wordt een parameter meegegeven die de seriële poort van de beacon aangeeft (regel 28 -36 en 120 - 131). De node wordt vervolgens ingesteld met de API van de beacons (de `setup_rtlsdevice` methode) en als elke node afstandsdata ontvangt wordt deze op de `/drone/object_detection` topic gepubliceerd (regel 133 - 159).

Na het implementeren van de code voor de beacons is getest hoe accuraat deze zijn. Hierbij werd echter duidelijk dat de positie van de middelste beacon erg inaccuraat is. De conclusie is getrokken dat de locatie in de 5G Hub te veel obstructies bevat om de locatie van de drone accuraat te bepalen met maar vier stationaire beacons.

Aansturing relais

Het aansturen van het relais gebeurt ook in een ROS 2 node. Deze is te zien in Bijlage A.9: *RelaisController* node. Hierin worden de GPIO pins ingesteld in de `init_gpio` methode (regel 22 - 31). Hierna wordt het relais geschakeld door de GPIO pinnen hoog te zetten (regel 33 - 36). Deze node biedt ook een service aan waarmee het relais geschakeld kan worden, te zien in de `control_relais_callback` methode (regel 38 - 51).

Test controller

Om de aansturing van de drone grondig te kunnen testen is een test controller node geschreven. Deze is te vinden op de [Github repository](#) van dit project. Deze biedt functionaliteit om de drone handmatig via het toetsenbord aan te sturen, mits deze wordt uitgevoerd vanaf een SSH verbinding met de flight computer. De node wacht eerst totdat de services voor de *Heartbeat* en *PX4Controller* nodes beschikbaar zijn en begint dan met de toetsen af te vangen en de commando's naar de nodes te sturen.

Status van de drone

Om de gebruiker een duidelijk overzicht te geven van de status van de drone, is de *DroneStatus* node geschreven. Deze is te zien in Bijlage A.10: *DroneStatus* node. Deze node verzamelt gegevens over of de drone gearmd is, welke mate van besturing momenteel actief is, over de batterij en processor van de flight controller en of de drone momenteel een route volgt en wat de status hiervan is. De informatie voor de processor en batterij vanaf de flight controller is niet standaard beschikbaar vanaf ROS 2. Dit kan gedaan worden door de firmware van de flight controller aan te passen en deze informatie beschikbaar te maken voor de XRCE-DDS middleware (Pixhawk, 2023i). De firmware is aangepast en functionaliteit voor het uitlezen van het percentage van de batterij en het gebruik van de processor is toegevoegd. Deze nieuwe functionaliteit wordt gebruikt in de *DroneStatus* node op regel 23 en 59 – 69 voor de batterij en regel 24 en 62 – 63 voor de processor. Door het implementeren van deze node wordt bijgedragen aan eis *SH2*.

Bij het implementeren van alle sensoren is tijdens dit project geen tijd over geweest voor het monteren en aansluiten van het scherm aan de onderkant van de drone. Dit komt omdat er veel tijd besteed is aan de verbinding van de flight controller en flight computer en het configureren van de flight computer. Het wordt dus aangeraden om dit na dit project op te pakken. Omdat de flight

computer geen gebruikersinterface bevat en om te zorgen dat deze niet toegevoegd hoeft te worden zodat de flight computer niet onnodig veel resources gebruikt, wordt aangeraden onderdelen op het scherm weer te geven door de framebuffer van de flight computer aan te passen. Zo kunnen onderdelen op het scherm weergegeven worden zonder dat een gebruikersinterface op de flight computer gezet hoeft te worden. Dit bespaart resources.

De flight computer en modem zitten op de drone gemonteerd met een tie-wrap. Dit is niet wenselijk omdat dit geen professionele uitstraling geeft. Tijdens het project was er onvoldoende tijd en kennis om hier een behuizing voor te 3D-printen. Het wordt aangeraden om dit na dit project te doen.

De camera zit op de drone gemonteerd met duct tape. Dit is niet wenselijk omdat dit geen stevige bevestiging creëert. Deze keuze is gemaakt omdat er uiteindelijk onvoldoende kennis en tijd was om hier een bevestiging voor te ontwerpen en te 3D-printen. Het wordt ook aangeraden om dit na dit project te doen.

5.4 Aansturing via 5G

De drone verbindt via 5G met de edge computer. De 5G modem die hiervoor zorgt wordt aangesloten op de USB C poort van de flight computer. Om te valideren of deze werkt is hier een SIM kaart met 5G in gezet. Deze SIM heeft extra netwerk mogelijkheden gerelateerd aan 5G. Eén hiervan is *local break-out*. Dit houdt in dat het netwerkverkeer van de SIM niet naar het publieke internet gaat, maar naar het netwerk van de 5G Hub. Dit betekent dat de drone niet buiten het netwerk van de 5G Hub bediend kan worden en hiermee wordt voldaan aan eis *MH8*. Een andere feature van deze SIM is dat deze gebruik maakt van netwerk slicing. Dit houdt in dat deze altijd recht heeft op 20% van het netwerk, zodat het camerabeeld van de drone altijd beschikbaar blijft, ongeacht hoeveel verkeer zich op het netwerk bevindt.

Om de modem uit te kunnen lezen via de USB C modem moesten twee regels toegevoegd worden aan `/boot/firmware/config.txt: dtoverlay=dwc2,dr_mode=host` om de modem als netwerkinterface via de USB C poort te kunnen gebruiken en `dtoverlay=disable-bt` om de flight controller aan te kunnen sturen via de UART GPIO pins. Het instellen van de flight computer om de 5G modem te gebruiken draagt bij aan eis *MH7* en *MH8*.

Bij het aansturen via 5G worden ten opzichte van 4G standaard de hogere capaciteit van het netwerk gebruikt. Dit is vooral gunstig voor het streamen van video en het ontvangen van foto's van de drone. Dit is ook gunstig in het geval dat de drone buiten het gezichtsveld van de gebruiker vliegt. Hierbij is het van belang dat de videostream gezond blijft in het geval van een noodlanding, zodat de gebruiker ten alle tijden een overzicht houdt van wat er bij de drone gebeurt.

Hiernaast is het eerder genoemde netwerk slicing belangrijk voor het aansturen van drones via 5G. Doordat de drone recht heeft op een bepaald percentage van het netwerk, heeft de drone altijd gegarandeerde connectiviteit, ongeacht van hoe vol het netwerk zit.

Bij het aansturen van drones via 5G in het algemeen zijn verder de principes *beamforming* en *handovers* van belang. Beamforming is een techniek van signaalbewerking waarbij meerdere kleine antennes hetzelfde signaal met een klein tijdsverschil uitzenden. Door dit op een bepaalde manier te doen kan een draadloos signaal versterkt worden (Shaw, 2022).

In 5G-antennes wordt dit gebruikt om een signaal directer op een bepaald apparaat te kunnen richten (Verizon, 2022). Dit is vooral nuttig bij het gebruik van drones, omdat deze veel ruis kunnen

ontvangen als ze boven de 5G-antennes vliegen. Dit komt omdat de meeste 5G-antennes momenteel met een kleine hoek naar beneden uitzenden, omdat daar het signaal nodig is. Dit zorgt er echter voor dat een drone die boven dit gebied vliegt minder goede dekking krijgt dan wanneer deze dichterbij de grond zou vliegen. Hierbij kan beamforming een oplossing zijn door het signaal meer te richten naar de drone.

5G-antennes hebben een beperkte dekking. Om ervoor te zorgen dat apparaten verbonden blijven terwijl ze bewegen worden handovers gebruikt. Handovers zorgen ervoor dat een apparaat altijd verbonden is met de antenne die de beste dekking voor dat gebied heeft (Guowang Miao, 2016). Dit is voor het vliegen met drones ook erg belangrijk, omdat op deze manier de drone altijd de beste verbinding met een antenne heeft. Hierdoor kan er geen sprake zijn van vermindering van de verbindingsskwaliteit doordat de drone geen goede verbinding met een antenne heeft.

5.5 Testen

Naast het ontwerpen en implementeren van de drone is het ook belangrijk om de gemaakte onderdelen te testen om er zeker van te zijn dat deze werken. Hierom zijn er een aantal testspecificaties opgesteld en uitgevoerd. Deze zijn opgedeeld in unittests voor individuele onderdelen en integratietests om de werking tussen verschillende onderdelen te testen. De namen van de testen zijn opgesteld met het format <package>.<soort test>.<testnummer>. Hierbij is het <package> gedeelte een afkorting van de package waarin de test wordt uitgevoerd. Het <soort test> gedeelte bevat een U voor een unittest, een I voor een integratietest en een F voor een functionele test. Het <testnummer> gedeelte geeft het nummer van die test in die package aan. Een testnaam van bijvoorbeeld PX4.U.2 betekent de tweede test in de px4_connection package, die een unittest voorstelt.

Om er zeker van te zijn dat de gemaakte onderdelen voor de drone goed werken zijn verschillende testen uitgevoerd. Aparte nodes worden getest doormiddel van unit tests. Voor de koppeling tussen de onderdelen zijn integratietests uitgevoerd in de vorm van ROS tests. Deze testen hebben als doel aan te tonen dat verschillende nodes op de juiste manier met elkaar samen kunnen werken. Verder worden ook systeemtests uitgevoerd waarbij een aantal stappen worden genomen en wordt waargenomen hoe het systeem hierop reageert.

5.5.1 Unit tests

Voor het testen van de camera is een unit test CAM.U.1 uitgevoerd. Het doel van deze test is om aan te tonen dat de functionaliteit van een foto nemen door de `/drone/picture` service aan te roepen werkt. De specificatie voor deze test is te zien in Tabel 5.2. Deze test draagt bij aan eis *MH6* en *SH6*.

Tabel 5.2: Testspecificatie unit test camera

Testnaam	CAM.U.1
Beschrijving	Deze test wordt uitgevoerd om aan te tonen dat het nemen van foto's via de CameraController node werkt.
Requirements	MH6 en SH6.
Te testen onderdelen	- CameraController node
Benodigde nodes	- CameraController
Voorwaarden	De batterij is aangesloten en de drone staat aan.
Testprocedure	1. Roep de <code>/drone/picture</code> service van de CameraController aan met een output bestandsnaam van <code>/home/ubuntu/test_image.jpg</code>. 2. Wacht tot de service uitgevoerd is 3. Controleer of de bestandsnaam in het resultaat <code>/home/ubuntu/test_image.jpg</code> is. 4. Controleer of het bestand aangemaakt is.
Verwacht resultaat	Nadat de service aangeroepen is, wordt het resultaat teruggegeven met de zelfde bestandsnaam als bij het aanroepen, namelijk <code>/home/ubuntu/test_image.jpg</code> . Dit bestand bestaat en kan bekeken worden.
Werkelijk resultaat	De service geeft de bestandsnaam <code>/home/ubuntu/test_image.jpg</code> terug. Dit bestand is aangemaakt en kan bekeken worden
Geslaagd?	Ja
Opmerkingen	De code en het resultaat van deze test is te zien in Bijlage B.1: Camera unittest.

Een belangrijk onderdeel van de drone is de failsafe. Deze moet goed werken om zeker te zijn dat de drone geen gevaarlijke situaties veroorzaakt. Een test om aan te tonen dat het aanzetten van de failsafe met een bepaalde reden werkt is beschreven in test FAIL.U.1. De specificatie voor deze test is te zien in Tabel 5.3.

Tabel 5.3: Testspecificatie unit test failsafe

Testnaam	FAIL.U.1
Beschrijving	Deze test wordt uitgevoerd om te controleren of het aanzetten van de failsafe door de bijbehorende service aan te roepen werkt.
Te testen onderdelen	- Failsafe node
Benodigde nodes	- Failsafe
Voorwaarden	De batterij is aangesloten en de drone staat aan.
Testprocedure	1. Zet de failsafe aan door de <code>/drone/enable_failsafe</code> service aan te roepen met als reden "test". 2. Wacht tot de failsafe status wordt gepubliceerd op de <code>/drone/failsafe</code> topic. 3. Controleer of het gepubliceerde bericht een failsafe waarde van true bevat. 4. Controleer of het gepubliceerde bericht als reden de waarde "test" bevat.
Verwacht resultaat	Als de failsafe service aangeroepen wordt, wordt daarna een bericht op de <code>/drone/failsafe</code> topic gepubliceerd over dat de failsafe aan staat. Hierbij is de reden van de failsafe "test".
Werkelijk resultaat	Na het aanroepen van de failsafe wordt het bericht op de failsafe topic ontvangen. Dit bericht bevat de juiste failsafe waarde en de juiste reden van "test".
Geslaagd?	Ja
Opmerkingen	De code voor deze test is te zien in Bijlage B.2: Failsafe unittest. Hierin is in Figuur B.2 het resultaat van de test te zien.

Voor de Heartbeat node is een test opgesteld om te controleren of het veranderen van de mate van besturing werkt door het aanroepen van de `set_vehicle_control` service. Dit is opgesteld in test PX4.U.1 en deze is te zien in Tabel 5.4. Deze test draagt bij aan eis MH1.

Tabel 5.4 Testspecificatie unit test voor het veranderen van de mate van besturing

Testnaam	PX4.U.1
Beschrijving	Deze test wordt uitgevoerd om te controleren of het veranderen van de mate van besturing werkt via de <code>/drone/set_vehicle_control</code> service van de Heartbeat node. Bij deze test wordt gekeken of een juiste waarde en een onjuiste waarde goed worden afgehandeld.
Requirements	MH1
Te testen onderdelen	- Heartbeat node
Benodigde nodes	- Heartbeat
Voorwaarden	De batterij is aangesloten en de drone staat aan.
Testprocedure	1. Maak een request naar de <code>/drone/set_vehicle_control</code> service met een mate van besturing op houding. 2. Wacht tot de service aangeropen is 3. Controleer of het resultaat van de service positief is 4. Maak een request naar de <code>/drone/set_vehicle_control</code> service met een onjuiste mate van besturing met waarde 33. 5. Wacht tot de service aangeropen is 6. Controleer of het resultaat van de service negatief is
Verwacht resultaat	Na het aanroepen van de service met een juiste waarde moet een positief antwoord ontvangen worden met een waarde van <code>success = True</code> . Na het aanroepen van de service met een onjuiste waarde moet een negatief antwoord ontvangen worden met een waarde van <code>success = False</code> .
Werkelijk resultaat	Na het aanroepen van de service met een juiste waarde wordt een antwoord ontvangen met de waarde <code>success = True</code> . Na het aanroepen van de service met een onjuiste waarde wordt een antwoord ontvangen met de waarde <code>success = False</code> .
Geslaagd?	Ja
Opmerkingen	De code voor deze test is te zien in Bijlage B.3: Heartbeat control mode unittest. Hierin is in Figuur B.3 de output van deze test te zien.

5.5.2 Integratietests

Zoals al eerder in Paragraaf 5.5.1 genoemd, is het testen van de failsafe erg belangrijk. Er moet ook gecontroleerd worden of de andere nodes de failsafe goed afhandelen. Om te controleren of de integratie tussen de failsafe en de PX4Controller goed werkt, is een integratietest opgesteld die controleert of de PX4Controller het aansturen van de drone weigert als de failsafe aangezet wordt. Dit is test PX4.I.1 en de specificatie hiervan is te zien in Tabel 5.5. Deze test draagt bij aan eis *MH1*.

Tabel 5.5: Integratietest specificatie voor de integratie tussen de failsafe en de PX4Controller

Testnaam	PX4.I.1
Beschrijving	Deze test wordt uitgevoerd om te controleren of de PX4Controller requests om te armen, de drone te besturen op basis van houding en de drone te besturen op basis van snelheid weigert als de failsafe aan staat.
Requirements	MH1
Te testen onderdelen	- PX4Controller node
Benodigde nodes	- PX4Controller node - Failsafe node
Voorwaarden	De batterij is aangesloten en de drone staat aan. De failsafe is niet geactiveerd.
Testprocedure	1. Activeer de failsafe door de <code>/drone/enable_failsafe</code> service aan te roepen. 2. Maak een request om de drone te armen door de <code>/drone/arm</code> service aan te roepen. 3. Controleer het resultaat van de serviceaanroep. 4. Maak een request om de drone te besturen op basis van houding door de <code>/drone/set_attitude</code> service aan te roepen. 5. Controleer het resultaat van de serviceaanroep. 6. Maak een request om de drone te besturen op basis van snelheid door de <code>/drone/set_trajectory</code> service aan te roepen. 7. Controleer het resultaat van de serviceaanroep.
Verwacht resultaat	Na het aanroepen van elke service bevat het resultaat de waarde <code>success = False</code> .
Werkelijk resultaat	Als de services aangeroepen worden, bevat het resultaat van elke serviceaanroep de waarde <code>success = False</code> .
Geslaagd?	Ja
Opmerkingen	De code voor deze test is te zien in Bijlage B.4: Integratietest failsafe en PX4Controller. Hierin is in Figuur B.4 de output van de test te zien.

Naast de PX4Controller moet ook gecontroleerd worden of de PositionChanger de failsafe aanzet als dit nodig is. Dit is voor de PositionChanger belangrijk als de communicatie met de LIDAR wegvalt en als de LIDAR helemaal geen berichten stuurt. Voor het wegvallen van de communicatie met de LIDAR is test POS.I.1 opgesteld. Dit is een integratietest om te controleren of de PositionChanger de failsafe aanzet als de communicatie met de LIDAR wegvalt. De specificatie voor deze test is te zien in Tabel 5.6. Deze test draagt bij aan eis MH3.

Tabel 5.6: Testspecificatie integratietest wegvallen communicatie LIDAR

Testnaam	POS.I.1
Beschrijving	Deze test wordt uitgevoerd om te controleren of de PositionChanger de failsafe aanzet als de communicatie met de LIDAR wegvalt. Hierna wordt gecontroleerd of de PositionChanger commando's weigert als de failsafe is ingeschakeld.
Requirements	MH3
Te testen onderdelen	- PositionChanger node
Benodigde nodes	- PositionChanger node - Failsafe node - PX4Controller node - Heartbeat node
Voorwaarden	De batterij is aangesloten en de drone staat aan. De failsafe is nog niet geactiveerd.
Testprocedure	1. Start de PositionChanger node op zonder dat de LIDAR node data stuurt. 2. Maak een request om de drone te besturen via de <code>/drone/move_position</code> service. 3. Controleer of het resultaat van de serviceaanroep. 4. Controleer of de failsafe is geactiveerd.
Verwacht resultaat	Nadat de request naar de <code>/drone/move_position</code> service is gemaakt, moet het resultaat van deze service de waarde <code>success = False</code> bevatten. Ook moet de failsafe geactiveerd zijn.
Werkelijk resultaat	Nadat het request naar de service is gemaakt, bevat het resultaat hiervan de waarde <code>success = False</code> . De failsafe is ook geactiveerd.
Geslaagd?	Ja
Opmerkingen	De code voor deze test is te zien in Bijlage B.5: Integratietest PositionChanger, LIDAR en Failsafe. Hierin is de code voor deze test specifiek regel 69 – 102. De output van deze test is in de bijlage te zien in Figuur B.5.

Voor het wegvallen van de communicatie met de LIDAR, dus als de LIDAR eerst wel berichten stuurt maar daarna stopt, is integratietest POS.I.2 opgesteld. Deze test controleert of de PositionChanger de failsafe aanzet als de LIDAR geen berichten meer zou sturen tijdens het vliegen. De specificatie voor deze test is te zien in Tabel 5.7. Met het slagen van deze test en test POS.I.3 wordt voldaan aan eis MH3.

Tabel 5.7: Integratietest failsafe activeren als LIDAR wegvalt

Testnaam	POS.I.2
Beschrijving	Deze test wordt uitgevoerd om te controleren of de PositionChanger de failsafe activeert als de LIDAR stopt met berichten sturen. Hierbij wordt ook gecontroleerd of de PositionChanger weigert commando's te ontvangen als de failsafe geactiveerd is.
Requirements	MH3
Te testen onderdelen	- PositionChanger node
Benodigde nodes	- PositionChanger node - Failsafe node - PX4Controller node - Heartbeat node
Voorwaarden	De batterij is aangesloten en de drone staat aan. De failsafe is nog niet geactiveerd.
Testprocedure	1. Publiceer één testbericht op de <code>/drone/object_detection</code> topic die valide waarden voor de LIDAR bevat. 2. Nadat dit bericht is gestuurd en er na 5 seconden geen LIDAR berichten meer zijn gestuurd, maak een request om de drone te besturen via de <code>/drone/move_position</code> service. 3. Controleer of de failsafe is aangezet. 4. Controleer het resultaat van de serviceaanroep.
Verwacht resultaat	Nadat er na 5 seconden geen berichten meer zijn gestuurd door de LIDAR, is de failsafe aangezet. Als een request wordt gemaakt naar de <code>/drone/move_position</code> service, bevat deze de waarde <code>success = False</code> om aan te geven dat de PositionChanger geen commando's meer ontvangt.
Werkelijk resultaat	Als de <code>/drone/move_position</code> service is aangeroepen bevat het resultaat hiervan de waarde <code>success = False</code> . De failsafe is geactiveerd.
Geslaagd?	Ja
Opmerkingen	De code voor deze test is te zien in Bijlage B.5: Integratietest PositionChanger, LIDAR en Failsafe vanaf regel 104. Het resultaat van de test is in deze bijlage te zien in Figuur B.5.

Als laatst is een integratietest geschreven om te controleren of de PositionChanger waarden van de LIDAR goed ontvangt en op basis hiervan de drone laat bewegen zodat het nergens tegenaan vliegt. De test hiervoor is test POS.I.3 en de specificatie hiervan is te zien in Tabel 5.8. Met het slagen van deze test wordt voldaan aan eis *MH3*. Met het slagen van deze test en test POS.I.2 wordt voldaan aan eis *MH3*.

Tabel 5.8: Integratietest voor objectdetectie

Testnaam	POS.I.3
Beschrijving	Deze test wordt uitgevoerd om te controleren of de PositionChanger de waarden die de LIDAR stuurt goed ontvangt en de drone van een object weg laat bewegen als deze te dicht in de buurt komt.
Requirements	MH3 en MH4
Te testen onderdelen	- PositionChanger node
Benodigde nodes	- PositionChanger node - Failsafe node - PX4Controller node - Heartbeat node
Voorwaarden	De batterij is aangesloten en de drone staat aan.
Testprocedure	1. Laat de drone stil hangen 2. Publiceer valide LIDAR berichten op de <code>/drone/object_detection</code> topic waarbij een object te dicht bij de drone in de buurt komt. 3. Controleer in de output van de PositionChanger of deze de drone aanstuurt om weg te bewegen. 4. Roep de <code>/drone/move_position</code> service aan om de drone naar voren te laten bewegen 5. Publiceer valide LIDAR berichten op de <code>/drone/object_detection</code> topic waarbij een object te dicht bij de voorkant van de drone in de buurt komt. 6. Controleer in de output van de PositionChanger of deze de drone aanstuurt om naar achteren te bewegen. 7. Roep de <code>/drone/move_position</code> service aan om de drone naar achteren te laten bewegen 8. Publiceer valide LIDAR berichten op de <code>/drone/object_detection</code> topic waarbij een object te dicht bij de achterkant van de drone in de buurt komt. 9. Controleer in de output van de PositionChanger of deze de drone aanstuurt om naar voren te bewegen. 10. Roep de <code>/drone/move_position</code> service aan om de drone naar links te laten bewegen 11. Publiceer valide LIDAR berichten op de <code>/drone/object_detection</code> topic waarbij een object te dicht bij de linker kant van de drone in de buurt komt.

	12. Controleer in de output van de PositionChanger of deze de drone aanstuurt om naar rechts te bewegen. 13. Roep de <code>/drone/move_position</code> service aan om de drone naar rechts te laten bewegen 14. Publiceer valide LIDAR berichten op de <code>/drone/object_detection</code> topic waarbij een object te dicht bij de rechter kant van de drone in de buurt komt. 15. Controleer in de output van de PositionChanger of deze de drone aanstuurt om naar links te bewegen.
Verwacht resultaat	Als de drone niet beweegt en LIDAR berichten ontvangt dat iets te dicht in de buurt komt, herkent de PositionChanger dit en is in de output te zien dat deze de drone aanstuurt om weg te bewegen. Voor elke richting van een object is te zien dat de PositionChanger de drone de andere kant op stuurt met een snelheid die gelijk is aan hoe dicht een object zich bij de drone bevindt.
Werkelijk resultaat	De output van de PositionChanger bevat berichten waaruit blijkt dat deze de drone de andere kant op beweegt dan waar zich een object bevindt. De snelheid waarmee de drone bewogen wordt is gelijk aan hoe dicht een object bij de drone in de buurt is.
Geslaagd?	Ja
Opmerkingen	De code voor deze test is te zien in Bijlage B.6: Integratietest PositionChanger en objectdetectie. Hierin is in Figuur B.6 de output van de test zichtbaar.

5.6 Conclusie aansturing drone

Op basis van de informatie uit Hoofdstuk 3 en 4 is de keuze gemaakt om ROS 2 te gebruiken voor het uitlezen van de sensoren en het samenvoegen van de sensordata. Deze keuze is gemaakt omdat ROS 2 veel verbeteringen bevat ten opzichte van ROS 1. Als versie is ROS 2 Foxy gekozen naar aanleiding van advies van PX4 ontwikkelaars. Hierdoor is Ubuntu 20 Server als OS voor de flight computer gekozen.

Omdat gebruik gemaakt wordt van ROS 2, is PX4 als autopilot software gekozen. Hierbij is ook gekozen voor de nieuwste beta-versie (v1.14) in combinatie met XRCE-DDS omdat dit het beste resultaat opleverde. Hiermee is voldaan aan eis MH1. De firmware van de flight controller is aangepast om de status van de batterij en belasting van de processor te kunnen ontvangen.

Figuur 5.1 geeft een schematisch ontwerp weer van de onderdelen van de drone. Hoe de drone eruit ziet en welke onderdelen waar zitten is te zien in Figuur 5.2. Figuur 5.3 geeft een ontwerp weer van alle nodes op de drone en hoe deze met elkaar communiceren.

Bij het configureren van de flight computer hebben zich veel problemen voorgedaan bij het bouwen van de software om met de flight controller te communiceren. Uiteindelijk werd duidelijk dat hier een flight computer met meer RAM (8GB i.p.v. 2GB) nodig was.

Bij het configureren van de flight controller werd bekend dat de vooraf bestelde motoren en propellers niet voldoende waren om de drone te laten vliegen. Hierom zijn nieuwe motoren,

propellers en motorsteunen besteld. Hiermee is voldaan aan eis *MH2*. Tijdens het project is er te weinig kennis en tijd geweest om beschermkappen voor de propellers, een behuizing voor de flight computer en modem en de camera te 3D-printen. Het wordt aangeraden dit na dit project te doen.

Om alle functies van de flight controller te gebruiken en de drone aan te sturen op basis van een gewenste snelheid heeft de flight controller een schatting van de positie nodig. Er is geprobeerd deze schatting binnen te krijgen door gebruik van een GPS. Dit bleek binnen onvoldoende te werken. Vervolgens is een optical flow sensor besteld, die op basis van een naar de grond gerichte camera en een afstandssensor de positie van de drone binnen kan meten. Echter zit er een bug in de beta-versie van de PX4 firmware waardoor het gebruik van optical flow niet werkt. Het wordt aangeraden om na dit project op de hoogte te blijven van de ontwikkeling van PX4 en te wachten tot een update is uitgekomen waarbij de optical flow sensor werkt. Hiermee kan niet worden voldaan aan eis *MH5* en eis *MH9*.

Als laatste poging om de drone binnen te laten vliegen is een beacon positioning systeem besteld genaamd Marvelmind Indoor Navigation System. Dit systeem werkt hetzelfde als de vooraf bestelde beacons, maar deze kunnen direct aangesloten worden op de flight controller. Deze bleken uiteindelijk ook niet voldoende te werken en omwille van de tijd is uiteindelijk de keuze gemaakt om de drone alleen buiten met GPS te laten vliegen. Hiermee kunnen de 5G antennes die buiten hangen wel geïnspecteerd worden. Het scherm voor op de drone is niet geïmplementeerd en het wordt aangeraden dit na dit project op te pakken.

Bij het verbinden van de flight controller en flight computer lukte het eerst niet om de motoren in te schakelen. Op advies van een PX4 ontwikkelaar zijn een aantal parameters van de flight controller aangepast om dit werkend te krijgen. De code voor alle nodes is te zien in Bijlage A: ROS 2 nodes. Omdat de drone nu alleen handmatig bestuurd kan worden door het versturen van setpoints voor de richting en snelheid, wordt aangeraden dit autonoom te maken door deze achter elkaar te sturen. Bij het aansturen via 5G wordt de hogere capaciteit dan bij 4G gebruikt. Hiernaast is netwerk slicing erg belangrijk omdat zo gegarandeerd kan worden dat een deel van het netwerk beschikbaar blijft, hoe vol het netwerk ook zit. Ook wordt gebruik gemaakt van beamforming en handovers. Hiermee is voldaan aan eis *MH8*.

Om te controleren of de nodes van de drone goed werken zijn verschillende unit tests en integratietests uitgevoerd. Deze zijn te zien in Bijlage B: Code en output tests. Het is aangetoond dat voldaan is aan eis *MH3*.

Naast de drone zelf is ook gewerkt aan een API op de drone en een GUI op de edge computer om de drone aan te kunnen sturen. Het ontwerp, de realisatie en het testen hiervan wordt weergegeven in het volgende hoofdstuk, Hoofdstuk 6.

6 API

Dit hoofdstuk beschrijft het onderzoek, ontwerp, de implementatie en het testen van de API waarmee de drone aangestuurd kan worden. Het onderzoek naar de mogelijkheden om de API te ontwikkelen wordt beschreven samen met het ontwerp dat uit dit onderzoek is gekomen. Hierna wordt de implementatie beschreven en wordt uitgelegd hoe de API getest is en wat de resultaten hiervan zijn.

Het doel van de API is om een manier te bieden om de drone via 5G vanaf een edge computer te kunnen besturen. Een edge computer is een aparte computer die dicht bij het 5G netwerk staat en zo via 5G met de drone kan communiceren. Hierbij is het belangrijk dat het videobeeld van de camera op de drone te zien is en dat de drone handmatig bestuurd kan worden. De API wordt op de flight computer uitgevoerd en gedraagt zich als server. De edge computer gedraagt zich als client. Door het gebruik van de API hoeft de implementatie van de aansturing van de flight controller niet bekend te zijn voor de edge computer en hoeft deze alleen rekening te houden met welke acties de drone uit moet voeren. Dit zorgt ook voor een gestandaardiseerde communicatie tussen de flight- en edge computer (Johl, 2022).

6.1 Communicatieprotocol

Omdat gebruik gemaakt wordt van 5G voor een draadloze verbinding tussen de flight- en edge computer, moet gebruik gemaakt worden van een netwerkcommunicatieprotocol (Hoeven, Intelligentie voor een 5G aangestuurde drone, 2023a). HTTPS en TCP/IP zijn de meest gebruikte protocollen om deze communicatie te waarborgen (Jithin, 2016). De manieren waarop deze gebruikt kunnen worden is hieronder gesommeerd (Torver, 2022):

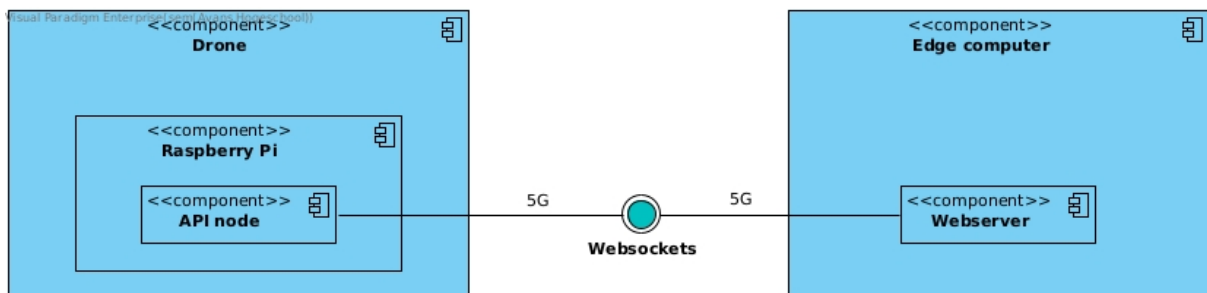
- **Polling:** Hierbij stuurt de client via HTTP regelmatig een bericht naar de server om te vragen of bepaalde data beschikbaar is. De server stuurt dan terug of deze data al beschikbaar is of niet (Zola, 2023).
- **Long polling:** Hierbij stuurt de client via HTTP een request naar de server, en de server antwoordt pas als de data hiervoor beschikbaar is.
- **Websockets:** Dit is een communicatieprotocol die bidirectionele communicatie over één TCP/IP verbinding mogelijk maakt. Dit zorgt voor een tweewegverbinding tussen de server en client waarbij ze allebei berichten naar elkaar kunnen sturen (Ogundeyi & Yinka-Banjo, 2019).
- **Server-Sent Events (SSE):** : Dit is een HTTP-standaard die het mogelijk maakt voor een client om automatisch updates van een server te ontvangen nadat er een verbinding is gemaakt. Deze verbinding is unidirectioneel en de server kan alleen updates pushen naar de client (Ayrancıoğlu, 2022).

Op basis van de hierboven genoemde opties is gekozen om de API te implementeren met websockets. Deze keuze is gemaakt omdat er een constante verbinding tussen de drone en edge computer nodig is. Deze is nodig om zo de drone veilig handmatig te kunnen besturen. Ook moeten berichten tussen de drone en edge computer heen en weer gestuurd worden, dus SSE is geen optie. Omdat bij polling vaak opnieuw een bericht verstuurd moet worden, zorgt dit voor onnodige overhead in het netwerk. Bij long polling: hoeft dit niet, maar moet worden gewacht op een

antwoord van de drone. Dit is ook onwenselijk omdat de edge computer gelijk berichten moet ontvangen van de drone zodra deze beschikbaar zijn.

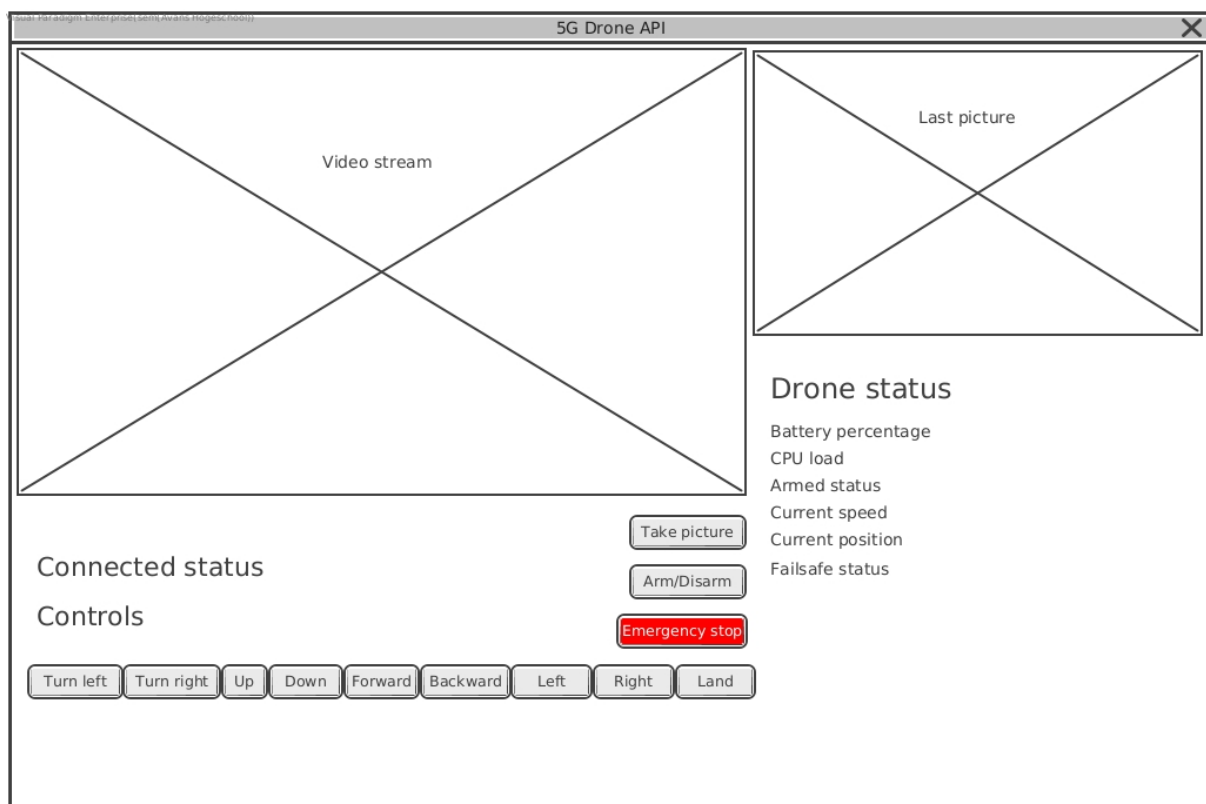
6.2 Ontwerp

Het globale ontwerp van de communicatie tussen de drone en de edge computer is te zien in Figuur 6.1. Hierin is te zien dat de flight computer een API node bevat die dient als API voor de edge computer. Deze communiceert dan via websockets via het 5G-netwerk met de edge computer. Op de edge computer draait een webserver met een gebruikersinterface (GUI). Hiervoor is gekozen omdat zo vanaf overal in de 5G Hub de drone bestuurd kan worden door de webpagina van de edge computer te bezoeken. Hierbij is alleen een verbinding met het wifi netwerk van de 5G Hub vereist.



Figuur 6.1: Overzicht verbinding edge computer en drone

Het ontwerp voor de GUI die op de edge computer draait is te zien in Figuur 6.2. Hierin is te zien dat de video van de drone bovenin links weergegeven wordt met daarnaast de laatst genomen foto. Onder de laatst genomen foto staat de status van de drone.



Figuur 6.2: ontwerp GUI

Onder de video staat de status van de verbinding met de drone en bevinden zich de knoppen om de drone te besturen, te armen of disarmen en om een foto te maken.

Voor de edge computer is een VM (Virtual Machine) gemaakt die op de server van de 5G Hub draait, zodat elk apparaat die met het WIFI-netwerk van de 5G Hub verbonden is deze kan benaderen. Bij het aanmaken van de VM is gekozen voor een configuratie met 8 cores en 16 GB RAM, zodat de VM geen bottleneck kan zijn voor de performance van het ontvangen van de video met hoge resolutie (Richards, 2021).

De API en edge computer communiceren met elkaar via JSON berichten. Hierbij stuurt de edge computer *RequestCommand* berichten die een commando ID bevatten wat aangeeft wat voor actie de API moet ondernemen. De API stuurt *ResponseMessage* berichten terug die een ID en bepaalde data bevatten. De beschikbare RequestCommands die de edge computer kan sturen, wat deze betekenen en welke ResponseMessages hierbij horen zijn weergegeven in Tabel 6.1. De beschikbare ResponseMessages die de API terug kan sturen zijn opgenomen in Tabel 6.2.

Tabel 6.1: Beschikbare RequestCommands

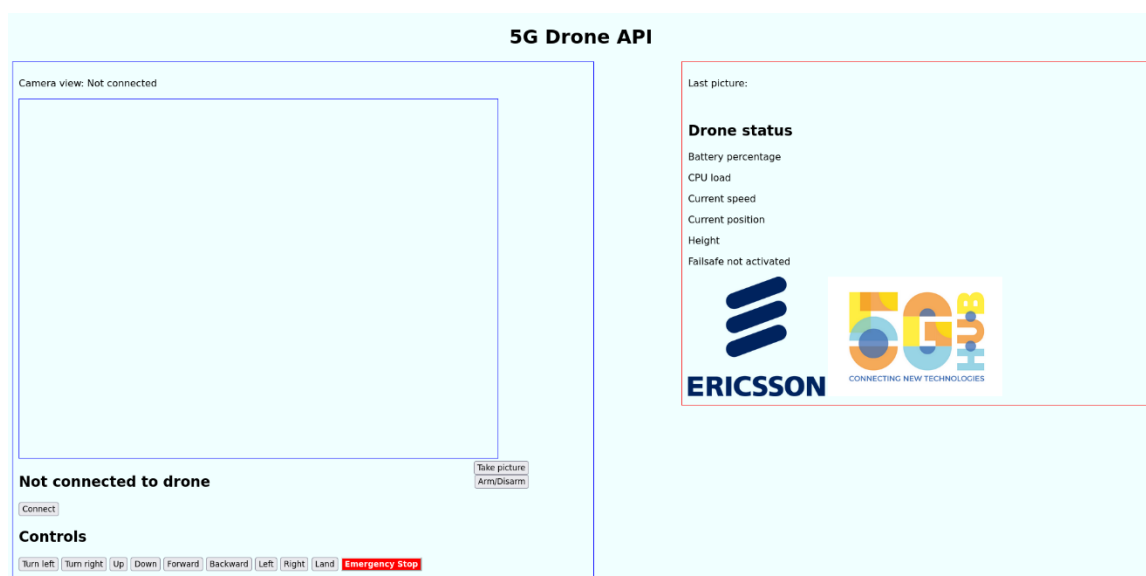
ID	Naam	Betekenis	ResponseMessage
-1	GET_COMMANDS_TYPES	De GUI vraagt de beschikbare RequestCommands en ResponseMessages op.	ALL_REQUESTS_RESPONSES
0	LAND	De drone moet gaan landen	Geen
1	ARM_DISARM	De drone moet worden gearmd/gedisarmd	Geen
3	MOVE_DIRECTION	De drone moet in een bepaalde richting bewegen	MOVE_DIRECTION_RESULT
6	EMERGENCY_STOP	De drone moet een noodstop uitvoeren en de propellers moeten stoppen met draaien	Geen

Tabel 6.2: Beschikbare ResponseMessages

ID	Naam	Betekenis	Requestcommand
-1	ALL_REQUESTS_RESPONSES	Bevat alle mogelijke RequestCommands en ResponseMessages	GET_COMMANDS_TYPES
0	STATUS	Bevat de status van de drone (snelheid, positie, percentage batterij, CPU load, armed status, failsafe status)	Geen
2	MOVE_DIRECTION_RESULT	Resultaat van bewegen in een richting	MOVE_DIRECTION
3	FAILSAFE	Wordt gestuurd als de failsafe is aangezet. Bevat de reden van de failsafe	Geen

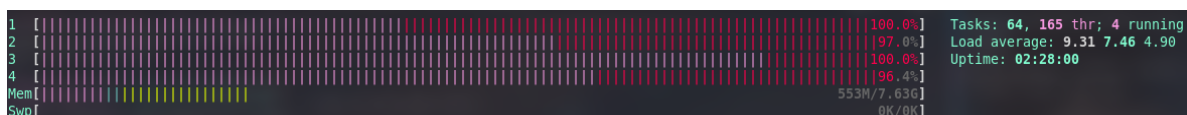
6.3 Implementatie

De *ApiListener* node is te zien in Bijlage A.12: *ApiListener* node. Deze zet de verbinding op met de edge computer en zorgt voor de afhandeling van de berichten die de edge computer stuurt. Ook stuurt deze de status van de drone door naar de edge computer en stuurt het een bericht als de failsafe aangezet wordt, zodat dit weergegeven kan worden aan de gebruiker. De code voor de GUI op de edge computer is te zien in Bijlage A.13: *GUI op edge computer*. Deze stuurt de commando's door naar de API en ontvangt de berichten die de API stuurt. Deze ontvangt ook het camerabeeld van de drone en geeft dit weer aan de gebruiker. De geïmplementeerde GUI is te zien in Figuur 6.3. Hierin zijn de elementen uit Figuur 6.2 zichtbaar. De vormgeving van de GUI is voor dit project niet belangrijk en is dus nog vrij simpel. Het wordt aangeraden om deze aan te passen naar een aantrekkelijkere vormgeving voor klanten.



Figuur 6.3: GUI op edge computer

De communicatie tussen de flight- en edge computer via websockets voor berichten werkt naar behoren. Voor het streamen van video met hoge resolutie is de performance echter te laag. In video 5 in Bijlage C: Overzicht video's is te zien dat de framerate vrij laag is om te gebruiken tijdens het vliegen. Dit komt doordat de flight computer weinig resources over heeft naast het aansturen van alle andere nodes. Dit is weergegeven in Figuur 6.4. Hierin is te zien dat de 4 cores van de flight computer bijna allemaal volledig gebruikt worden. Deze framerate wordt iets beter naarmate de drone langer vliegt. In de video wordt een resolutie van 1024x768 pixels gebruikt. De camera op de drone kan ook maar maximaal met 10 FPS in hoge resolutie filmen. Voor het soepel streamen van video in hoge resolutie wordt aangeraden om een andere oplossing te vinden in de vorm van een andere camera. Ook wordt aangeraden om onderzoek te doen naar de mogelijkheden van een flight computer met een sterkere processor.



Figuur 6.4: De flight computer heeft weinig resources over als alle nodes aangestuurd worden.

6.4 Testen

Om er zeker van te zijn dat de API naar behoren werkt zijn een aantal tests uitgevoerd. Dit zijn functionele tests die uitgevoerd worden door bepaalde stappen te doorlopen en waar te nemen hoe het systeem hierop reageert. Met het slagen van alle tests in deze paragraaf wordt voldaan aan eis *MH7* en *SH2*.

Allereerst is een integratietest opgesteld om te controleren of de integratie tussen de failsafe en status node goed werkt. Hierbij wordt getest of de ApiListener een te laag batterijniveau correct afleest en hierna de failsafe aanzet. De specificatie voor deze test is te zien in Tabel 6.3.

Tabel 6.3: Testspecificatie API integratietest 1

Testnaam	API.I.1
Beschrijving	Deze integratietest wordt uitgevoerd om te controleren of de integratie tussen de failsafe voor de batterij en de ApiListener node correct werkt. Er wordt gekeken of als de status node informatie over een lage batterij publiceert, de ApiListener node dit herkent en de failsafe aanzet.
Requirements	MH7 en SH2
Te testen onderdelen	- ApiListener node - Failsafe node - DroneStatus node
Benodigde nodes	- PositionChanger - PX4Controller - Heartbeat
Voorwaarden	De drone moet opgestart zijn en alle te testen en benodigde nodes moeten opgestart zijn.
Testprocedure	1. Publiceer gedurende 10 seconden <i>/drone/status</i> berichten met een batterij percentage van 10%. 2. Controleer of de Failsafe aangezet wordt 3. Controleer of het bericht van de failsafe de juiste reden bevat.
Verwacht resultaat	De failsafe wordt aangezet en bevat het bericht <i>"Battery level too low! Failsafe enabled to prevent damage to battery"</i> .
Werkelijk resultaat	De failsafe wordt aangezet zodra de ApiListener het <i>/drone/status</i> bericht ontvangt. Het bericht van de failsafe is <i>"Battery level too low! Failsafe enabled to prevent damage to battery"</i> .
Geslaagd?	Ja
Opmerkingen	De code en de output van deze test zijn te zien in Bijlage B.7: ApiListener integratietest.

Hiernaast is een functionele test opgesteld om te controleren of de drone correct opstart en de GUI op de webserver zonder problemen met de API kan verbinden als de drone is opgestart. Hierbij wordt ook getest of correct aan de gebruiker wordt weergegeven of de verbinding nog niet gemaakt kan worden. De specificatie voor deze test is te vinden in Tabel 6.4.

Tabel 6.4: Functionele test verbinden met drone

Tabel 6.4: Functionele test verbinden met drone

Testnaam	API.F.1
Beschrijving	Deze functionele test geeft aan of alle onderdelen van de drone goed opgestart worden en of de gebruiker met de API kan verbinden als de drone wordt opgestart. Hierbij wordt ook getest of een melding aan de gebruiker wordt gegeven als de drone nog niet is opgestart.
Requirements	MH7 en SH2
Te testen onderdelen	- Nodes van drone - GUI op edge computer
Benodigde nodes	- Alle nodes van de drone
Voorwaarden	De webpagina voor de API is geopend en de batterij van de drone is nog niet aangesloten.
Testprocedure	1. Sluit de batterij van de drone aan. 2. Na 10 seconden, probeer te verbinden met de drone door op "Connect" te drukken. 3. Controleer of een melding wordt weergegeven over dat de verbinding is mislukt. 4. Wacht totdat het relais op de drone omschakelt (dit geeft aan dat de RelaisController node opgestart is, en dus de andere nodes ook opgestart zijn). 5. Druk wederom op de "Connect" knop. 6. Controleer of de status data van de drone zichtbaar wordt. 7. Controleer of het camerabeeld van de drone verschijnt.
Verwacht resultaat	Bij de eerste keer verbinden wordt een melding aan de gebruiker gegeven waarin staat dat het verbinden met de API mislukt is. Als even later weer verbinding wordt gemaakt, wordt nu de data van de drone weergegeven en kan het camerabeeld bekeken worden.
Werkelijk resultaat	Bij de eerste keer verbinden wordt een melding correct weergegeven, maar wordt daarna een andere melding weergegeven over dat de verbinding met de API is weggevallen. Bij de tweede keer verbinden wordt de data correct weergegeven en is het camerabeeld zichtbaar.
Geslaagd?	Ja
Opmerkingen	Een video van deze test is te zien in video 7 in Bijlage C: Overzicht video's.

Verder is een functionele test opgesteld om te controleren of het wegvallen van de communicatie tussen de GUI op de edge computer en de API juist wordt weergegeven aan de gebruiker. De specificatie voor deze test is te zien in Tabel 6.5.

Tabel 6.5: Functionele test verliezen communicatie met API

Testnaam	API.F.2
Beschrijving	Deze functionele test wordt uitgevoerd om aan te tonen dat het wegvallen van de communicatie met de drone wordt weergegeven aan de gebruiker.
Requirements	MH7 en SH2
Te testen onderdelen	- GUI op edge computer
Benodigde nodes	- Alle nodes op drone
Voorwaarden	De drone staat aan en de webpagina voor de API is verbonden met de drone.
Testprocedure	1. Schakel de nodes op de flight computer uit door het <code>stop_services.sh</code> script (<code>/home/ubuntu/ros2_ws/drone_scripts/stop_services.sh</code>) op de flight computer uit te voeren. 2. Controleer of een melding wordt gegeven over het verlies van de verbinding met de API. 3. Sluit deze melding door op <i>OK</i> te klikken. 4. Controleer of de status van de camera verandert naar <i>Disconnected</i>. 5. Controleer of de status van de verbinding verandert naar <i>Not connected to drone</i>
Verwacht resultaat	Zodra de nodes op de flight computer worden gestopt moet een melding op de client weergegeven worden over dat de verbinding met de API is weggefallen.
Werkelijk resultaat	Zodra de nodes op de flight computer worden gestopt wordt een melding weergegeven over dat de verbinding met de API is weggefallen. Als deze wordt gesloten zijn de status van de camera en de verbinding met de API veranderd naar de juiste waarden.
Geslaagd?	Ja
Opmerkingen	Een foto van de weergegeven melding is te zien in Figuur B.11 in Bijlage B.10: Output wegvallen verbinding met API. De veranderde status over de verbinding is in deze bijlage te zien in Figuur B.12.

Er is ook een functionele test opgesteld om te testen of de GUI een laag batterijpercentage juist weergeeft aan de gebruiker. Hierbij wordt gekeken of de GUI een melding geeft aan de gebruiker als de failsafe voor een laag batterijpercentage aangezet wordt. De specificatie voor deze test is te zien in Tabel 6.6.

Tabel 6.6: Testspecificatie functionele test voor een te lage batterij

Testnaam	API.F.3
Beschrijving	Deze functionele test wordt uitgevoerd om de functionaliteit van het weergeven van een laag batterijpercentage aan de gebruiker te testen. Er wordt getest of als de failsafe voor een laag batterijpercentage wordt geactiveerd, dit ook wordt weergegeven aan de gebruiker via de API.
Requirements	MH7 en SH2
Te testen onderdelen	- GUI op edge computer
Benodigde nodes	- Alle nodes van de drone
Voorwaarden	Alle nodes zijn opgestart en de webpagina voor de API is geopend. De batterij van de drone is nog niet onder de 15%.
Testprocedure	1. Verbind met de API via de webserver. 2. Laat de drone vliegen en stilhangen op een hoogte van minimaal 1 meter. 3. Wacht tot de batterij lager dan 15% raakt. 4. Controleer of er een melding wordt gegeven aan de gebruiker over het aanzetten van de failsafe 5. Controleer of het bericht van de melding de tekst <i>"Battery level too low! Failsafe enabled to prevent damage to battery"</i> bevat. 6. Controleer of het percentage dat wordt weergegeven lager is dan 15%. 7. Druk op "OK" om de melding te sluiten 8. Controleer of er in het status overzicht "Failsafe: ENABLED" staat. 9. Controleer of de drone gaat landen.
Verwacht resultaat	Zodra het percentage van de batterij onder de 15% komt, wordt een melding weergegeven aan de gebruiker met het bericht <i>"Failsafe enabled! Drone is landing. The failsafe message is: Battery level too low! Failsafe enabled to prevent damage to battery (PERCENTAGE)"</i> . Hierbij moet de waarde van PERCENTAGE lager zijn dan 15%. Als deze melding verkregen is moet de drone gaan landen.
Werkelijk resultaat	Zodra het percentage van de batterij onder de 15% komt, wordt een melding verkregen met het juiste bericht. Het percentage van de batterij wordt weergegeven met te veel getallen achter de komma wat het minder leesbaar maakt, maar het getal is juist. Hierna landt de drone.
Geslaagd?	Ja

Opmerkingen

De weergegeven melding is te zien in Figuur B.10 in Bijlage B.9: Output failsafe door te lage batterij.

Verder is er ook een functionele test opgesteld voor de totale besturing van de drone via de GUI op de edge computer. Hierbij wordt getest of de drone gearmd, gedisarmd en bestuurd kan worden door de knoppen op de GUI te gebruiken. De specificatie voor deze test is te zien in Tabel 6.7. Met het slagen van deze test wordt voldaan aan *SH5*.

Tabel 6.7: Testspecificatie functionele test besturing drone

Testnaam	API.F.4
Beschrijving	Deze functionele test wordt uitgevoerd om te testen of de drone bediend kan worden door de GUI op de edge computer. Er wordt getest of de drone gearmd kan worden, of deze bestuurd kan worden en of deze kan landen.
Requirements	MH7, SH2 en SH5
Te testen onderdelen	- GUI op edge computer
Benodigde nodes	- Alle nodes van de drone
Voorwaarden	De drone staat aan en de GUI op de edge computer heeft verbinding gemaakt met de API.
Testprocedure	1. Arm de drone door op de <i>Arm/disarm</i> knop te drukken. 2. Controleer of de <i>armed</i> status verandert naar <i>Armed: true</i> 3. Laat de drone omhoog vliegen door de knop <i>Up</i> ingedrukt te houden. 4. Laat de drone in de verschillende richtingen vliegen door de knoppen <i>Forward</i>, <i>Backward</i>, <i>Left</i>, <i>Right</i>, <i>Up</i> en <i>Down</i> in te drukken. 5. Controleer of de drone stopt als één van de knoppen losgelaten wordt. 6. Laat de drone naar links of rechts draaien door de knoppen <i>Turn right</i> en <i>Turn left</i> ingedrukt te houden. 7. Laat de drone landen door op de knop <i>Land</i> te drukken. 8. Disarm de drone door op de knop <i>Arm/disarm</i> te drukken. 9. Controleer of de <i>armed</i> status verandert naar <i>Armed: false</i>
Verwacht resultaat	De drone kan worden gearmd en bestuurd door de knoppen op de API. De drone stopt met bewegen als een knop niet meer ingedrukt wordt.
Werkelijk resultaat	De drone kan worden gearmd. De besturing werkt goed en de drone kan in alle richtingen worden bewogen. Als op de <i>Land</i> knop gedrukt wordt begint de drone met landen en hierna kan deze gedisarmd worden. Het is wel opgemerkt dat de locatie en verbinding van de GPS invloed hebben op de stabiliteit van de drone. Als de drone zich dicht bij een rij bomen of een gebouw bevindt, drijft deze meer tijdens het stilhangen dan wanneer deze in een open veld vliegt.
Geslaagd?	Ja

Als laatste test voor de failsafe is een functionele test opgesteld voor het aanzetten van de failsafe vanaf de GUI op de edge computer door de noodstop in te drukken. Hierbij moet de failsafe aangezet worden in de drone en moet dit terug weergegeven worden aan de gebruiker. De specificatie voor deze test is te zien in Tabel 6.8.

Tabel 6.8: Testspecificatie functionele test noodstop

Testnaam	API.F.5
Beschrijving	Deze functionele test wordt uitgevoerd om te controleren of de failsafe aangezet kan worden vanaf de GUI op de edge computer door op de noodstop te drukken.
Requirements	MH7 en SH2
Te testen onderdelen	- GUI op edge computer
Benodigde nodes	- Alle nodes op de drone
Voorwaarden	De drone staat aan en de GUI heeft verbinding gemaakt met de API.
Testprocedure	1. Druk op de rode <i>emergency stop</i> knop op de GUI. 2. Controleer of er een melding wordt weergegeven over dat de failsafe is aangezet 3. Sluit de melding door op <i>OK</i> te drukken 4. Controleer of de failsafe status veranderd is naar <i>failsafe: ACTIVATED</i>
Verwacht resultaat	Als op de noodstop ingedrukt wordt, wordt daarna een melding weergegeven over dat de failsafe aangezet is. De reden van de failsafe moet zijn "Emergency stop activated". Als deze gesloten wordt is de failsafe status rood en veranderd naar <i>failsafe: ENABLED</i> .
Werkelijk resultaat	Als de noodstop wordt ingedrukt verschijnt na een halve seconde de melding over dat de failsafe aangezet is met het bericht "Emergency stop activated". Hierna is de failsafe status veranderd naar <i>failsafe: ACTIVATED</i> .
Geslaagd?	Ja
Opmerkingen	Het resultaat van de melding is te zien in Figuur B.8 in Bijlage B.8: Output failsafe aanzetten door noodstop op GUI. In Figuur B.9 in deze bijlage is te zien dat de failsafe status goed veranderd wordt.

Als laatst is een functionele test opgesteld om te controleren of het beeld vanaf de camera ontvangen kan worden op de edge computer en of hiermee foto's gemaakt kunnen worden om de antennes te inspecteren. De specificatie hiervan is te zien in . Met het slagen van deze test is voldaan aan eis *MH6* en *SH6*.

Testnaam	API.F.6
Beschrijving	Deze test wordt uitgevoerd om te controleren of het videobeeld van de camera op de edge computer ontvangen kan worden en of de gebruiker van de webpagina deze kan bekijken. Ook wordt getest of een foto gemaakt kan worden vanaf de webpagina.
Requirements	MH6 en SH6
Te testen onderdelen	- GUI op edge computer
Benodigde nodes	- Alle nodes op de drone
Voorwaarden	De batterij is aangesloten en de drone staat aan. De edge computer is nog niet verbonden met de API op de drone.
Testprocedure	1. Maak verbinding met de drone door op de knop <i>Connect</i> te drukken. 2. Wacht tot verbinding wordt gemaakt. 3. Als de verbinding is gemaakt, controleer of een bewegend videobeeld van de camera ontvangen wordt. 4. Druk op de knop <i>Take picture</i> om een foto te maken. 5. Controleer of de foto de inhoud van het camerabeeld bevat.
Verwacht resultaat	Bij het verbinden met de drone wordt het videobeeld van de camera doorgestuurd en deze kan bekeken worden. Als een foto genomen wordt, bevat deze foto de inhoud van het camerabeeld op het moment dat de knop ingedrukt is.
Werkelijk resultaat	Nadat de edge computer verbonden is met de drone, wordt het videobeeld van de camera ontvangen. De framerate van de video is niet hoog maar wel bruikbaar. Als een foto genomen wordt bevat deze de correcte inhoud.
Geslaagd?	Ja
Opmerkingen	Het nemen van de foto is te zien aan het eind van video 8 in Bijlage C: Overzicht video's. Het videobeeld is te zien in deze bijlage in video 7.

6.5 Conclusie API

Het doel van de API is om een manier te bieden om de drone via 5G vanaf een edge computer te kunnen besturen. De API wordt op de flight controller uitgevoerd en deze gedraagt zich als server. De edge computer gedraagt zich als client.

De drone kan op afstand bestuurd worden door de API aan te sturen via de gebruikersinterface op de edge computer. Dit is een webpagina die vanaf elk apparaat op het wifi netwerk van de 5G Hub te benaderen is.

Voor de communicatie tussen de API en de edge computer is gekozen voor websockets. Deze keuze is gemaakt omdat er een constante verbinding tussen de drone en de edge computer nodig is. Op de edge computer draait een webserver waar apparaten die verbonden zijn met het wifi-netwerk van de 5G Hub mee kunnen verbinden.

Voor de edge computer is een VM aangemaakt met een processor met 8 cores en 16 GB RAM. De edge computer en API communiceren met elkaar via JSON berichten. De gerealiseerde GUI is te zien in Figuur 6.3 in de vorm van een webpagina. Voor het streamen van hoge resolutie video is de performance te laag, omdat de flight computer weinig resources over heeft naast het aansturen van alle andere nodes. Dit komt ook omdat de camera maximaal maar met 10 FPS op hoge resolutie kan filmen. Het wordt dus aangeraden om een oplossing te zoeken voor een andere camera of een flight computer met een sterkere processor. Om er zeker van te zijn dat de API naar behoren werkt zijn een aantal integratietest uitgevoerd. Deze zijn te zien in Bijlage B: Code en output tests. Door het slagen van deze tests voldoet de drone aan eis *MH7* en *SH2*. Door het slagen van test API.F.4 wordt voldaan aan eis *SH5*. Door het slagen van test API.F.6 wordt voldaan aan eis *MH6* en *SH6*.

7 Conclusies en aanbevelingen

Dit hoofdstuk weergeeft de conclusies die getrokken zijn uit de vorige hoofdstukken. Hiermee wordt een antwoord gegeven op de hoofdvraag. Ook worden aanbevelingen gegeven over hoe het project verder ontwikkeld kan worden. In Paragraaf 7.1 worden de conclusies van het verslag gegeven. Paragraaf 7.2 geeft aanbevelingen voor de verdere ontwikkeling van het project.

7.1 Conclusies

Het doel van dit verslag is antwoord geven op de vraag *“Hoe kan een drone ontwikkeld worden die met 5G aangestuurd kan worden om zo op afstand autonoom inspecties uit te voeren?”*.

De drone gebruikt een Pixhawk 4 Mini flight controller. Deze heeft autopilot software nodig om de drone autonoom te kunnen laten vliegen. Als autopilot software moet de 1.14 beta versie van PX4 gebruikt worden. Deze communiceert via ROS 2 en XRCE-DDS met de flight computer. De PX4 software moet aangepast zijn om ook het percentage van de batterij en belasting van de processor uit te kunnen lezen.

De flight controller heeft een schatting van de positie nodig om stabiel op snelheid aangestuurd te kunnen worden. Dit is nog niet mogelijk met de bestelde onderdelen. Om toch te kunnen vliegen gebruikt de drone GPS en vliegt deze alleen buiten. Hierdoor kan deze wel gebruikt worden om 5G antennes te inspecteren die buiten hangen. Bij de aansturing via 5G wordt gebruik gemaakt van de hogere capaciteit dan 4G, beamforming en handovers.

De flight computer is een Raspberry Pi 4B+. Deze leest de sensoren op de drone uit, stuurt de flight controller aan en zorgt voor de communicatie met de edge computer. Deze gebruikt Ubuntu 20 Server als OS en ROS 2 Foxy om met de flight controller te communiceren. De flight computer is vervangen door een versie met 8 GB RAM in plaats van 2 GB om met de flight controller te kunnen communiceren.

De motoren en propellers zijn vervangen omdat de originele niet voldoende waren om de drone te kunnen laten vliegen. De drone kan op afstand via 5G bestuurd worden door de API aan te sturen via de gebruikersinterface op de edge computer. Dit is een webpagina die vanaf elk apparaat op het wifi netwerk van de 5G Hub te benaderen is.

7.2 Aanbevelingen

Voor het binnen vliegen met de drone wordt als eerst aanbevolen om op de hoogte te blijven van de ontwikkeling van PX4 en in de gaten te houden of een nieuwe update uitgegeven wordt waarmee de optical flow sensor gebruikt kan worden. Voor de constructie van de drone wordt aanbevolen om een aantal onderdelen te vervangen door 3D geprinte onderdelen. Dit zijn een houder voor de camera en een behuizing voor de flight computer en de 5G modem. Naast deze onderdelen wordt ook sterk aanbevolen om beschermkappen te ontwerpen en te 3D printen zodat de propellers van de drone niks kunnen raken als de drone ergens te dicht bij in de buurt komt.

Verder wordt aangeraden om te zoeken naar een andere camera die met meer dan 10 FPS in hoge resolutie kan filmen. Omdat de flight computer weinig resources over heeft na het aansturen van alle nodes, wordt ook aangeraden om onderzoek te doen naar een flight computer met een sterkere processor.

Omdat het niet gelukt is om het scherm op de drone te implementeren, wordt aangeraden om dit na het project te doen. De aanbevolen manier hiervoor is door naar de framebuffer van de flight computer te schrijven om resources te besparen. Ook is de gebruikersinterface van de webpagina op de edge computer vrij simpel. Om een aantrekkelijke demo voor klanten te realiseren wordt aanbevolen om de vormgeving hiervan aantrekkelijker te maken.

Aangezien de drone nog niet binnen kan vliegen, maar alleen buiten met GPS, wordt aangeraden om de drone nu te gebruiken om de 5G antennes buiten te inspecteren. Hiermee kan wel een demo voor bezoekers van de 5G Hub gerealiseerd worden. Voor het buiten vliegen met de drone is echter wel een vergunning nodig.

8 Bibliografie

- 5G Hub. (2023). 5G Hub neemt zijn raceauto de straat op. Eindhoven. Opgeroepen op Maart 29, 2023, van <https://5ghub.nl/nl/2021/05/12/5g-hub-neemt-zijn-raceauto-de-sstraat-op/>
- Ademovic, A. (2015). *An Introduction to Robot Operating System: The Ultimate Robot Application Framework*. Opgeroepen op Maart 13, 2023, van An Introduction to Robot Operating System: The Ultimate Robot Application Framework: <https://www.toptal.com/robotics/introduction-to-robot-operating-system>
- Alexander, T. (2022). *ROS 2 on Raspberry Pi*. Opgeroepen op Maart 13, 2023, van ROS 2 on Raspberry Pi: <https://docs.ros.org/en/humble/How-To-Guides/Installing-on-Raspberry-Pi.html>
- Ansic. (2022, Mei 15). *RPI 4 won't boot up when UART is connected*. Opgeroepen op Mei 26, 2023, van Raspberry Pi Forums: <https://forums.raspberrypi.com/viewtopic.php?t=334488>
- ArduPilot. (2022). *What is a MultiCopter and How Does it Work?: What is the difference between a MultiCopter and a UAV/Drone?* Opgeroepen op Maart 8, 2023, van ArduPilot.org: <https://ardupilot.org/copter/docs/what-is-a-multicopter-and-how-does-it-work.html#what-is-the-difference-between-a-multicopter-and-a-uav-drone>
- Arguedas, M., Ragnarok, S., Thomas, D., Nash, A., & Biggs, G. (2022). *REP 2000: ROS 2 Releases and Target Platforms*.
- Ayrancıoğlu, G. (2022). *What is Server-Sent Events (SSE) and how to implement it?* Opgeroepen op Maart 15, 2023, van What is Server-Sent Events (SSE) and how to implement it?: <https://medium.com/yemeksepeti-teknoloji/what-is-server-sent-events-sse-and-how-to-implement-it-904938bffd73>
- Bilbao, A. (2017). *ROS 2 RPi 3 prerequisites*. Opgeroepen op April 11, 2023, van GitHub: <https://gist.github.com/abilbaotm/eaf12299c7a37d867713b5f42e820470>
- Canonical. (2023). *Ubuntu Core*. Opgeroepen op April 4, 2023, van Ubuntu.com: <https://ubuntu.com/core>
- Defining Drones. (2023, Februari 2). *Do Drones Have Autopilot? | Different Types Of Autonomous UAV Flight*. Opgeroepen op Juni 15, 2023, van Website van Defining Drones: <https://definingdrones.com/drone-autopilot-autonomous-flight/>
- Docker. (2023). *Runtime privilege and Linux capabilities*. Opgeroepen op Maart 13, 2023, van <https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities>
- eProsima. (2018). *Micro XRCE-DDS Overview*. Opgeroepen op Maart 14, 2023, van Micro XRCE-DDS Overview: <https://micro-xrce-dds.docs.eprosima.com/en/stable/introduction.html>
- Ericsson. (2023a). About Us. Opgeroepen op Maart 29, 2023, van <https://www.ericsson.com/en/about-us>
- Ericsson. (2023b). Ready to create the future? Opgeroepen op Maart 29, 2023, van <https://www.ericsson.com/en/about-us/new-world-of-possibilities>
- Ericsson. (2023c). *Passive antenna*. Opgeroepen op Juni 16, 2023, van Website van Ericsson: <https://www.ericsson.com/en/portfolio/networks/ericsson-radio-system/antenna-system/antenna/passive-antenna>
- Guowang Miao, J. Z. (2016). 8 - Association and handover. In G. Miao, *Fundamentals of Mobile Data Networks* (p. 199). Cambridge University Press.
- HBO-i. (2022). HBO-i research methods. Opgeroepen op Maart 19, 2023, van <https://ictresearchmethods.nl/Methods>
- Hoeven, S. v. (2023). *5G RCID: Voorstel veranderen motoren*. Bergen op Zoom: Avans Hogeschool Breda.
- Hoeven, S. v. (2023, Maart). *Can I mount ESCs underneath the rotors of my hexacopter drone like this? Will it impact flight performance?* Opgeroepen op April 14, 2023, van Reddit: https://www.reddit.com/r/diydrones/comments/11sv3mc/can_i_mount_escs_underneath_the_rotors_of_my/

- Hoeven, S. v. (2023, April 6). *Cannot arm drone with companion computer: "Arming denied! manual control lost"*. Opgeroepen op April 14, 2023, van PX4 Forum: <https://discuss.px4.io/t/cannot-arm-drone-with-companion-computer-arming-denied-manual-control-lost/31565>
- Hoeven, S. v. (2023a). *Intelligentie voor een 5G aangestuurde drone*. Bergen op Zoom: Avans Hogeschool Breda.
- Hoeven, S. v. (2023b). *Plan van Aanpak*. Bergen op Zoom: Avans Hogeschool Breda.
- Hoeven, S. v. (2023c, Februari 15). *Starting QGroundControl ApplImage gives symbol lookup error*. Opgeroepen op April 11, 2023, van GitHub: <https://github.com/mavlink/qgroundcontrol/issues/10578>
- Hughes, M. (2017, Maart 10). *Don't Get Lost in Deep Space: Understanding Quaternions*. Opgeroepen op Mei 26, 2023, van All about circuits: <https://www.allaboutcircuits.com/technical-articles/dont-get-lost-in-deep-space-understanding-quaternions/>
- Jithin. (2016). *Common Network Protocols and Their Ports*. Opgeroepen op Maart 15, 2023, van Common Network Protocols and Their Ports: <https://www.interserver.net/tips/kb/common-network-protocols-ports/>
- Johl, J. (2022). *What are APIs and how do APIs work?* Opgeroepen op Maart 15, 2023, van What are APIs and how do APIs work?: <https://blogs.mulesoft.com/learn-apis/api-led-connectivity/what-are-apis-how-do-apis-work/>
- Klimaj, A. (2023, Januari 14). *Pre flight failure with Optical Flow on Ground*. Opgeroepen op Mei 26, 2023, van PX4 Github: <https://github.com/PX4/PX4-Autopilot/issues/20929>
- Lunsing, H. (2018). *Ubuntu voor beginners*. Opgeroepen op Maart 13, 2023, van Ubuntu voor beginners: <https://www.compusers.nl/linux/ubuntu-beginners>
- Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7, eabm6074. doi:10.1126/scirobotics.abm6074
- MarvelMind Robotics. (2021). *Starter Set Super-MP-3D*. Opgeroepen op Mei 26, 2023, van Website van MarvelMind Robotics: <https://marvelmind.com/product/starter-set-super-mp-3d/>
- MarvelMind Robotics. (2022). *Marvelmind and PX4 integration*. Opgehaald van https://marvelmind.com/pics/marvelmind_px4_integration.pdf
- MarvelMind Robotics. (2023). *Marvelmind Indoor Navigation System Operating Manual*. Opgehaald van https://marvelmind.com/pics/marvelmind_navigation_system_manual.pdf
- MAVLink. (2021). *MAVROS*. Opgeroepen op Maart 14, 2023, van MAVROS: <https://github.com/mavlink/mavros/blob/ros2/mavros/README.md>
- MAVLink. (2022). *MAVLink Developer Guide*. Opgeroepen op March 8, 2023, van MAVLink Developer Guide: <https://mavlink.io/en/>
- Munguia, R., & Grau, A. (2014). A Practical Method for Implementing an Attitude and Heading Reference System. *International Journal of Advanced Robotic Systems*.
- Narkhede, P., Poddar, S., Walambe, R., Ghinea, G., & Kotecha, K. (2021). Cascaded Complementary Filter Architecture for Sensor Fusion in Attitude Estimation. *Sensors*, 6.
- Noury, G. A. (2021). *ROS Docker; 6 reasons why they are not a good fit*. Opgeroepen op Maart 13, 2023, van ROS Docker; 6 reasons why they are not a good fit: <https://ubuntu.com/blog/ros-docker>
- Ogundeyi, K. E., & Yinka-Banjo, C. (2019). Websocket in Real Time Application. *Nigerian Journal of Technology*, 38. doi:10.4314/njt.v38i4.26
- Pixhawk. (2021). *Radio Control Systems*. Opgeroepen op April 11, 2023, van PX4 handleiding: https://docs.px4.io/main/en/getting_started/rc_transmitter_receiver.html
- Pixhawk. (2022a, December 7). *DS-011 Pixhawk Autopilot v5X Standard*. Pixhawk.
- Pixhawk. (2022b). *Airframes reference*. Opgeroepen op April 11, 2023, van PX4 handleiding: https://docs.px4.io/main/en/airframes/airframe_reference.html#hexarotor-x
- Pixhawk. (2022c). *PMW3901-Based Flow Sensors*. Opgeroepen op Mei 21, 2023, van PX4 documentatie: <https://docs.px4.io/main/en/sensor/pmw3901.html>

- Pixhawk. (2023a, April 6). *ROS 2 User Guide*. Opgeroepen op April 7, 2023, van PX4 documentatie: https://docs.px4.io/main/en/ros/ros2_comm.html
- Pixhawk. (2023b, Maart 17). *RTPS/DDS Interface: PX4-Fast RTPS(DDS) Bridge*. Opgeroepen op April 7, 2023, van PX4 documentatie: <https://docs.px4.io/v1.13/en/middleware/micrortps.html>
- Pixhawk. (2023c, April 22). *Offboard mode*. Opgeroepen op April 24, 2023, van PX4 Documentatie: https://docs.px4.io/main/en/flight_modes/offboard.html
- Pixhawk. (2023d, Mei 5). *Parameter reference*. Opgeroepen op Mei 5, 2023, van PX4 documentatie: https://docs.px4.io/main/en/advanced_config/parameter_reference.html
- Pixhawk. (2023e, Maart 5). *Optical Flow*. Opgeroepen op Mei 21, 2023, van PX4 documentatie: https://docs.px4.io/main/en/sensor/optical_flow.html
- Pixhawk. (2023f). *Pixhawk 4 Mini*. Opgeroepen op Maart 7, 2023, van Pixhawk 4 Mini: https://docs.px4.io/main/en/flight_controller/pixhawk4_mini.html
- Pixhawk. (2023g, Mei 20). *Using the ECL EKF*. Opgeroepen op Mei 26, 2023, van PX4 documentatie: https://docs.px4.io/main/en/advanced_config/tuning_the_ecl_ekf.html
- Pixhawk. (2023h, Mei 16). *Parameter reference: UAVCAN*. Opgeroepen op Mei 26, 2023, van PX4 documentatie: https://docs.px4.io/main/en/advanced_config/parameter_reference.html#uavcan
- Pixhawk. (2023i, Mei 10). *ROS 2 User Guide: Custom uORB Topics*. Opgeroepen op Mei 26, 2023, van PX4 documentatie: https://docs.px4.io/main/en/ros/ros2_comm.html#custom-uorb-topics
- Pixhawk. (2023j, 26 April). *Controller diagrams: multicopter attitude controller*. Opgeroepen op Mei 26, 2023, van PX4 documentatie: https://docs.px4.io/main/en/flight_stack/controller_diagrams.html#multicopter-attitude-controller
- Putten, L. v. (sd). *5G antenne*. Hollandse Hoogte, Rotterdam.
- Raspberry Pi. (2023). *Raspberry Pi 4*. Opgeroepen op Juni 16, 2023, van Website van Raspberry Pi: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- Rehm, N. (2021). *The Difference Between a Drone Flight Controller and Flight Computer*. Opgehaald van The Difference Between a Drone Flight Controller and Flight Computer: <https://www.youtube.com/watch?v=PlKeFj5teo4>
- Richards, P. (2021, Maart 24). *What Type of Computer Do I Need to Live Stream?* Opgeroepen op Mei 26, 2023, van StreamGeeks: <https://streamgeeks.us/what-computer-do-i-need-to-live-stream/>
- Robotics Backend. (2023). *Using ROS on Raspberry Pi: Best Practices*. Opgeroepen op Maart 13, 2023, van Using ROS on Raspberry Pi: Best Practices: <https://roboticsbackend.com/using-ros-on-raspberry-pi-best-practices/>
- Shaw, K. (2022, Maart 14). *What is beamforming and how does it make wireless better?* Opgeroepen op Juni 13, 2023, van NetworkWorld: What is beamforming and how does it make wireless better?
- SkyWalker. (2015). *HW-SM003DUL-20150512: User Manual of Brushless Speed Controller*.
- Tarot. (2023). *Tarot 4006 High Power Brushless Motor (620kv)*. Opgeroepen op Maart 17, 2023, van Helipal: <https://www.helipal.com/products/tarot-parts-4006-high-power-brushless-motor-620kv>
- Tawil, Y. (2017). *An Introduction to Robot Operating System (ROS)*. Opgeroepen op Maart 13, 2023, van An Introduction to Robot Operating System (ROS): <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/>
- The Drone Dojo. (2021). *Make Your Own Pixhawk Raspberry Pi Drone in 36 Minutes (2020) | The Ultimate Project Drone*. Opgeroepen op April 14, 2023, van <https://www.youtube.com/watch?v=kB9YyG2V-nA&t=1579s>
- Torver, P. K. (2022). *Realtime data streaming using server-sent events(SSE) with react.js and node.js*. Opgeroepen op Maart 15, 2023, van Realtime data streaming using server-sent events(SSE)

- with react.js and node.js: <https://dev.to/techfortified/realtime-data-streaming-using-server-sent-events-sse-with-reactjs-and-nodejs-2aak>
- Verizon. (2022). *What is 5G beamforming?* Opgeroepen op Juni 13, 2023, van Verizon.com: <https://www.verizon.com/business/resources/articles/5g-beamforming-massive-mimo/>
- White, F. E. (1991). Data Fusion Lexicon. *Data Fusion Lexicon*.
- Wilcher, D. (2019). *ROS 101: An Intro to the Robot Operating System*. Opgeroepen op Maart 13, 2023, van ROS 101: An Intro to the Robot Operating System: <https://www.designnews.com/gadget-freak/ros-101-intro-robot-operating-system>
- Willee, H. (2023). *XRCE-DDS (PX4-FastDDS Bridge)*. Opgehaald van XRCE-DDS (PX4-FastDDS Bridge): https://docs.px4.io/main/en/middleware/xrce_dds.html
- Willee, H., & Chiap, R. (2022). *PX4 Architectural Overview*. Opgehaald van PX4 documentatie: <https://docs.px4.io/main/en/concept/architecture.html>
- Willee, H., & Sriram, K. (2022). *PX4 System Architecture*. Opgehaald van PX4 System Architecture: https://docs.px4.io/main/en/concept/px4_systems_architecture.html
- Willee, H., Fjare, P., & Poon, V. (2022). *Pixhawk 4 Mini: Serial Port mapping*. Opgeroepen op Maart 14, 2023, van Pixhawk 4 Mini: Serial Port mapping: http://docs.px4.io/main/en/flight_controller/pixhawk4_mini.html#serial-port-mapping
- Willee, H., Oes, J., & Otu, J. (2022). *uORB Messaging*. Opgeroepen op Maart 14, 2023, van uORB Messaging: <https://docs.px4.io/main/en/middleware/uorb.html>
- Willee, H., Roche, R., Higgason, J., Marques, N., & Hwang, J. (2023). *ROS (Robot Operating System)*. Opgeroepen op Maart 14, 2023, van ROS (Robot Operating System): <https://docs.px4.io/main/en/ros/>
- Zanj, S. (2018). *Extended Kalman Filter*. Opgeroepen op March 8, 2023, van Extended Kalman Filter: <https://medium.com/@siddheshzanj/extended-kalman-filter-94fe07fd5c79>
- Zola, A. (2023). *Polling (computing)*. Opgeroepen op Maart 15, 2023, van Polling (computing): <https://www.techtarget.com/whatis/definition/polling>

Bijlage A: ROS 2 nodes

A.1: PositionChanger node

Dit is de node die de drone aanstuurt op basis van commando's die de ApiListener ontvangt. Deze ontvangt ook de data van de LIDAR en zorgt dat de drone nergens tegenaan vliegt. Verder ontvangt deze de data van de hoogtemeter die wordt gebruikt tijdens het landen.

```
1  #include "rclcpp/rclcpp.hpp"
2
3  #include <px4_msgs/msg/vehicle_odometry.hpp>
4  #include <drone_services/msg/lidar_reading.hpp>
5  #include <drone_services/msg/height_data.hpp>
6  #include <drone_services/srv/set_vehicle_control.hpp>
7  #include <drone_services/srv/set_trajectory.hpp>
8  #include <drone_services/srv/set_attitude.hpp>
9  #include <drone_services/srv/move_position.hpp>
10 #include <drone_services/srv/ready_drone.hpp>
11 #include <drone_services/srv/arm_drone.hpp>
12 #include <drone_services/srv/land.hpp>
13
14 #include <drone_services/srv/enable_failsafe.hpp>
15 #include <drone_services/msg/failsafe_msg.hpp>
16
17 #define _USE_MATH_DEFINES
18 #include <cmath>
19
20 #define LIDAR_SENSOR_FR 0 // front right
21 #define LIDAR_SENSOR_FL 1 // front left
22 #define LIDAR_SENSOR_RL 2 // rear left
23 #define LIDAR_SENSOR_RR 3 // rear right
24
25 #define MOVE_DIRECTION_FRONT 0 // position in array for moving forward
26 #define MOVE_DIRECTION_LEFT 1 // position in array for moving left
27 #define MOVE_DIRECTION_BACK 2 // position in array for moving backward
28 #define MOVE_DIRECTION_RIGHT 3 // position in array for moving right
29
30 #define MIN_DISTANCE 1.0 // in meters
31
32 #define CONTROL_MODE_ATTITUDE 4 // attitude control mode bitmask
33 #define DEFAULT_CONTROL_MODE 16 // default velocity control bitmask
34 // converts bitmask control mode to control mode used by PX4Controller
35 #define PX4_CONTROLLER_CONTROL_MODE(x) ((x) == 4 ? 1 : ((x) == 16 ? 2 : ((x) == 32 ? 3 : -1)))
36
37 #define HEIGHT_DELTA 0.1
38
39 using namespace std::chrono_literals;
40
41 struct Quaternion
42 {
43     float w, x, y, z;
44 };
45
46 class PositionChanger : public rclcpp::Node
47 {
48 public:
49     PositionChanger() : Node("position_changer")
50     {
51         rmw_qos_profile_t qos_profile = rmw_qos_profile_sensor_data;
52         auto qos = rclcpp::QoS(rclcpp::QoSInitialization(qos_profile.history, 5), qos_profile);
53
54         this->lidar_subscription = this->create_subscription<drone_services::msg::LidarReading>("/drone/object_detection", qos,
55             std::bind(&PositionChanger::handle_lidar_message, this, std::placeholders::_1));
56         this->height_subscription = this->create_subscription<drone_services::msg::HeightData>("/drone/height", qos,
57             std::bind(&PositionChanger::handle_height_message, this, std::placeholders::_1));
58         this->odometry_subscription = this->create_subscription<px4_msgs::msg::VehicleOdometry>("/fmu/out/vehicle_odometry", qos,
59             std::bind(&PositionChanger::handle_odometry_message, this, std::placeholders::_1));
60         this->move_position_service = this->create_service<drone_services::srv::MovePosition>("/drone/move_position",
61             std::bind(&PositionChanger::handle_move_position_request, this, std::placeholders::_1, std::placeholders::_2,
62                 std::placeholders::_3));
63         this->ready_drone_service = this->create_service<drone_services::srv::ReadyDrone>("/drone/ready",
64             std::bind(&PositionChanger::handle_ready_drone_request, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
65         this->land_service = this->create_service<drone_services::srv::Land>("/drone/land",
66             std::bind(&PositionChanger::handle_land_request, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
67
68         this->failsafe_publisher = this->create_publisher<drone_services::msg::FailsafeMsg>("/drone/failsafe", 10);
69
70         this->trajectory_client = this->create_client<drone_services::srv::SetTrajectory>("/drone/set_trajectory");
71         this->attitude_client = this->create_client<drone_services::srv::SetAttitude>("/drone/set_attitude");
72         this->vehicle_control_client = this->create_client<drone_services::srv::SetVehicleControl>("/drone/set_vehicle_control");
73         this->failsafe_client = this->create_client<drone_services::srv::EnableFailsafe>("/drone/enable_failsafe");
74         this->arm_drone_client = this->create_client<drone_services::srv::ArmDrone>("/drone/arm");
75
76         RCLCPP_INFO(this->get_logger(), "waiting for trajectory service...");
77         wait_for_service<rclcpp::Client<drone_services::srv::SetTrajectory>::SharedPtr>(this->trajectory_client,
78             "/drone/set_trajectory");
79         RCLCPP_INFO(this->get_logger(), "waiting for attitude service...");
80         wait_for_service<rclcpp::Client<drone_services::srv::SetAttitude>::SharedPtr>(this->attitude_client, "/drone/set_attitude");
81         RCLCPP_INFO(this->get_logger(), "waiting for vehicle control service...");
82         wait_for_service<rclcpp::Client<drone_services::srv::SetVehicleControl>::SharedPtr>(this->vehicle_control_client,
83             "/drone/set_vehicle_control");
84         RCLCPP_INFO(this->get_logger(), "waiting for failsafe service...");
```

```

76     wait_for_service<rclcpp::Client<drone_services::srv::EnableFailsafe>::SharedPtr>(this->failsafe_client,
77     "/drone/enable_failsafe");
78     RCLCPP_INFO(this->get_logger(), "waiting for arm service...");
79     wait_for_service<rclcpp::Client<drone_services::srv::ArmDrone>::SharedPtr>(this->arm_drone_client, "/drone/arm");
80
81     this->trajectory_request = std::make_shared<drone_services::srv::SetTrajectory::Request>();
82     this->attitude_request = std::make_shared<drone_services::srv::SetAttitude::Request>();
83     this->vehicle_control_request = std::make_shared<drone_services::srv::SetVehicleControl::Request>();
84     this->failsafe_request = std::make_shared<drone_services::srv::EnableFailsafe::Request>();
85     this->arm_drone_request = std::make_shared<drone_services::srv::ArmDrone::Request>();
86
87     lidar_health_timer = this->create_wall_timer(1s, std::bind(&PositionChanger::check_lidar_health, this));
88
89     this->vehicle_control_request->control = DEFAULT_CONTROL_MODE;
90     auto control_mode_response = this->vehicle_control_client->async_send_request(this->vehicle_control_request,
91     std::bind(&PositionChanger::vehicle_control_service_callback, this, std::placeholders::_1));
92 }
93
94 /**
95  * @brief callback function for the vehicle control service
96  * @param future the future object that holds the result of the service call
97  */
98 void vehicle_control_service_callback(rclcpp::Client<drone_services::srv::SetVehicleControl>::SharedPtr future)
99 {
100     auto status = future.wait_for(1s);
101     if (status == std::future_status::ready)
102     {
103         RCLCPP_INFO(this->get_logger(), "Vehicle control mode set result: %d", future.get()->success);
104         if (this->got_ready_drone_request && future.get()->success)
105         {
106             auto arm_response = this->arm_drone_client->async_send_request(this->arm_drone_request,
107             std::bind(&PositionChanger::arm_drone_service_callback, this, std::placeholders::_1));
108         }
109     }
110     else
111     {
112         RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
113     }
114 }
115
116 /**
117  * @brief callback function for the /drone/arm service
118  * @param future the future object that holds the result of the service call
119  */
120 void arm_drone_service_callback(rclcpp::Client<drone_services::srv::ArmDrone>::SharedPtr future)
121 {
122     auto status = future.wait_for(1s);
123     if (status == std::future_status::ready)
124     {
125         RCLCPP_INFO(this->get_logger(), "Arm result: %d", future.get()->success);
126         if (this->got_ready_drone_request)
127         {
128             this->got_ready_drone_request = false;
129
130             this->attitude_request->pitch = 0.0;
131             this->attitude_request->yaw = 0.0;
132             this->attitude_request->roll = 0.0;
133             this->attitude_request->thrust = 0.15;
134             auto attitude_response = this->attitude_client->async_send_request(this->attitude_request,
135             std::bind(&PositionChanger::attitude_message_callback, this, std::placeholders::_1));
136         }
137     }
138     else
139     {
140         RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
141     }
142 }
143
144 /**
145  * @brief callback function for the /drone/set_attitude service
146  * @param future the future object that holds the result of the service call
147  */
148 void attitude_message_callback(rclcpp::Client<drone_services::srv::SetAttitude>::SharedPtr future)
149 {
150     auto status = future.wait_for(1s);
151     if (status == std::future_status::ready)
152     {
153         RCLCPP_INFO(this->get_logger(), "Attitude set result: %d", future.get()->success);
154     }
155     else
156     {
157         RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
158     }
159 }
160
161 /**
162  * @brief callback function for the trajectory service
163  * @param future the future object that holds the result of the service call
164  */
165 void trajectory_message_callback(rclcpp::Client<drone_services::srv::SetTrajectory>::SharedPtr future)
166 {
167     auto status = future.wait_for(1s);
168     if (status == std::future_status::ready)
169     {
170         RCLCPP_INFO(this->get_logger(), "Set trajectory set result: %d", future.get()->success);
171     }

```

```

171     }
172     else
173     {
174         RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
175     }
176 }
177
178 /**
179  * @brief callback function for the failsafe service
180  *
181  * @param future the future object that holds the result of the service call
182  */
183 void failsafe_message_callback(rclcpp::Client<drone_services::srv::EnableFailsafe>::SharedFuture future)
184 {
185     auto status = future.wait_for(1s);
186     if (status == std::future_status::ready)
187     {
188         RCLCPP_INFO(this->get_logger(), "Set failsafe enabled: %d, message: %s", future.get()->enabled, future.get()->message);
189     }
190     else
191     {
192         RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
193     }
194 }
195
196 /**
197  * @brief sends the current x, y and z speed and yaw to the trajectory service
198  *
199  */
200 void send_trajectory_message()
201 {
202     this->trajectory_request->values[0] = this->current_speed_x;
203     this->trajectory_request->values[1] = this->current_speed_y;
204     this->trajectory_request->values[2] = this->current_speed_z;
205     this->trajectory_request->yaw = this->current_yaw;
206     this->trajectory_request->control_mode = PX4_CONTROLLER_CONTROL_MODE(DEFAULT_CONTROL_MODE);
207     RCLCPP_INFO(this->get_logger(), "Sending trajectory message\nx: %f\ny: %f\nz: %f\nyaw: %f", this->current_speed_x, this-
>current_speed_y, this->current_speed_z, this->current_yaw);
208     auto trajectory_response = this->trajectory_client->async_send_request(this->trajectory_request,
std::bind(&PositionChanger::trajectory_message_callback, this, std::placeholders::_1));
209 }
210
211 /**
212  * @brief Enables the failsafe with the specified message
213  *
214  * @param message the message indicating the cause of the failsafe
215  */
216 void enable_failsafe(std::u16string message)
217 {
218     this->failsafe_enabled = true;
219     this->failsafe_request->message = message;
220     auto failsafe_response = this->failsafe_client->async_send_request(this->failsafe_request,
std::bind(&PositionChanger::failsafe_message_callback, this, std::placeholders::_1));
221 }
222
223 /**
224  * @brief applies the collision prevention weights to the current x and y speed if the drone is too close to an object.
225  * It moves the drone away from the object until it is far enough away
226  *
227  */
228 void apply_collision_weights()
229 {
230     if (!this->move_direction_allowed[MOVE_DIRECTION_FRONT])
231     {
232         RCLCPP_INFO(this->get_logger(), "Collision prevention front: %f", collision_prevention_weights[MOVE_DIRECTION_FRONT]);
233         get_x_y_with_rotation(collision_prevention_weights[MOVE_DIRECTION_FRONT], 0, this->current_yaw, &this->current_speed_x,
&this->current_speed_y);
234     }
235     if (!this->move_direction_allowed[MOVE_DIRECTION_BACK])
236     {
237         RCLCPP_INFO(this->get_logger(), "Collision prevention back: %f", collision_prevention_weights[MOVE_DIRECTION_BACK]);
238         get_x_y_with_rotation(collision_prevention_weights[MOVE_DIRECTION_BACK], 0, this->current_yaw, &this->current_speed_x,
&this->current_speed_y);
239     }
240     if (!this->move_direction_allowed[MOVE_DIRECTION_RIGHT])
241     {
242         RCLCPP_INFO(this->get_logger(), "Collision prevention right: %f", collision_prevention_weights[MOVE_DIRECTION_RIGHT]);
243         get_x_y_with_rotation(0, collision_prevention_weights[MOVE_DIRECTION_RIGHT], this->current_yaw, &this->current_speed_x,
&this->current_speed_y);
244     }
245     if (!this->move_direction_allowed[MOVE_DIRECTION_LEFT])
246     {
247         RCLCPP_INFO(this->get_logger(), "Collision prevention left: %f", collision_prevention_weights[MOVE_DIRECTION_LEFT]);
248         get_x_y_with_rotation(0, collision_prevention_weights[MOVE_DIRECTION_LEFT], this->current_yaw, &this->current_speed_x,
&this->current_speed_y);
249     }
250 }
251
252 /**
253  * @brief checks for every direction is an object is too close and if we can move in that direction.
254  * If the object is too close to the drone, calculate the amount we need to move away from it
255  */
256 void check_move_direction_allowed()
257 {
258     this->move_direction_allowed[MOVE_DIRECTION_FRONT] = this->lidar_sensor_values[LIDAR_SENSOR_FR] > MIN_DISTANCE && this-
>lidar_sensor_values[LIDAR_SENSOR_FL] > MIN_DISTANCE;
259     this->move_direction_allowed[MOVE_DIRECTION_LEFT] = this->lidar_sensor_values[LIDAR_SENSOR_FL] > MIN_DISTANCE && this-
>lidar_sensor_values[LIDAR_SENSOR_RL] > MIN_DISTANCE;

```

```

260     this->move_direction_allowed[MOVE_DIRECTION_BACK] = this->lidar_sensor_values[LIDAR_SENSOR_RL] > MIN_DISTANCE && this->lidar_sensor_values[LIDAR_SENSOR_RR] > MIN_DISTANCE;
261     this->move_direction_allowed[MOVE_DIRECTION_RIGHT] = this->lidar_sensor_values[LIDAR_SENSOR_RR] > MIN_DISTANCE && this->lidar_sensor_values[LIDAR_SENSOR_FR] > MIN_DISTANCE;
262
263     // calculate the amount we need to move away from the object to be at the minimum distance
264     collision_prevention_weights[MOVE_DIRECTION_FRONT] = this->move_direction_allowed[MOVE_DIRECTION_FRONT] ? 0 : -(MIN_DISTANCE -
265     std::min(this->lidar_sensor_values[LIDAR_SENSOR_FR], this->lidar_sensor_values[LIDAR_SENSOR_FL]));
266     collision_prevention_weights[MOVE_DIRECTION_LEFT] = this->move_direction_allowed[MOVE_DIRECTION_LEFT] ? 0 : (MIN_DISTANCE -
267     std::min(this->lidar_sensor_values[LIDAR_SENSOR_FL], this->lidar_sensor_values[LIDAR_SENSOR_RL]));
268     collision_prevention_weights[MOVE_DIRECTION_BACK] = this->move_direction_allowed[MOVE_DIRECTION_BACK] ? 0 : (MIN_DISTANCE -
269     std::min(this->lidar_sensor_values[LIDAR_SENSOR_RL], this->lidar_sensor_values[LIDAR_SENSOR_RR]));
270     collision_prevention_weights[MOVE_DIRECTION_RIGHT] = this->move_direction_allowed[MOVE_DIRECTION_RIGHT] ? 0 : -(MIN_DISTANCE -
271     std::min(this->lidar_sensor_values[LIDAR_SENSOR_RR], this->lidar_sensor_values[LIDAR_SENSOR_FR]));
272     apply_collision_weights();
273 }
274
275 /**
276  * @brief Callback for when a set_attitude request is made while landing
277  *
278  * @param future the future object containing the result of the request.
279  */
280 void attitude_land_callback(rclcpp::Client<drone_services::srv::SetAttitude>::SharedFuture future)
281 {
282     auto status = future.wait_for(1s);
283     if (status == std::future_status::ready)
284     {
285         if (future.get()->success)
286         {
287             RCLCPP_INFO(this->get_logger(), "Attitude set to 0 for landing, landing done");
288             this->has_landed = true;
289         }
290     }
291     else
292     {
293         RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
294     }
295 }
296
297 /**
298  * @brief Callback for when a set_vehicle_control request is made while landing
299  *
300  * @param future the future object containing the result of the request.
301  */
302 void vehicle_control_land_request_callback(rclcpp::Client<drone_services::srv::SetVehicleControl>::SharedFuture future)
303 {
304     auto status = future.wait_for(1s);
305     if (status == std::future_status::ready)
306     {
307         if (future.get()->success)
308         {
309             RCLCPP_INFO(this->get_logger(), "Vehicle Control mode set to attitude for landing");
310             this->attitude_request->pitch = 0;
311             this->attitude_request->roll = 0;
312             this->attitude_request->yaw = 0;
313             this->attitude_request->thrust = 0;
314             auto attitude_response = this->attitude_client->async_send_request(this->attitude_request,
315             std::bind(&PositionChanger::attitude_land_callback, this, std::placeholders::_1));
316         }
317     }
318     else
319     {
320         RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
321     }
322 }
323
324 /**
325  * @brief Callback method for receiving new height data
326  *
327  * @param message the message containing the height data
328  */
329 void handle_height_message(const drone_services::msg::HeightData::SharedPtr message)
330 {
331     this->current_height = message->min_height;
332     if (!this->got_ready_drone_request)
333     {
334         this->start_height = message->min_height;
335     }
336     if (this->is_landing)
337     {
338         if (this->current_height <= this->start_height + HEIGHT_DELTA)
339         {
340             this->vehicle_control_request->control = 4;
341             auto control_mode_response = this->vehicle_control_client->async_send_request(this->vehicle_control_request,
342             std::bind(&PositionChanger::vehicle_control_land_request_callback, this, std::placeholders::_1));
343         }
344     }
345 }
346
347 /**
348  * @brief Callback function for receiving new LIDAR data
349  *
350  * @param message the message containing the LIDAR data
351  */

```

```

351 void handle_lidar_message(const drone_services::msg::LidarReading::SharedPtr message)
352 {
353     this->has_received_first_lidar_message = true;
354
355     this->received_lidar_message = true;
356     if (message->sensor_1 > 0)
357     {
358         this->lidar_sensor_values[LIDAR_SENSOR_FR] = message->sensor_1;
359     }
360     if (message->sensor_2 > 0)
361     {
362         this->lidar_sensor_values[LIDAR_SENSOR_FL] = message->sensor_2;
363     }
364     if (message->sensor_3 > 0)
365     {
366         this->lidar_sensor_values[LIDAR_SENSOR_RL] = message->sensor_3;
367     }
368     if (message->sensor_4 > 0)
369     {
370         this->lidar_sensor_values[LIDAR_SENSOR_RR] = message->sensor_4;
371     }
372
373     for (int i = 0; i < 4; i++)
374     {
375         this->lidar_imu_readings[i] = message->imu_data[i];
376     }
377
378     check_move_direction_allowed();
379 }
380
381 /**
382  * @brief Checks if the LIDAR is still sending messages. If not, enable failsafe
383  */
384 void check_lidar_health()
385 {
386     if (this->has_received_first_lidar_message)
387     {
388         if (!this->received_lidar_message)
389         {
390             RCLCPP_WARN(this->get_logger(), "Lidar not sending messages, enabling failsafe");
391             enable_failsafe(u"No healthy connection to LIDAR! Check the LIDAR USB cable and restart the drone.");
392         }
393     }
394     this->received_lidar_message = false;
395 }
396
397 /**
398  * @brief Callback function for receiving the odometry data from the drone.
399  * This is used to get the current yaw angle before the drone takes off,
400  * to make sure it doesn't want to spin to a certain angle while taking off.
401  * @param message the message containing the odometry data
402  */
403 void handle_odometry_message(const px4_msgs::msg::VehicleOdometry::SharedPtr message)
404 {
405     Quaternion q = {message->q[0], message->q[1], message->q[2], message->q[3]};
406     this->current_yaw = get_yaw_angle(q);
407 }
408
409 /**
410  * @brief Callback function for when the /drone/move_position service is called
411  * @param request_header the header of the request
412  * @param request the request containing the new parameters to move to
413  * @param response the response to send back. true if the request was successful, false otherwise
414  */
415 void handle_move_position_request(
416     const std::shared_ptr<rmw_request_id_t> request_header,
417     const std::shared_ptr<drone_services::srv::MovePosition::Request> request,
418     const std::shared_ptr<drone_services::srv::MovePosition::Response> response)
419 {
420     if (!this->failsafe_enabled)
421     {
422         if (!this->has_received_first_lidar_message)
423         {
424             this->enable_failsafe(u"Waiting for LIDAR timed out! Check the LIDAR USB connection and consider restarting the
425 drone.");
426             return;
427         }
428         RCLCPP_INFO(this->get_logger(), "Incoming request\nfront_back: %f\nleft_right: %f\nup_down: %f\nangle: %f", request-
429 >front_back, request->left_right, request->up_down, request->angle);
430         if (request->angle > 360 || request->angle < -360)
431         {
432             RCLCPP_ERROR(this->get_logger(), "Angle is not in range [-360, 360]");
433             response->success = false;
434             return;
435         }
436         if (this->vehicle_control_request->control != DEFAULT_CONTROL_MODE)
437         {
438             this->vehicle_control_request->control = DEFAULT_CONTROL_MODE;
439             auto control_mode_response = this->vehicle_control_client->async_send_request(this->vehicle_control_request,
440 std::bind(&PositionChanger::vehicle_control_service_callback, this, std::placeholders::_1));
441             this->current_yaw += request->angle * (M_PI / 180.0); // get the angle in radians
442             this->current_speed_z = request->up_down;
443             get_x_y_with_rotation(request->front_back, request->left_right, this->current_yaw, &this->current_speed_x, &this-
444 >current_speed_y);

```

```

446         check_move_direction_allowed();
447         RCLCPP_INFO(this->get_logger(), "Calculated speed x: %f, y: %f", this->current_speed_x, this->current_speed_y);
448         send_trajectory_message();
449         response->success = true;
450     }
451     else
452     {
453         response->success = false;
454     }
455 }
456
457 /**
458  * @brief Callback function for when the /drone/land service is called
459  *
460  * @param request_header the header of the request
461  * @param request the request to land
462  * @param response the response to send back.
463  */
464 void handle_land_request(
465     const std::shared_ptr<rmw_request_id_t> request_header,
466     const std::shared_ptr<drone_services::srv::Land::Request> request,
467     const std::shared_ptr<drone_services::srv::Land::Response> response)
468 {
469     this->is_landing = true;
470     response->is_landing = this->is_landing;
471 }
472
473 /**
474  * @brief Callback function for when the /drone/ready_drone service is called
475  *
476  * @param request_header the header of the request
477  * @param request the request to ready the drone
478  * @param response the response to send back. Contains true if the request was successful, false otherwise
479  */
480 void handle_ready_drone_request(
481     const std::shared_ptr<rmw_request_id_t> request_header,
482     const std::shared_ptr<drone_services::srv::ReadyDrone::Request> request,
483     const std::shared_ptr<drone_services::srv::ReadyDrone::Response> response)
484 {
485     if (this->failsafe_enabled)
486     {
487         response->success = false;
488         return;
489     }
490     if (!this->has_received_first_lidar_message)
491     {
492         this->enable_failsafe(u"Waiting for LIDAR timed out! Check the LIDAR USB connection and consider restarting the
493 drone.");
494         response->success = false;
495         return;
496     }
497     this->got_ready_drone_request = true;
498     this->vehicle_control_request->control = CONTROL_MODE_ATTITUDE;
499     auto control_mode_response = this->vehicle_control_client->async_send_request(this->vehicle_control_request,
500 std::bind(&PositionChanger::vehicle_control_service_callback, this, std::placeholders::_1));
501     response->success = true;
502 }
503
504 /**
505  * @brief Get the yaw angle from a specified normalized quaternion.
506  * Uses the theory from https://en.wikipedia.org/wiki/Conversion\_between\_quaternions\_and\_Euler\_angles
507  *
508  * @param q the quaternion that holds the rotation
509  * @return the yaw angle in radians
510  */
511 static float get_yaw_angle(Quaternion q)
512 {
513     float siny_cosp = 2 * (q.w * q.z + q.x * q.y);
514     float cosy_cosp = 1 - 2 * (q.y * q.y + q.z * q.z);
515     return std::atan2(siny_cosp, cosy_cosp);
516 }
517
518 /**
519  * @brief Get the new x and y coordinates after a rotation of yaw radians.
520  * Uses the standard rotation matrix: https://en.wikipedia.org/wiki/Rotation\_matrix
521  *
522  * @param x the original x coordinate
523  * @param y the original y coordinate
524  * @param yaw the angle to rotate by in radians
525  * @param x_out the resulting x coordinate
526  * @param y_out the resulting y coordinate
527  */
528 static void get_x_y_with_rotation(float x, float y, float yaw, float *x_res, float *y_res)
529 {
530     *x_res = x * std::cos(yaw) - y * std::sin(yaw);
531     *y_res = x * std::sin(yaw) + y * std::cos(yaw);
532 }
533
534 private:
535 rclcpp::Publisher<drone_services::msg::FailsafeMsg>::SharedPtr failsafe_publisher;
536 rclcpp::Subscription<drone_services::msg::LidarReading>::SharedPtr lidar_subscription;
537 rclcpp::Subscription<px4_msgs::msg::VehicleOdometry>::SharedPtr odometry_subscription;
538 rclcpp::Subscription<drone_services::msg::HeightData>::SharedPtr height_subscription;
539
540 rclcpp::Client<drone_services::srv::SetTrajectory>::SharedPtr trajectory_client;
541 rclcpp::Client<drone_services::srv::SetAttitude>::SharedPtr attitude_client;
542 rclcpp::Client<drone_services::srv::SetVehicleControl>::SharedPtr vehicle_control_client;
543 rclcpp::Client<drone_services::srv::EnableFailsafe>::SharedPtr failsafe_client;

```

```

543 rclcpp::Client<drone_services::srv::ArmDrone>::SharedPtr arm_drone_client;
544
545 rclcpp::Service<drone_services::srv::MovePosition>::SharedPtr move_position_service;
546 rclcpp::Service<drone_services::srv::ReadyDrone>::SharedPtr ready_drone_service;
547 rclcpp::Service<drone_services::srv::Land>::SharedPtr land_service;
548 rclcpp::TimerBase::SharedPtr lidar_health_timer;
549
550 drone_services::srv::SetTrajectory::Request::SharedPtr trajectory_request;
551 drone_services::srv::SetAttitude::Request::SharedPtr attitude_request;
552 drone_services::srv::SetVehicleControl::Request::SharedPtr vehicle_control_request;
553 drone_services::srv::EnableFailsafe::Request::SharedPtr failsafe_request;
554 drone_services::srv::ArmDrone::Request::SharedPtr arm_drone_request;
555
556 float lidar_sensor_values[4]{MIN_DISTANCE}; // last distance measured by the lidar sensors
557 float lidar_imu_readings[4]; // last imu readings from the lidar sensors
558 float current_yaw = 0; // in radians
559 float current_speed_x = 0;
560 float current_speed_y = 0;
561 float current_speed_z = 0;
562 float current_height = 0;
563 float start_height = -1;
564 bool is_landing = false;
565 bool has_landed = false;
566 bool move_direction_allowed[4] = {true}; // whether the drone can move in a certain direction
567 float collision_prevention_weights[4] = {0}; // the amount to move away from an object in a certain direction if the drone is
too close
568 bool failsafe_enabled = false;
569 bool received_lidar_message = false;
570 int lidar_health_checks = 0;
571 bool has_received_first_lidar_message = false;
572 bool got_ready_drone_request = false;
573
574 /**
575  * @brief waits for a service to be available
576  *
577  * @tparam T an rclcpp::Client object
578  * @param client the client object to wait for the service
579  */
580 template <class T>
581 void wait_for_service(T client, std::string service_name)
582 {
583     while (!client->wait_for_service(1s))
584     {
585         if (!rclcpp::ok())
586         {
587             RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the service. Exiting.");
588             return;
589         }
590         RCLCPP_INFO(this->get_logger(), "service not available, waiting again on service %s", service_name.c_str());
591     }
592 }
593 };
594
595 int main(int argc, char *argv[])
596 {
597     rclcpp::init(argc, argv);
598     rclcpp::spin(std::make_shared<PositionChanger>());
599     rclcpp::shutdown();
600     return 0;
601 }

```

A.2: PX4Controller node

Deze node stuurt de flight controller aan via de ROS 2 PX4 Bridge. Deze biedt de `drone/set_trajectory` service aan om de drone te laten bewegen en zorgt dat de flight controller deze commando's kan ontvangen. Ook biedt deze de `drone/arm` en `drone/disarm` service aan om de drone te armen en te disarmen. Deze zorgt ook dat de drone niet meer kan bewegen als de failsafe aangezet wordt.

```
1  #include <chrono>
2  #include <math.h>
3
4  #include "rclcpp/rclcpp.hpp"
5
6  #include <px4_msgs/msg/vehicle_attitude_setpoint.hpp>
7  #include <px4_msgs/msg/trajectory_setpoint.hpp>
8  #include <px4_msgs/msg/vehicle_command.hpp>
9  #include <px4_msgs/msg/vehicle_control_mode.hpp>
10 #include <px4_msgs/msg/vehicle_attitude.hpp>
11 #include <px4_msgs/msg/vehicle_local_position.hpp>
12
13 #include <drone_services/srv/set_attitude.hpp>
14 #include <drone_services/srv/set_trajectory.hpp>
15 #include <drone_services/srv/arm_drone.hpp>
16 #include <drone_services/srv/disarm_drone.hpp>
17 #include <drone_services/msg/drone_control_mode.hpp>
18 #include <drone_services/msg/drone_arm_status.hpp>
19 #include <drone_services/msg/failsafe_msg.hpp>
20
21 #include <std_srvs/srv/empty.hpp>
22
23 #include "drone_control_modes.h"
24
25 #define DEFAULT_YAW_SPEED 0.6 // default turning speed in radians per second
26 #define D_SPEED(x) -x - 9.81 // a speed up of x m/s has to be adjusted for the earths gravity
27
28 using namespace std::chrono_literals;
29
30 class PX4Controller : public rclcpp::Node
31 {
32 public:
33     PX4Controller() : Node("px4_controller")
34     {
35         rmw_qos_profile_t qos_profile = rmw_qos_profile_sensor_data;
36         auto qos = rclcpp::QoS(rclcpp::QoSInitialization(qos_profile.history, 5), qos_profile);
37
38         // publishers for controlling the drone
39         vehicle_setpoint_publisher_ = this->create_publisher<px4_msgs::msg::VehicleAttitudeSetpoint>("/fmu/in/vehicle_attitude_setpoint", 10);
40         vehicle_command_publisher_ = this->create_publisher<px4_msgs::msg::VehicleCommand>("/fmu/in/vehicle_command", 10);
41         trajectory_setpoint_publisher_ = this->create_publisher<px4_msgs::msg::TrajectorySetpoint>("/fmu/in/trajectory_setpoint",
42         10);
43         // publisher for publishing if the drone is armed
44         arm_status_publisher_ = this->create_publisher<drone_services::msg::DroneArmStatus>("/drone/arm_status", 10);
45
46         // subscriptions from the Pixhawk
47         vehicle_attitude_subscription_ = this->create_subscription<px4_msgs::msg::VehicleAttitude>("/fmu/out/vehicle_attitude", qos,
48         std::bind(&PX4Controller::on_attitude_receive, this, std::placeholders::_1));
49         vehicle_local_position_subscription_ = this->create_subscription<px4_msgs::msg::VehicleLocalPosition>("/fmu/out/vehicle_local_position", qos,
50         std::bind(&PX4Controller::on_local_position_receive, this, std::placeholders::_1));
51         // subscription on receiving a new control mode
52         control_mode_subscription_ = this->create_subscription<drone_services::msg::DroneControlMode>("/drone/control_mode", qos,
53         std::bind(&PX4Controller::on_control_mode_receive, this, std::placeholders::_1));
54         failsafe_subscription_ = this->create_subscription<drone_services::msg::FailsafeMsg>("/drone/failsafe", qos,
55         std::bind(&PX4Controller::on_failsafe_receive, this, std::placeholders::_1));
56         // services for controlling the drone
57         set_attitude_service_ = this->create_service<drone_services::srv::SetAttitude>("/drone/set_attitude",
58         std::bind(&PX4Controller::handle_attitude_setpoint, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
59         set_trajectory_service_ = this->create_service<drone_services::srv::SetTrajectory>("/drone/set_trajectory",
60         std::bind(&PX4Controller::handle_trajectory_setpoint, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
61         disarm_service_ = this->create_service<drone_services::srv::DisarmDrone>("/drone/disarm",
62         std::bind(&PX4Controller::handle_disarm_request, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
63         arm_service_ = this->create_service<drone_services::srv::ArmDrone>("/drone/arm",
64         std::bind(&PX4Controller::handle_arm_request, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
65         // create timer to send setpoints every 100 milliseconds
66         timer_ = this->create_wall_timer(100ms, std::bind(&PX4Controller::send_setpoint, this));
67
68         start_time_ = this->get_clock()->now().seconds();
69         RCLCPP_INFO(this->get_logger(), "Finished initializing at %f seconds.", start_time_);
70     }
71
72 private:
73     rclcpp::Publisher<px4_msgs::msg::VehicleAttitudeSetpoint>::SharedPtr vehicle_setpoint_publisher_;
74     rclcpp::Publisher<px4_msgs::msg::TrajectorySetpoint>::SharedPtr trajectory_setpoint_publisher_;
75     rclcpp::Publisher<px4_msgs::msg::VehicleCommand>::SharedPtr vehicle_command_publisher_;
76     rclcpp::Publisher<drone_services::msg::DroneArmStatus>::SharedPtr arm_status_publisher_;
77
78     rclcpp::Subscription<px4_msgs::msg::VehicleAttitude>::SharedPtr vehicle_attitude_subscription_;
79     rclcpp::Subscription<px4_msgs::msg::VehicleLocalPosition>::SharedPtr vehicle_local_position_subscription_;
80     rclcpp::Subscription<drone_services::msg::DroneControlMode>::SharedPtr control_mode_subscription_;
81     rclcpp::Subscription<drone_services::msg::FailsafeMsg>::SharedPtr failsafe_subscription_;
```

```

74 rclcpp::Service<drone_services::srv::SetAttitude>::SharedPtr set_attitude_service_;
75 rclcpp::Service<drone_services::srv::SetTrajectory>::SharedPtr set_trajectory_service_;
76 rclcpp::Service<drone_services::srv::DisarmDrone>::SharedPtr disarm_service_;
77 rclcpp::Service<drone_services::srv::ArmDrone>::SharedPtr arm_service_;
78
79 rclcpp::TimerBase::SharedPtr timer_;
80 double start_time_; // time when the node was started
81 bool user_in_control = false; // if user has taken over control
82 bool armed = false; // if the drone is armed
83 bool ready_to_fly = false; // if the drone is ready to fly
84 bool new_setpoint = false; // for printing new q_d when a new setpoint has been received
85
86 float last_setpoint[3] = {0, 0, 0}; // yaw, pitch, roll
87 float last_angle = 0; // angle in radians (-PI .. PI)
88 float last_thrust = 0; // default 10% thrust for when the drone gets armed
89
90 float velocity[3] = {0, 0, 0}; // velocity setpoint [x,y,z]
91 float position[3] = {0, 0, 0}; // position setpoint [x,y,z]
92
93 float base_q[4] = {0, 0, 0, 0}; // base local position quaternion
94 int base_q_amount = 0; // amount of base quaternions received
95
96 char current_control_mode = CONTROL_MODE_ATTITUDE; // start with attitude control
97
98 const float omega = 0.3; // angular speed of the POLAR trajectory
99 const float K = 0; // [m] gain that regulates the spiral pitch
100
101 float rho_0 = 0; // initial rho of polar coordinate
102 float theta_0 = 0; // initial theta of polar coordinate
103 float p0_z = -0.0; // initial height
104 float des_x = 0.0, des_y = 0.0, des_z = p0_z; // desired position
105 float gamma = M_PI_4; // desired heading direction
106
107 float local_x = 0; // local position x
108 float local_y = 0; // local position y
109
110 bool failsafe_enabled = false;
111
112 std::array<float, 4> q = {0, 0, 0, 0}; // result quaternion
113
114 /**
115  * @brief handles reception of attitude setpoints through service
116  *
117  * @param request_header the header of the request
118  * @param request the request containing the new attitude setpoint
119  * @param response the response indicating if the setpoint was changed successfully
120  */
121 void handle_attitude_setpoint(
122     const std::shared_ptr<rmw_request_id_t> request_header,
123     const std::shared_ptr<drone_services::srv::SetAttitude::Request> request,
124     const std::shared_ptr<drone_services::srv::SetAttitude::Response> response)
125 {
126     if (this->failsafe_enabled)
127     {
128         RCLCPP_INFO(this->get_logger(), "Failsafe enabled, ignoring attitude setpoint");
129         response->success = false;
130         return;
131     }
132     if (armed)
133     {
134         if (request->yaw == 0 && request->pitch == 0 && request->roll == 0 && request->thrust == 0)
135         {
136             last_setpoint[0] = degrees_to_radians(request->yaw);
137             last_setpoint[1] = degrees_to_radians(request->pitch);
138             last_setpoint[2] = degrees_to_radians(request->roll);
139             last_thrust = request->thrust;
140             RCLCPP_INFO(this->get_logger(), "STOPPING MOTORS");
141         }
142         else
143         {
144             last_setpoint[0] += degrees_to_radians(request->yaw);
145             last_setpoint[1] += degrees_to_radians(request->pitch);
146             last_setpoint[2] += degrees_to_radians(request->roll);
147             last_thrust += request->thrust;
148             if (last_thrust > 1)
149                 last_thrust = 1;
150             else if (last_thrust < 0)
151                 last_thrust = 0;
152         }
153
154         RCLCPP_INFO(this->get_logger(), "got values: yaw:%f pitch:%f roll:%f thrust:%f", request->yaw, request->pitch, request->roll, request->thrust);
155         RCLCPP_INFO(this->get_logger(), "New setpoint: yaw:%f pitch:%f roll:%f thrust:%f", last_setpoint[0], last_setpoint[1], last_setpoint[2], last_thrust);
156
157         response->success = true;
158     }
159     else
160     {
161         last_thrust = 0;
162         last_setpoint[0] = 0;
163         last_setpoint[1] = 0;
164         last_setpoint[2] = 0;
165         response->success = false;
166     }
167 }
168
169 /**
170  * @brief handles reception of trajectory setpoints through service

```

```

171 *
172 * @param request_header the header of the request
173 * @param request the request containing the new trajectory setpoint (velocity or position)
174 * @param response the response indicating if the setpoint was changed successfully
175 */
176 void handle_trajectory_setpoint(
177     const std::shared_ptr<rmw_request_id_t> request_header,
178     const std::shared_ptr<drone_services::srv::SetTrajectory::Request> request,
179     const std::shared_ptr<drone_services::srv::SetTrajectory::Response> response)
180 {
181     if (this->failsafe_enabled)
182     {
183         RCLCPP_INFO(this->get_logger(), "Failsafe enabled, ignoring trajectory setpoint");
184         response->success = false;
185         return;
186     }
187     if (!(request->control_mode == CONTROL_MODE_VELOCITY || request->control_mode == CONTROL_MODE_POSITION))
188     {
189         RCLCPP_INFO(this->get_logger(), "Got invalid trajectory control mode: %d", request->control_mode);
190         response->success = false;
191     }
192     else
193     {
194         RCLCPP_INFO(this->get_logger(), "Got new trajectory setpoint with control mode: %d", request->control_mode);
195         if (request->control_mode == CONTROL_MODE_VELOCITY)
196         {
197             for (int i = 0; i < 3; i++)
198             {
199                 velocity[i] = request->values[i];
200             }
201             RCLCPP_INFO(this->get_logger(), "Got new velocity setpoint. %f %f %f", velocity[0], velocity[1], velocity[2]);
202         }
203         else if (request->control_mode == CONTROL_MODE_POSITION)
204         {
205             position[0] += request->values[0];
206             position[1] += request->values[1];
207             position[2] -= request->values[2]; // height is negative
208
209             RCLCPP_INFO(this->get_logger(), "Got new position setpoint: %f %f %f", position[0], position[1], position[2]);
210         }
211
212         last_angle = request->yaw;
213         new_setpoint = true;
214         RCLCPP_INFO(this->get_logger(), "Yaw: %f", last_angle);
215         response->success = true;
216     }
217 }
218
219 /**
220 * @brief disarms the drone when a call to the disarm service is made
221 *
222 * @param request_header the header for the service request
223 * @param request the request (empty)
224 * @param response the response (empty)
225 */
226 void handle_disarm_request(
227     const std::shared_ptr<rmw_request_id_t> request_header,
228     const std::shared_ptr<drone_services::srv::DisarmDrone::Request> request,
229     const std::shared_ptr<drone_services::srv::DisarmDrone::Response> response)
230 {
231     RCLCPP_INFO(this->get_logger(), "Got disarm request...");
232
233     if (armed)
234     {
235         armed = false;
236         user_in_control = false;
237         publish_vehicle_command(px4_msgs::msg::VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM, 0.0, 0);
238
239         RCLCPP_INFO(this->get_logger(), "Publishing disarm status message");
240         auto msg = drone_services::msg::DroneArmStatus();
241         msg.armed = false;
242         arm_status_publisher->publish(msg);
243
244         response->success = true;
245     }
246     else
247     {
248         RCLCPP_ERROR(this->get_logger(), "Got disarm request but drone was not armed!");
249         response->success = false;
250     }
251 }
252
253 /**
254 * @brief arms the drone when a call to the arm service is made
255 *
256 * @param request_header the header for the service request
257 * @param request the request (empty)
258 * @param response the response (empty)
259 */
260 void handle_arm_request(
261     const std::shared_ptr<rmw_request_id_t> request_header,
262     const std::shared_ptr<drone_services::srv::ArmDrone::Request> request,
263     const std::shared_ptr<drone_services::srv::ArmDrone::Response> response)
264 {
265     RCLCPP_INFO(this->get_logger(), "Got arm request...");
266     if (this->failsafe_enabled)
267     {
268         response->success = false;
269         return;

```

```

270     }
271     if (!armed)
272     {
273         this->publish_vehicle_command(px4_msgs::msg::VehicleCommand::VEHICLE_CMD_DO_SET_MODE, 1, 6);
274         RCLCPP_INFO(this->get_logger(), "Set to offboard mode");
275         // arm the drone
276         this->publish_vehicle_command(px4_msgs::msg::VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM, 1.0, 0);
277
278         RCLCPP_INFO(this->get_logger(), "Arm command sent");
279         this->last_thrust = -0.1; // spin motors at 10% thrust
280         armed = true;
281
282         RCLCPP_INFO(this->get_logger(), "Publishing arm status message");
283         auto msg = drone_services::msg::DroneArmStatus();
284         msg.armed = true;
285         arm_status_publisher->publish(msg);
286
287         response->success = true;
288     }
289     else
290     {
291         RCLCPP_ERROR(this->get_logger(), "Arm request received but the drone is already armed!");
292         response->success = false;
293     }
294 }
295
296 /**
297  * @brief sends the latest received attitude setpoint to the drone
298  *
299  */
300 void send_attitude_setpoint()
301 {
302     // set message to enable attitude
303     auto msg = px4_msgs::msg::VehicleAttitudeSetpoint();
304
305     // move up
306     msg.thrust_body[0] = 0; // north
307     msg.thrust_body[1] = 0; // east
308     msg.thrust_body[2] = -last_thrust;
309
310     calculate_quaternion(q, last_setpoint[0], last_setpoint[1], last_setpoint[2]);
311     // set quaternion
312     msg.q_d[0] = q.at(0);
313     msg.q_d[1] = base_q[1] + q.at(1);
314     msg.q_d[2] = base_q[2] + q.at(2);
315     msg.q_d[3] = base_q[3] + q.at(3);
316
317     msg.yaw_sp_move_rate = 0;
318     msg.reset_integral = false;
319     msg.fw_control_yaw_wheel = false;
320
321     msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
322     vehicle_setpoint_publisher->publish(msg);
323 }
324
325 /**
326  * @brief sends the latest received velocity setpoint to the drone
327  *
328  */
329 void send_velocity_setpoint()
330 {
331     auto msg = px4_msgs::msg::TrajectorySetpoint();
332
333     msg.velocity[0] = velocity[0];
334     msg.velocity[1] = velocity[1];
335     msg.velocity[2] = -velocity[2];
336     // set no position control
337     for (int i = 0; i < 3; i++)
338     {
339         msg.position[i] = NAN;
340     }
341     publish_trajectory_setpoint(msg);
342 }
343
344 /**
345  * @brief sends the latest received position setpoint to the drone
346  *
347  */
348 void send_position_setpoint()
349 {
350     auto msg = px4_msgs::msg::TrajectorySetpoint();
351
352     RCLCPP_INFO(this->get_logger(), "Sending position setpoint: %f %f %f", des_x, des_y, des_z);
353     RCLCPP_INFO(this->get_logger(), "local position: %f %f", local_x, local_y);
354     msg.position = {local_x, local_y, des_z};
355     msg.yaw = gamma; // -3.14; // [-PI:PI]
356
357     msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
358     trajectory_setpoint_publisher->publish(msg);
359 }
360
361 /**
362  * @brief publishes a trajectory setpoint
363  *
364  * @param msg the message containing the trajectory setpoint
365  */
366 void publish_trajectory_setpoint(px4_msgs::msg::TrajectorySetpoint msg)
367 {
368

```

```

369     msg.yaw = last_angle;
370     msg.yawspeed = DEFAULT_YAW_SPEED;
371     msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
372     trajectory_setpoint_publisher->publish(msg);
373 }
374
375 /**
376  * @brief Send setpoints to PX4.
377  *
378  */
379 void send_setpoint()
380 {
381     if (failsafe_enabled)
382     {
383         // immediately land if failsafe enabled
384         RCLCPP_INFO(this->get_logger(), "Failsafe enabled, landing...");
385         this->current_control_mode = CONTROL_MODE_VELOCITY;
386         this->velocity[0] = 0;
387         this->velocity[1] = 0;
388         this->velocity[2] = -0.1; // move down slowly
389         send_velocity_setpoint();
390         return;
391     }
392     float theta = theta_0 + omega * 0.1;
393     float rho = rho_0 + K * theta;
394
395     // from polar to cartesian coordinates
396     des_x = rho * cos(theta);
397     des_y = rho * sin(theta);
398     des_z = position[2]; // the z position can be set to the received height
399
400     if (!user_in_control)
401     {
402         send_attitude_setpoint();
403     }
404     else
405     {
406         if (current_control_mode == CONTROL_MODE_ATTITUDE)
407         {
408             send_attitude_setpoint();
409         }
410         else
411         {
412             if (!new_setpoint)
413             {
414                 return;
415             }
416             if (current_control_mode == CONTROL_MODE_VELOCITY)
417             {
418                 send_velocity_setpoint();
419             }
420             else if (current_control_mode == CONTROL_MODE_POSITION)
421             {
422                 send_position_setpoint();
423             }
424             new_setpoint = false;
425         }
426     }
427 }
428
429 /**
430  * @brief Callback function for the attitude topic.
431  * This function is called every time a new message is received on the topic.
432  * It only stores the first two measurements while the drone is not armed.
433  *
434  * @param msg the message containing the attitude
435  */
436 void on_attitude_receive(const px4_msgs::msg::VehicleAttitude::SharedPtr msg)
437 {
438     if (!armed)
439     {
440         if (base_q_amount > 2)
441         {
442             base_q[1] = (base_q[1] + msg->q[1]) / 2;
443             base_q[2] = (base_q[2] + msg->q[2]) / 2;
444             base_q[3] = (base_q[3] + msg->q[3]) / 2;
445         }
446         else
447         {
448             base_q[1] = msg->q[1];
449             base_q[2] = msg->q[2];
450             base_q[3] = msg->q[3];
451             base_q_amount++;
452         }
453     }
454 }
455
456 /**
457  * @brief Callback function for receiving the local position from the Pixhawk
458  * The local position is only received while the drone is not yet flying (user has not yet taken over control)
459  *
460  * @param msg the message containing the local position
461  */
462 void on_local_position_receive(const px4_msgs::msg::VehicleLocalPosition::SharedPtr msg)
463 {
464     // set start values to current position
465     if (!user_in_control)
466     {
467         // https://www.math.usm.edu/lambers/mat169/fall09/lecture33.pdf

```

```

468         rho_0 = pow(msg->x, 2) + pow(msg->y, 2);
469         theta_0 = atan2(msg->y, msg->x);
470         last_angle = msg->heading;
471
472         local_x = msg->x;
473         local_y = msg->y;
474     }
475 }
476
477 void on_control_mode_receive(const drone_services::msg::DroneControlMode::SharedPtr msg)
478 {
479     if (msg->control_mode >= CONTROL_MODE_MIN && msg->control_mode <= CONTROL_MODE_MAX)
480     {
481         current_control_mode = msg->control_mode;
482         RCLCPP_INFO(this->get_logger(), "Got valid control mode");
483         user_in_control = true; // user has taken over control
484     }
485     else
486     {
487         RCLCPP_ERROR(this->get_logger(), "Received invalid control mode %d", msg->control_mode);
488     }
489     RCLCPP_INFO(this->get_logger(), "Control mode set to %d", current_control_mode);
490 }
491
492 /**
493  * @brief Callback function for receiving failsafe messages
494  *
495  * @param msg the message indicating that the failsafe was enabled
496  */
497 void on_failsafe_receive(const drone_services::msg::FailsafeMsg::SharedPtr msg)
498 {
499     if (msg->enabled)
500     {
501         this->failsafe_enabled = true;
502     }
503 }
504
505 /**
506  * @brief Publish vehicle commands
507  * @param command Command code (matches VehicleCommand and MAVLink MAV_CMD codes)
508  * @param param1 Command parameter 1
509  * @param param2 Command parameter 2
510  */
511 void publish_vehicle_command(uint16_t command, float param1, float param2)
512 {
513     px4_msgs::msg::VehicleCommand msg{};
514     msg.param1 = param1;
515     msg.param2 = param2;
516     msg.command = command;
517     msg.target_system = 1;
518     msg.target_component = 1;
519     msg.source_system = 1;
520     msg.source_component = 1;
521     msg.from_external = true;
522     msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
523     vehicle_command_publisher->publish(msg);
524 }
525
526 /**
527  * @brief calculates a quaternion based on the given input values
528  *
529  * @param ptr array of result quaternion. Must be of size 4.
530  * @param heading desired heading (yaw) in radians.
531  * @param attitude desired attitude (pitch) in radians.
532  * @param bank desired bank (roll) in radians.
533  */
534 static void calculate_quaternion(std::array<float, 4> &ptr, float heading, float attitude, float bank)
535 {
536     float c1 = cos(heading / 2);
537     float c2 = cos(attitude / 2);
538     float c3 = cos(bank / 2);
539     float s1 = sin(heading / 2);
540     float s2 = sin(attitude / 2);
541     float s3 = sin(bank / 2);
542
543     float c1c2 = c1 * c2;
544     float s1s2 = s1 * s2;
545
546     ptr.at(0) = c1c2 * c3 - s1s2 * s3; // w
547     ptr.at(1) = c1c2 * s3 + s1s2 * c3; // x
548     ptr.at(2) = s1 * c2 * c3 + c1 * s2 * s3; // y
549     ptr.at(3) = c1 * s2 * c3 - s1 * c2 * s3; // z
550 }
551
552 /**
553  * @brief converts degrees to radians
554  *
555  * @param deg angle in degrees
556  * @return float new angle in radians
557  */
558 static float degrees_to_radians(float deg)
559 {
560     return deg * (M_PI / 180.0);
561 }
562
563 };
564
565 int main(int argc, char *argv[])
566 {

```

```
567     rclcpp::init(argc, argv);
568
569     rclcpp::Node::SharedPtr node = std::make_shared<PX4Controller>();
570     rclcpp::spin(node);
571     rclcpp::shutdown();
572     return 0;
573 }
```

A.3: Heartbeat node

Deze node stuurt *OffboardControl* berichten naar de flight controller en zorgt ervoor dat de verbinding met de flight controller in stand blijft. Deze biedt ook de service

`drone/set_vehicle_control` aan om de mate van besturing van de drone aan te passen.

```
1  #include <chrono>
2  #include "rclcpp/rclcpp.hpp"
3  #include <px4_msgs/msg/offboard_control_mode.hpp>
4  #include <drone_services/srv/set_vehicle_control.hpp>
5  #include <drone_services/msg/drone_control_mode.hpp>
6
7  #include "drone_control_modes.h"
8
9  using namespace std::chrono_literals;
10
11 class HeartBeat : public rclcpp::Node
12 {
13 public:
14     HeartBeat() : Node("heartbeat")
15     {
16         vehicle_control_mode_service_ = this->create_service<drone_services::srv::SetVehicleControl>("/drone/set_vehicle_control",
17 std::bind(&HeartBeat::handle_vehicle_control_change, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
18         RCLCPP_INFO(this->get_logger(), "initialized service /drone/set_vehicle_control");
19         // create a publisher on the drone control mode topic
20         drone_control_mode_publisher_ = this->create_publisher<drone_services::msg::DroneControlMode>("/drone/control_mode", 10);
21         // create a publisher on the offboard control mode topic
22         offboard_control_mode_publisher_ = this->create_publisher<px4_msgs::msg::OffboardControlMode>("/fmu/in/offboard_control_mode", 10);
23         // create timer to send heartbeat messages (offboard control) every 100ms
24         timer_ = this->create_wall_timer(10ms, std::bind(&HeartBeat::send_heartbeat, this));
25         start_time = this->get_clock()->now().seconds();
26         RCLCPP_INFO(this->get_logger(), "done initializing at %d seconds. Sending heartbeat...", start_time);
27     }
28 private:
29     // publisher for offboard control mode messages
30     rclcpp::Publisher<px4_msgs::msg::OffboardControlMode>::SharedPtr offboard_control_mode_publisher_;
31     // publisher for the control mode of the drone
32     rclcpp::Publisher<drone_services::msg::DroneControlMode>::SharedPtr drone_control_mode_publisher_;
33     // timer for sending the heartbeat
34     rclcpp::TimerBase::SharedPtr timer_;
35     // service to change vehicle mode
36     rclcpp::Service<drone_services::srv::SetVehicleControl>::SharedPtr vehicle_control_mode_service_;
37
38     // start time of node in seconds
39     double start_time;
40     // which level of control is needed, only attitude control by default
41     int control_mode = 1 << CONTROL_ATTITUDE_POS;
42     /**
43      * @brief Publish offboard control mode messages as a heartbeat.
44      * Only the attitude is enabled, because that is how the drone will be controlled.
45      */
46     void send_heartbeat()
47     {
48         {
49             auto msg = px4_msgs::msg::OffboardControlMode();
50
51             msg.position = (this->control_mode >> CONTROL_POSITION_POS) & 1 ? true : false;
52             msg.velocity = (this->control_mode >> CONTROL_VELOCITY_POS) & 1 ? true : false;
53             msg.acceleration = (this->control_mode >> CONTROL_ACCELERATION_POS) & 1 ? true : false;
54             msg.attitude = (this->control_mode >> CONTROL_ATTITUDE_POS) & 1 ? true : false;
55             msg.body_rate = (this->control_mode >> CONTROL_BODY_RATE_POS) & 1 ? true : false;
56             msg.actuator = (this->control_mode >> CONTROL_ACTUATOR_POS) & 1 ? true : false;
57             // get timestamp and publish message
58             msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
59             offboard_control_mode_publisher_->publish(msg);
60         }
61     }
62     /**
63      * @brief Publish the current control mode of the drone onto the 'drone/control_mode' topic
64      */
65     void publish_new_control_mode()
66     {
67         {
68             auto msg = drone_services::msg::DroneControlMode();
69             if (this->control_mode & (1 << CONTROL_ATTITUDE_POS))
70             {
71                 msg.control_mode = CONTROL_MODE_ATTITUDE;
72                 RCLCPP_INFO(this->get_logger(), "set control mode to attitude");
73             }
74             else if (this->control_mode & (1 << CONTROL_VELOCITY_POS))
75             {
76                 msg.control_mode = CONTROL_MODE_VELOCITY;
77                 RCLCPP_INFO(this->get_logger(), "set control mode to velocity");
78             }
79             else if (this->control_mode & (1 << CONTROL_POSITION_POS))
80             {
81                 msg.control_mode = CONTROL_MODE_POSITION;
82                 RCLCPP_INFO(this->get_logger(), "set control mode to position");
83             }
84             RCLCPP_INFO(this->get_logger(), "publishing new control mode %d", msg.control_mode);
85             drone_control_mode_publisher_->publish(msg);
86         }
87     }
88 }
```

```

86     }
87
88     /**
89     * @brief Handle a request to change the vehicle control mode
90     *
91     * @param request_header the header of the service request
92     * @param request the request that was sent to the service
93     * @param response the response that the service sends back to the client.
94     */
95
96     void handle_vehicle_control_change(
97         const std::shared_ptr<rmw_request_id_t> request_header,
98         const std::shared_ptr<drone_services::srv::SetVehicleControl::Request> request,
99         const std::shared_ptr<drone_services::srv::SetVehicleControl::Response> response)
100     {
101         if (request->control < 0 || request->control > (1 << CONTROL_POSITION_POS))
102         {
103             response->success = false;
104         }
105         else
106         {
107             this->control_mode = request->control;
108             RCLCPP_INFO(this->get_logger(), "set control mode to %d", this->control_mode);
109             publish_new_control_mode();
110             response->success = true;
111         }
112     }
113 };
114
115 int main(int argc, char *argv[])
116 {
117     rclcpp::init(argc, argv);
118     rclcpp::spin(std::make_shared<HeartBeat>());
119     rclcpp::shutdown();
120     return 0;
121 }
122

```

A.4: Hoogtemeter node

Deze node leest de waarden van de hoogtemeter uit en publiceert die op de *drone/height* topic. Bij het opstarten van de node wordt de USB poort waar de hoogtemeter aan verbonden zit doorgegeven via een parameter. Omdat de sensor in vier kwadranten meet, wordt de waarde van elk van deze kwadranten uitgelezen en wordt de kleinste waarde berekend. Deze vijf waarden vormen dan het bericht wat gepubliceerd wordt.

```
1  #include <chrono>
2  #include "rclcpp/rclcpp.hpp"
3  #include "height/msg/height_data.hpp"
4  #include <terabee/ITerangerFactory.hpp>
5  #include <terabee/ITerangerEvoMini.hpp>
6  #include <terabee/DistanceData.hpp>
7
8  using namespace std::chrono_literals;
9
10 using terabee::DistanceData;
11 using terabee::ITerangerEvoMini;
12
13 class HeightReader : public rclcpp::Node
14 {
15 public:
16     HeightReader() : rclcpp::Node("height_reader")
17     {
18         rcl_interfaces::msg::ParameterDescriptor descriptor = rcl_interfaces::msg::ParameterDescriptor{};
19         descriptor.description = "serial port for the USB port of the height sensor";
20         this->declare_parameter("height_serial_port", "/dev/ttyACM0", descriptor);
21
22         factory = terabee::ITerangerFactory::getFactory();
23         evo_mini = factory->createTerangerEvoMini(this->get_parameter("height_serial_port").as_string());
24
25         if (!evo_mini)
26         {
27             RCLCPP_ERROR(this->get_logger(), "Failed to create Evo Mini!");
28             return;
29         }
30
31         evo_mini->setPixelMode(ITerangerEvoMini::Px4Mode);
32
33         if (!evo_mini->initialize())
34         {
35             RCLCPP_ERROR(this->get_logger(), "Failed to initialize evo mini!");
36             return;
37         }
38
39         RCLCPP_INFO(this->get_logger(), "Succesfully initialized Evo mini!");
40
41         timer_ = this->create_wall_timer(500ms, std::bind(&HeightReader::read_height, this));
42         publisher_ = this->create_publisher<height::msg::HeightData>("drone/height", 10);
43     }
44
45 private:
46     // timer for publishing height readings
47     rclcpp::TimerBase::SharedPtr timer_;
48     // publisher for height readings
49     rclcpp::Publisher<height::msg::HeightData>::SharedPtr publisher_;
50
51     // factory for height sensor
52     std::unique_ptr<terabee::ITerangerFactory> factory;
53     // height sensor object pointer
54     std::unique_ptr<terabee::ITerangerEvoMini> evo_mini;
55
56     /**
57      * @brief reads the height value from the Terabee Evo Mini sensor and publishes a
58      */
59     void read_height()
60     {
61         auto msg = height::msg::HeightData();
62
63         // high initial minimal value
64         float min = 255;
65         terabee::DistanceData distance_data = evo_mini->getDistance();
66         // add readings and calculate minimum value
67         for (size_t i = 0; i < distance_data.size(); i++)
68         {
69             msg.heights[i] = distance_data.distance[i];
70             if (distance_data.distance[i] < min && distance_data.distance[i] >= 0)
71             {
72                 min = distance_data.distance[i];
73             }
74         }
75     }
76 }
```

```

73     }
74 }
75 // add minimum value and publish message
76 msg.min_height = min;
77 publisher_->publish(msg);
78
79 RCLCPP_INFO(this->get_logger(), "publishing message with min distance %f", msg.min_height);
80 }
81
82 };
83
84 int main(int argc, char **argv)
85 {
86
87     rclcpp::init(argc, argv);
88     rclcpp::spin(std::make_shared<HeightReader>());
89     rclcpp::shutdown();
90
91     return 0;
92 }
93

```

A.5: Multiflex node

Deze node leest de waarden van de objectdetectiesensoren uit en publiceert deze op de `drone/object_detection` topic. Bij het opstarten krijgt de node de USB poort waar de sensor op aangesloten zit mee via een parameter. De data van alle afstandssensoren wordt uitgelezen en gepubliceerd op de topic.

```
1  #include <chrono>
2  #include "rclcpp/rclcpp.hpp"
3  #include "object_detection/msg/multiflex_reading.hpp"
4  #include <terabee/ITerarangerFactory.hpp>
5  #include <terabee/ITerarangerMultiflex.hpp>
6  #include <terabee/DistanceData.hpp>
7
8  using terabee::DistanceData;
9  using namespace std::chrono_literals;
10
11 class MultiflexReader : public rclcpp::Node
12 {
13 public:
14     MultiflexReader() : Node("multiflex_reader")
15     {
16         // get serial port of sensor through parameter
17         rcl_interfaces::msg::ParameterDescriptor serial_port_descriptor =
18         rcl_interfaces::msg::ParameterDescriptor{};
19         serial_port_descriptor.description = "Serial port of the USB port that the multiflex PCB is
20         connected to.";
21         this->declare_parameter("multiflex_serial_port", "/dev/ttyACM0", serial_port_descriptor);
22
23         // create the sensor object
24         factory = terabee::ITerarangerFactory::getFactory();
25         multiflex = factory->createTerarangerMultiflex(this-
26         >get_parameter("multiflex_serial_port").as_string());
27
28         if (!multiflex) // check if the object can be created
29         {
30             RCLCPP_ERROR(this->get_logger(), "Failed to create multiflex");
31             return;
32         }
33
34         if (!multiflex->initialize()) // check if communication with the sensor works
35         {
36             RCLCPP_ERROR(this->get_logger(), "Failed to initialize multiflex");
37             return;
38         }
39
40         if (!multiflex->configureNumberOfSensors(0x3f)) // check if all 6 distance sensors work
41         {
42             RCLCPP_ERROR(this->get_logger(), "Failed to set the number of sensors to 6!");
43             return;
44         }
45
46         RCLCPP_INFO(this->get_logger(), "Multiflex initialized");
47
48         publisher_ = this-
49         >create_publisher<object_detection::msg::MultiflexReading>("drone/object_detection", 10);
50         timer_ = this->create_wall_timer(500ms, std::bind(&MultiflexReader::read_multiflex_data, this));
51     }
52 private:
53     // factory and object for multiflex sensor
54     std::unique_ptr<terabee::ITerarangerFactory> factory;
55     std::unique_ptr<terabee::ITerarangerMultiflex> multiflex;
56
57     // timer and publisher for publishing message onto topic
58     rclcpp::TimerBase::SharedPtr timer_;
59     rclcpp::Publisher<object_detection::msg::MultiflexReading>::SharedPtr publisher_;
60
61     /**
62      * @brief Reads distance data from the sensor and publishes it onto the drone/object_detection topic
63      */
64     void read_multiflex_data()
65     {
66         // get distance reading
67         terabee::DistanceData data = multiflex->getDistance();
68         // populate message with readings
69         auto msg = object_detection::msg::MultiflexReading();
70         for (size_t i = 0; i < data.size(); i++)
71         {
72             msg.distance_data[i] = data.distance[i];
73         }
74     }
75 }
```

```
71         // publish message
72         publisher_->publish(msg);
73     }
74 };
75 int main(int argc, char *argv[])
76 {
77     rclcpp::init(argc, argv);
78     rclcpp::spin(std::make_shared<MultiflexReader>());
79     rclcpp::shutdown();
80     return 0;
81 }
```

A.6: LIDAR node

Deze node leest de waarden van de LIDAR uit. Bij het opstarten ontvangt deze de USB poort waar de LIDAR op aangesloten zit via een parameter. Hierna wordt elke keer de afstandsdata van de LIDAR uitgelezen samen met de data van de ingebouwde IMU. Deze waarden worden dan in een bericht op de `drone/object_detection` topic gepubliceerd.

```
1  #include <chrono>
2  #include "rclcpp/rclcpp.hpp"
3  #include "object_detection/msg/lidar_reading.hpp"
4  #include <terabee/ITerangerFactory.hpp>
5  #include <terabee/ITerangerTowerEvo.hpp>
6  #include <terabee/TowerDistanceData.hpp>
7  #include <terabee/ImuData.hpp>
8
9  using terabee::ImuData;
10 using terabee::ITerangerTowerEvo;
11 using terabee::TowerDistanceData;
12 using namespace std::chrono_literals;
13
14 class LidarReader : public rclcpp::Node
15 {
16 public:
17     LidarReader()
18         : Node("lidar_reader")
19     {
20         // get serial port of sensor through parameter
21         rcl_interfaces::msg::ParameterDescriptor descriptor = rcl_interfaces::msg::ParameterDescriptor{};
22         descriptor.description = "serial port for the USB port of the LIDAR";
23         this->declare_parameter("lidar_serial_port", "/dev/ttyACM0", descriptor);
24
25         // create publisher and bind timer to publish function
26         publisher_ = this->create_publisher<object_detection::msg::LidarReading>("drone/object_detection", 10);
27         timer_ = this->create_wall_timer(500ms, std::bind(&LidarReader::read_lidar_data, this));
28
29         factory = terabee::ITerangerFactory::getFactory();
30         tower = factory->createTerangerTowerEvo(this->get_parameter("lidar_serial_port").as_string());
31
32         if (!tower) // check if the object could be created
33         {
34             RCLCPP_ERROR(this->get_logger(), "Failed to create TerangerTowerEvo");
35             return;
36         }
37         // set lidar to measure including IMU
38         tower->setImuMode(ITerangerTowerEvo::QuaternionLinearAcc);
39
40         if (!tower->initialize()) // check if communication with the sensor works
41         {
42             RCLCPP_ERROR(this->get_logger(), "Failed to initialize TerangerTowerEvo");
43             return;
44         }
45     }
46
47 private:
48     // publisher for lidar data
49     rclcpp::Publisher<object_detection::msg::LidarReading>::SharedPtr publisher_;
50     // timer for publishing readings
51     rclcpp::TimerBase::SharedPtr timer_;
52
53     // terabee tower evo variables
54     std::unique_ptr<terabee::ITerangerTowerEvo> tower;
55     std::unique_ptr<terabee::ITerangerFactory> factory;
56
57     /**
58      * @brief Reads the data from the LIDAR distance modules and the IMU, and publishes it onto the drone/object_detection topic
59      */
60     void read_lidar_data()
61     {
62         {
63             auto msg = object_detection::msg::LidarReading();
64
65             // read distance data from all sensors
66             msg.sensor_1 = tower->getDistance().distance.at(0);
67             msg.sensor_2 = tower->getDistance().distance.at(2);
68             msg.sensor_3 = tower->getDistance().distance.at(4);
69             msg.sensor_4 = tower->getDistance().distance.at(6);
70
71             // read data from built-in IMU
72             ImuData imu_data = tower->getImuData();
73             for (size_t i = 0; i < imu_data.data.size(); i++)
74             {
75                 msg.imu_data.push_back(imu_data.data[i]);
76             }
77
78             // publish message
79             publisher_->publish(msg);
80             RCLCPP_INFO(this->get_logger(), "Publishing message");
81         }
82     }
83 };
84
85 int main(int argc, char *argv[])
```

```
96 {  
97     rclcpp::init(argc, argv);  
98     rclcpp::spin(std::make_shared<LidarReader>());  
99     rclcpp::shutdown();  
100     return 0;  
101 }
```

A.7: CameraController node

Deze node zorgt voor het uitlezen van de camera. Deze start een websocket server op voor de video stream waar de edge computer mee verbindt. Ook biedt deze een *drone/picture* service aan waarmee een foto genomen kan worden.

```
1  import rclpy
2  from rclpy.node import Node
3  from drone_services.srv import TakePicture
4  from sensor_msgs.msg import Image
5  import os
6  from datetime import datetime
7
8  import asyncio
9  import websockets.server
10 import threading
11 import cv2
12 import sys
13 import requests
14
15 #resolution of the camera
16 RES_4K_H = 3496
17 RES_4K_W = 4656
18
19 class CameraController(Node):
20     def __init__(self):
21         """Initialize the camera controller node.
22
23         Initializes the service and the video stream for the websocket connection.
24
25         """
26         super().__init__('camera_controller')
27
28         self.get_logger().info("Camera controller started. Waiting for service call..")
29         self.srv = self.create_service(
30             TakePicture, '/drone/picture', self.take_picture_callback)
31
32         self.websocket = None
33         self.server = None
34         self.event_loop = None
35         self.should_exit = False
36
37         self.timer = self.create_timer(1, self.timer_callback)
38
39         self.video_thread = threading.Thread(target=self.setup_websocket)
40         self.video_thread.start()
41
42     def timer_callback(self):
43         """Callback function for shutting down if an error occurred from a different thread."""
44         if self.should_exit:
45             self.get_logger().info("Shutting down...")
46             self.destroy_node()
47             sys.exit(-1)
48
49     def take_picture_callback(self, request, response):
50         """Callback function for the service call to the /drone/picture service.
51         takes a picture with the given filename and saves it to the drone_img folder.
52         Sends the filename back to the caller.
53
54         """
55         if (request.input_name == "default"):
56             self.get_logger().info("Taking picture with default filename")
57             now = datetime.now().strftime("droneimage_%Y-%m-%d_%H-%M-%S")
58             imagename = "/home/ubuntu/drone_img/" + now + ".jpg"
59             response.filename = imagename
60         else:
61             response.filename = request.input_name
62         os.system('fswebcam -r ' + str(RES_4K_W) + 'x' + str(RES_4K_H) + ' ' + response.filename)
63         self.get_logger().info("Picture saved as " + response.filename)
64
65         return response
66
67     def setup_websocket(self):
68         """Sets up the websocket connection for the video stream on port 9002."""
69         loop = asyncio.new_event_loop()
70         connected = False
71         while not connected:
72             try:
73                 start_server = websockets.serve(self.websocket_video, "0.0.0.0", 9002, loop=loop)
74                 connected = True
75             except Exception as e:
76                 self.get_logger().error("error " + str(e))
77                 connected = False
78         loop.run_until_complete(start_server)
79         loop.run_forever()
80         try:
81             self.destroy_node()
82         except Exception as e:
83             self.get_logger().error("error " + str(e))
84             sys.exit(-1)
85
86     async def websocket_video(self, websocket, path):
87         """Function for the websocket connection on port 9002.
88         This continuously captures frames from the video and sends them to the websocket client.
89         A resolution of 1920x1080 is chosen for performance reasons."""
```

```

89         vid = cv2.VideoCapture(0,cv2.CAP_V4L)
90
91         vid.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
92         vid.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
93         error_amount = 0
94         while True:
95             try:
96                 while(vid.isOpened()):
97                     img, frame = vid.read()
98                     encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 100]
99                     man = cv2.imencode('.jpg', frame)[1]
100                     await websocket.send(man.tobytes())
101                     self.get_logger().error("Not opened")
102                     error_amount += 1
103             except Exception as e:
104                 self.get_logger().error("error " + str(e))
105                 error_amount += 1
106             if error_amount > 20:
107                 self.get_logger().error("Too many errors, closing node")
108                 self.should_exit = True
109                 sys.exit(-1)
110
111
112 def main(args=None):
113     rclpy.init(args=args)
114
115     test_controller = CameraController()
116     rclpy.spin(test_controller)
117
118     test_controller.destroy_node()
119     rclpy.shutdown()
120
121 if __name__ == '__main__':
122     main()

```

A.8: TrackerPositioning node

Deze node leest één beacon op de drone uit om de positie van de drone te bepalen. Deze krijgt de USB poort van de beacon in het begin binnen via een parameter. Daarna leest deze de data van de beacon uit in de vorm van de x-, y- en z coördinaten. Dit bericht wordt dan gepubliceerd op de drone/beacon_positioning topic.

```
1  #include <cstdint>
2  #include <chrono>
3
4  #include "rclcpp/rclcpp.hpp"
5  #include "std_msgs/msg/string.hpp"
6  #include "beacon_positioning/msg/tracker_position.hpp"
7  #include "rtls_driver/rtls_driver.hpp"
8  #include "serial_communication/serial.hpp"
9
10 #define TRACKER_0_PORT "/dev/ttyUSB0"
11 #define TRACKER_1_PORT "/dev/ttyUSB1"
12
13 #define TRACKER_0_PRIORITY 0
14 #define TRACKER_1_PRIORITY 1
15 using SerialInterface = terabee::serial_communication::Serial;
16
17 using namespace std::chrono_literals;
18
19 class TrackerPositioning : public rclcpp::Node
20 {
21 public:
22     TrackerPositioning() : Node("beacon_positioning")
23     {
24
25         this->declare_parameter("tracker_serial_port", "/dev/ttyUSB0");
26
27         std::string serial_port_name = this->get_parameter("tracker_serial_port").as_string();
28
29         RCLCPP_INFO(this->get_logger(), "serial port is %s\n", serial_port_name.c_str());
30
31         serial_port = std::make_shared<SerialInterface>(serial_port_name);
32         serial_port->setBaudrate(115200);
33         serial_port->setTimeout(800ms);
34
35         //rtls_device = terabee::RtlsDevice(serial_port);
36
37         RCLCPP_INFO(this->get_logger(), "Opening serial port of tracker!");
38         serial_port->open();
39
40         if (!serial_port->isOpen())
41         {
42             RCLCPP_ERROR(this->get_logger(), "Failed to open serial port!");
43             return;
44         }
45
46         publisher_ = this->create_publisher<beacon_positioning::msg::TrackerPosition>("drone/beacon_positioning", 10);
47     }
48
49     void setup_rtlsdevice(terabee::RtlsDevice *rtls_device, int priority, int label, int update_time, int network_id, bool
long_message)
50     {
51         rtls_device->disableTrackerStream();
52         serial_port->flushInput();
53         rtls_device->setDevice(terabee::RtlsDevice::device_type::tracker, priority);
54         rtls_device->setLabel(label);
55         rtls_device->setUpdateTime(update_time);
56         rtls_device->setNetworkId(network_id);
57         if (long_message)
58         {
59             rtls_device->setTrackerMessageLong();
60         }
61         else
62         {
63             rtls_device->setTrackerMessageShort();
64         }
65         rtls_device->enableLED();
66         rtls_device->requestConfig();
67         device_configuration = rtls_device->getConfig();
68
69         rtls_device->enableTrackerStream();
70
71         RCLCPP_INFO(this->get_logger(), "Tracker configured");
72     }
73
74     std::shared_ptr<terabee::serial_communication::ISerial> get_serial_port()
75     {
76         return serial_port;
77     }
78
79     void publish(beacon_positioning::msg::TrackerPosition msg)
80     {
81         publisher_->publish(msg);
82     }
83
84     void set_tracker_id(int id)
85     {

```

```

86     if (id >= 0) {
87         tracker_id = id;
88     }
89 }
90
91 int get_tracker_id()
92 {
93     return tracker_id;
94 }
95
96 private:
97 rclcpp::Publisher<beacon_positioning::msg::TrackerPosition>::SharedPtr publisher_; // pointer to publisher object
98 int tracker_id;
99
100 // terabee tower evo variables
101 std::shared_ptr<terabee::serial_communication::ISerial> serial_port; // serial port for communicating with tracker
102 terabee::RtlsDevice::config_t device_configuration;
103 terabee::RtlsDevice::OnTrackerDataCallback tracker_data_callback_;
104 terabee::RtlsDevice::tracker_msg_t tracker_msg;
105 };
106
107 int main(int argc, char **argv)
108 {
109     (void)argc;
110     (void)argv;
111
112     printf("starting tracker node of beacon_positioning package\n");
113
114     rclcpp::init(argc, argv);
115     rclcpp::executors::MultiThreadedExecutor executor;
116     std::shared_ptr<TrackerPositioning> node = std::make_shared<TrackerPositioning>();
117     executor.add_node(node);
118     terabee::RtlsDevice rtls_device(node->get_serial_port());
119     if (node->get_parameter("tracker_serial_port").as_string().compare(TRACKER_0_PORT))
120     {
121         RCLCPP_INFO(node->get_logger(), "Configuring RTLS device for tracker 0");
122         node->set_tracker_id(0);
123         node->setup_rtlsdevice(&rtls_device, 0, 0, 50, 0, true);
124     } else
125     {
126         RCLCPP_INFO(node->get_logger(), "Configuring RTLS device for tracker 1");
127         node->set_tracker_id(1);
128         node->setup_rtlsdevice(&rtls_device, 1, 1, 50, 0, true);
129     }
130
131     rtls_device.registerOnDistanceDataCaptureCallback([&node](const terabee::RtlsDevice::tracker_msg_t &tracker_msg)
132     {
133         if (tracker_msg.is_valid_position)
134         {
135             RCLCPP_INFO(node->get_logger(), "x = %f, y = %f, z = %f", tracker_msg.tracker_position_xyz.at(0),
136             tracker_msg.tracker_position_xyz.at(1), tracker_msg.tracker_position_xyz.at(2));
137             for (size_t i = 0; i < tracker_msg.anchors_data.size(); i++)
138             {
139                 RCLCPP_INFO(node->get_logger(), "anchor number= %d, distance = %d, x = %d, y = %d, z = %d",
140                 tracker_msg.anchors_data[i].number, tracker_msg.anchors_data[i].distance, tracker_msg.anchors_data[i].pos_x,
141                 tracker_msg.anchors_data[i].pos_y, tracker_msg.anchors_data[i].pos_z);
142             }
143             RCLCPP_INFO(node->get_logger(), "");
144             auto message = beacon_positioning::msg::TrackerPosition();
145             message.id = node->get_tracker_id();
146             message.x_pos = tracker_msg.tracker_position_xyz.at(0);
147             message.y_pos = tracker_msg.tracker_position_xyz.at(1);
148             message.z_pos = tracker_msg.tracker_position_xyz.at(2);
149             message.anchor_distances = {0, 0, 0, 0};
150             for (size_t i = 0; i < tracker_msg.anchors_data.size(); i++)
151             {
152                 message.anchor_distances[i] = tracker_msg.anchors_data[i].distance;
153             }
154             node->publish(message);
155         }
156         else
157         {
158             RCLCPP_ERROR(node->get_logger(), "invalid tracker position");
159         }
160     });
161
162     rtls_device.startReadingStream();
163     executor.spin();
164     rclcpp::shutdown();
165
166     rtls_device.stopReadingStream();
167
168     return node->get_serial_port()->close() ? 0 : -1;
169 }

```

A.9: RelaisController node

Deze node zorgt voor de aansturing van het relais waar de communicatielijnen van de flight controller op aangesloten zijn. Deze biedt de service `drone/control_relais` aan om het relais aan of uit te schakelen. Als deze node opgestart wordt schakelt deze het relais aan.

```
1  import rclpy
2  from rclpy.node import Node
3  import sys
4
5  try:
6      import RPi.GPIO as GPIO
7  except RuntimeError:
8      print("Error importing RPi.GPIO!")
9      sys.exit(-1)
10
11 from drone_services.srv import ControlRelais
12 class RelaisController(Node):
13     def __init__(self):
14         super().__init__('relais_controller')
15         self.srv = self.create_service(ControlRelais, '/drone/control_relais', self.control_relais_callback)
16
17         self.relais1_pin = 17
18         self.relais2_pin = 27
19         self.init_gpio()
20         self.turn_relais_on()
21
22     def init_gpio(self):
23         GPIO.setwarnings(False)
24
25         self.get_logger().info(str(GPIO.RPI_INFO))
26
27         GPIO.setmode(GPIO.BCM)
28
29         GPIO.setup(self.relais1_pin, GPIO.OUT)
30         GPIO.setup(self.relais2_pin, GPIO.OUT)
31         self.get_logger().info("GPIO initialized")
32
33     def turn_relais_on(self):
34         GPIO.output(self.relais1_pin, GPIO.HIGH)
35         GPIO.output(self.relais2_pin, GPIO.HIGH)
36         self.get_logger().info("Relais turned on")
37
38     def control_relais_callback(self, request, response):
39         if request.relais1_on:
40             GPIO.output(self.relais1_pin, GPIO.HIGH)
41             response.bits = response.bits | 1
42         else:
43             GPIO.output(self.relais1_pin, GPIO.LOW)
44             response.bits = response.bits & ~(1 << 0)
45         if request.relais2_on:
46             GPIO.output(self.relais2_pin, GPIO.HIGH)
47             response.bits = response.bits | (1 << 1)
48         else:
49             GPIO.output(self.relais2_pin, GPIO.LOW)
50             response.bits = response.bits & ~(1 << 1)
51         return response
52
53     def main(args=None):
54         rclpy.init(args=args)
55         relais_controller = RelaisController()
56         rclpy.spin(relais_controller)
57         relais_controller.destroy_node()
58         rclpy.shutdown()
59
60 if __name__ == '__main__':
61     main()
```

A.10: DroneStatus node

Deze node verzamelt informatie over de status van de drone en publiceert dit op de `drone/status` topic.

```
1 import rclpy
2 from rclpy.node import Node
3
4 from drone_services.msg import DroneStatus
5 from drone_services.msg import DroneControlMode
6 from drone_services.msg import DroneArmStatus
7 from drone_services.msg import DroneRouteStatus
8 from px4_msgs.msg import BatteryStatus
9 from px4_msgs.msg import CpuLoad
10
11 CONTROL_MODE_ATTITUDE = 1
12 CONTROL_MODE_VELOCITY = 2
13 CONTROL_MODE_POSITION = 3
14
15 class DroneStatus(Node):
16     def __init__(self):
17         super().__init__('drone_status')
18         #publish to drone/status topic
19         self.publisher = self.create_publisher(DroneStatus, '/drone/status', 10)
20         self.control_mode_subscriber = self.create_subscription(DroneControlMode, '/drone/control_mode',
21 self.control_mode_callback, 10)
22         self.arm_status_subscriber = self.create_subscription(DroneArmStatus, '/drone/arm_status',
23 self.arm_status_callback, 10)
24         self.route_status_subscriber = self.create_subscription(DroneRouteStatus, '/drone/route_status',
25 self.route_status_callback, 10)
26         self.battery_status_subscriber = self.create_subscription(BatteryStatus, '/fmu/out/battery_status',
27 self.battery_status_callback, 10)
28         self.cpu_load_subscriber = self.create_subscription(CpuLoad, '/fmu/out/cpload', self.cpu_load_callback,
29 10)
30
31         #publish every 0.5 seconds
32         self.timer = self.create_timer(0.5, self.publish_status)
33         self.armed = False
34         self.control_mode = "attitude"
35         self.battery_percentage = 100.0
36         self.cpu_usage = 0.0
37         self.route_setpoint = 0
38
39     def publish_status(self):
40         msg = DroneStatus()
41         msg.armed = self.armed
42         msg.control_mode = self.control_mode
43         msg.battery_percentage = self.battery_percentage
44         msg.cpu_usage = self.cpu_usage
45         msg.route_setpoint = self.route_setpoint
46         self.publisher.publish(msg)
47         self.get_logger().info('Publishing: "%s"' % msg.status)
48
49     def control_mode_callback(self, msg):
50         if msg.control_mode == CONTROL_MODE_ATTITUDE:
51             self.control_mode = "attitude"
52         elif msg.control_mode == CONTROL_MODE_VELOCITY:
53             self.control_mode = "velocity"
54         elif msg.control_mode == CONTROL_MODE_POSITION:
55             self.control_mode = "position"
56         else:
57             self.control_mode = "unknown"
58
59     def arm_status_callback(self, msg):
60         self.armed = msg.armed
61
62     def route_status_callback(self, msg):
63         self.route_setpoint = msg.current_setpoint_index
64
65     def battery_status_callback(self, msg):
66         self.battery_percentage = msg.remaining * 100.0
67
68     def cpu_load_callback(self, msg):
69         self.cpu_usage = msg.load
```

A.11: FailSafe node

Deze node zorgt voor de failsafe. Deze biedt een `drone/enable_failsafe` service aan waarmee de failsafe aangezet kan worden. Als deze aangezet wordt, wordt dit gepubliceerd met de bijbehorende reden op de `drone/failsafe` topic.

```
1  import rclpy
2  from rclpy.node import Node
3
4  from drone_services.srv import EnableFailsafe
5  from drone_services.msg import FailsafeMsg
6  class FailSafe(Node):
7      def __init__(self):
8          super().__init__("failsafe")
9          self.failsafe_enabled = False
10         self.failsafe_msg = ""
11         self.get_logger().info("Failsafe node started")
12         self.service = self.create_service(
13             EnableFailsafe, "/drone/enable_failsafe", self.failsafe_callback)
14         self.failsafe_publisher = self.create_publisher(FailsafeMsg, "/drone/failsafe", 10)
15
16     def failsafe_callback(self, request, response):
17         self.failsafe_enabled = True
18         self.failsafe_msg = request.message
19         response.enabled = self.failsafe_enabled
20         response.message = self.failsafe_msg
21         self.get_logger().info("Failsafe triggered")
22         self.publish_failsafe()
23         return response
24
25     def publish_failsafe(self):
26         msg = FailsafeMsg()
27         msg.enabled = self.failsafe_enabled
28         msg.message = self.failsafe_msg
29         self.failsafe_publisher.publish(msg)
30         self.get_logger().info("Publishing failsafe message")
31
32     def main(args=None):
33         rclpy.init(args=args)
34         failsafe_node = FailSafe()
35         failsafe_node.spin()
36         failsafe_node.destroy_node()
37         rclpy.shutdown()
38
39 if __name__ == '__main__':
40     main()
```

A.12: ApiListener node

Deze node verzorgt de API waar de drone mee bestuurd kan worden. Deze onderhoudt de communicatie met de edge computer en stuurt de drone aan op basis van de commando's die binnenkomen.

```
1  import rclpy
2  from rclpy.node import Node
3
4  from drone_services.msg import DroneStatus
5  from drone_services.msg import FailsafeMsg
6  from drone_services.srv import TakePicture
7  from drone_services.srv import MovePosition
8  from drone_services.srv import EnableFailsafe
9  from drone_services.srv import ArmDrone
10 from drone_services.srv import DisarmDrone
11 from drone_services.srv import ReadyDrone
12 from drone_services.srv import Land
13
14 import asyncio
15 import websockets.server
16 import threading
17 import json
18 from enum import Enum
19 from functools import partial
20 from functools import partial
21
22 class RequestCommand(Enum):
23     """
24     Enum for the commands that can be sent to the API
25     """
26     GET_COMMANDS_TYPES = -1
27     LAND = 0
28     ARM_DISARM = 1
29     MOVE_DIRECTION = 3
30     EMERGENCY_STOP = 6
31
32 class ResponseMessage(Enum):
33     """
34     Enum for the messages that can be sent to the client
35     """
36     ALL_REQUESTS_RESPONSES = -1
37     STATUS = 0
38     MOVE_DIRECTION_RESULT = 2
39     FAILSAFE = 3
40
41 class ApiListener(Node):
42     """
43     Node that listens to the client and sends the commands to the drone
44     """
45
46     def __init__(self):
47         super().__init__('api_listener')
48         self.get_logger().info('ApiListener node started')
49         self.drone_status_subscriber = self.create_subscription(DroneStatus, '/drone/status', self.drone_status_callback, 10)
50         self.failsafe_subscriber = self.create_subscription(FailsafeMsg, "/drone/failsafe", self.failsafe_callback, 10)
51
52         self.timer = self.create_timer(1, self.publish_status)
53
54         self.take_picture_client = self.create_client(TakePicture, '/drone/picture')
55         self.wait_for_service(self.take_picture_client, "Take picture")
56         self.take_picture_request = TakePicture.Request()
57
58         self.move_position_client = self.create_client(MovePosition, '/drone/move_position')
59         self.wait_for_service(self.move_position_client, "Move position")
60         self.move_position_request = MovePosition.Request()
61
62         self.enable_failsafe_client = self.create_client(EnableFailsafe, "/drone/enable_failsafe")
63         self.wait_for_service(self.enable_failsafe_client, "Enable failsafe")
64         self.enable_failsafe_request = EnableFailsafe.Request()
65
66         self.arm_drone_client = self.create_client(ArmDrone, "/drone/arm")
67         self.wait_for_service(self.arm_drone_client, "Arm drone")
68         self.arm_drone_request = ArmDrone.Request()
69
70         self.disarm_drone_client = self.create_client(DisarmDrone, "/drone/disarm")
71         self.wait_for_service(self.disarm_drone_client, "Disarm drone")
72         self.disarm_drone_request = DisarmDrone.Request()
73
74         self.ready_drone_client = self.create_client(ReadyDrone, "/drone/ready")
75         self.wait_for_service(self.ready_drone_client, "Ready drone")
76         self.ready_drone_request = ReadyDrone.Request()
77
78         self.land_client = self.create_client(Land, "/drone/land")
79         self.wait_for_service(self.land_client, "Land drone")
80         self.land_request = Land.Request()
81
82         self.status_data = {}
83         self.status_data_received = False
84         self.last_message = ""
85         self.message_queue = []
86         self.checking_for_message = False
87
88         self.websocket = None
```

```

89     self.server = None
90     self.server_thread = threading.Thread(
91         target=self.start_api_thread, daemon=True)
92     self.server_thread.start()
93     self.response_thread = threading.Thread(
94         target=self.handle_responses, daemon=True)
95     self.response_thread.start()
96
97     self.event_loop = None
98     self.armed = False
99     self.failsafe_enabled = False
100    self.has_sent_failsafe_msg = False
101
102    def wait_for_service(self, client, service_name):
103        """Waits for a client service to be available
104
105        Args:
106            client (ROS2 service client): The client to use to wait fo the service
107            service_name (str): The client name for logging
108        """
109        waiting = 0
110        while not client.wait_for_service(timeout_sec=1.0):
111            if (waiting > 10):
112                self.get_logger().error(service_name + ' service not available, exiting...')
113                exit(-1)
114            self.get_logger().info(service_name + 'service not available, waiting again...')
115            waiting = waiting + 1
116
117    def drone_status_callback(self, msg):
118        """Callback for when a drone_status message is received
119
120        Args:
121            msg (DroneStatus): The received message
122        """
123        try:
124            self.status_data_received = True
125            self.status_data['battery_percentage'] = msg.battery_percentage
126            if msg.battery_percentage < 15:
127                self.enable_failsafe("Battery level too low! Failsafe enabled to prevent damage to battery (" +
str(msg.battery_percentage) + "%)")
128            self.status_data['cpu_usage'] = msg.cpu_usage
129            self.status_data['armed'] = msg.armed
130            self.armed = msg.armed
131            self.status_data['control_mode'] = msg.control_mode
132            self.status_data['route_setpoint'] = msg.route_setpoint
133            self.status_data['velocity'] = [float(msg.velocity[0]), float(
msg.velocity[1]), float(msg.velocity[2])]
134            self.status_data['position'] = [float(msg.position[0]), float(
msg.position[1]), float(msg.position[2])]
135            self.status_data['failsafe'] = msg.failsafe
136            self.status_data['height'] = msg.height
137        except Exception as e:
138            self.get_logger().error(f'Error while parsing drone status message: {e}')
139
140    def failsafe_callback(self, msg):
141        """Callback for when the failsafe gets enabled. Queues a FAILSAFE message to the client
142
143        Args:
144            msg (FailSAFE): The message that was received
145        """
146        if self.failsafe_enabled:
147            return
148
149        if not self.has_sent_failsafe_msg:
150            self.has_sent_failsafe_msg = True
151            self.status_data['failsafe'] = msg.enabled
152            self.message_queue.append(json.dumps({'type': ResponseMessage.FAILSAFE.name, 'message': msg.msg}))
153
154    async def publish_message(self, message):
155        """publishes a message to the NodeJS client
156
157        Args:
158            message (JSON object): the message to send
159        """
160        # self.get_logger().info(f'Publishing message: {message}')
161        if self.websocket is not None:
162            try:
163                await self.websocket.send(message)
164            except Exception as e:
165                self.get_logger().error('Something went wrong while sending a message to the websocket: ' + str(e))
166        else:
167            self.get_logger().error('Trying to publish message but no websocket connection')
168
169    def publish_status(self):
170        """publishes the current status to the client by queueing a STATUS message
171
172        Args:
173            """
174        if self.status_data_received:
175            self.status_data_received = False
176            if self.websocket is not None:
177                try:
178                    self.message_queue.append(json.dumps({'type': ResponseMessage.STATUS.name, 'data': self.status_data}))
179                except Exception as e:
180                    self.get_logger().error('Something went wrong while publishing the status: ' + str(e))
181
182    def handle_responses(self):
183        """Thread to handle responses to send to the client
184
185        Args:
186            """
187        while True:
188            if len(self.message_queue) > 0 and self.websocket is not None:

```

```

187         # self.get_logger().info("sending message")
188         asyncio.run(self.publish_message(self.message_queue.pop(0)))
189
190     def start_api_thread(self):
191         """Starts the API thread"""
192         asyncio.run(self.handle_api())
193
194     async def handle_api(self):
195         """Handles the API requests and starts the websockets server"""
196         self.get_logger().info('Starting API')
197         self.event_loop = asyncio.get_event_loop()
198         self.server = await websockets.serve(self.api_handler, '0.0.0.0', 9001)
199         self.get_logger().info('API started on port 9001')
200         await self.server.wait_closed()
201
202     def send_available_commands(self):
203         """Sends the available commands to the client
204         """
205         print('Sending available commands')
206         requestcommands = {}
207         messagetypes = {}
208         result = {}
209         for command in RequestCommand:
210             requestcommands[command.name] = command.value
211         for message_type in ResponseMessage:
212             messagetypes[message_type.name] = message_type.value
213         result['request_commands'] = requestcommands
214         result['response_messages'] = messagetypes
215         self.message_queue.append(json.dumps({'type': ResponseMessage.ALL_REQUESTS_RESPONSES.name, 'data': result}))
216
217     def handle_direction_message(self, message):
218         """Calls the move position service with the given values
219
220         Args:
221             message (JSON object): the message containing the direction values
222         """
223         self.move_position_request.up_down = float(message['up_down'])
224         self.move_position_request.left_right = float(message['left_right'])
225         self.move_position_request.front_back = float(
226             message['forward_backward'])
227         self.move_position_request.angle = float(message['yaw'])
228         self.get_logger().info(f'Calling move position service with request: {str(self.move_position_request)}')
229
230         self.send_move_position_request()
231
232     def send_move_position_request(self):
233         """Sends the move position request to the move position service"""
234         future = self.move_position_client.call_async(
235             self.move_position_request)
236         future.add_done_callback(partial(self.move_position_service_callback))
237
238     def move_position_service_callback(self, future):
239         """Callback for the move position service
240
241         Args:
242             future (Future): Future object that holds the result
243         """
244         try:
245             result = future.result()
246             message_result = {}
247             message_result['type'] = ResponseMessage.MOVE_DIRECTION_RESULT.name
248             self.get_logger().info(f'Move position service call result: {str(result)}')
249             if result.success == True:
250                 self.get_logger().info('Move position service call success')
251                 message_result['success'] = True
252             else:
253                 self.get_logger().error('Move position service call failed')
254                 message_result['success'] = False
255             self.message_queue.append(json.dumps(message_result))
256         except Exception as e:
257             self.get_logger().error('Something went wrong while sending a move position request and waiting for the response: %r' %
(e))
258
259     def enable_failsafe(self, message):
260         self.get_logger().info("Enabling failsafe")
261         self.enable_failsafe_request.message = message
262         future = self.enable_failsafe_client.call_async(
263             self.enable_failsafe_request)
264         future.add_done_callback(partial(self.enable_failsafe_callback))
265
266     def enable_failsafe_callback(self, future):
267         try:
268             result = future.result()
269             if (result.enabled == True):
270                 self.get_logger().info("Failsafe activated")
271         except Exception as e:
272             self.get_logger().error("Something went wrong while trying to enable failsafe!\n" + str(e))
273
274     def emergency_stop(self):
275         """Sends an emergency stop request to the failsafe service"""
276         self.enable_failsafe("Emergency stop activated")
277
278     def takeoff(self):
279         """Sends a takeoff request to the move position service"""
280         self.get_logger().info('Takeoff command received')
281         self.move_position_request.up_down = 0.1
282         self.move_position_request.left_right = 0.0
283         self.move_position_request.front_back = 0.0
284         self.move_position_request.angle = 0.0

```

```

285         self.send_move_position_request()
286
287     def land(self):
288         """Sends a land request to the move position service"""
289         self.get_logger().info('Land command received')
290         self.move_position_request.up_down = -0.1
291         self.move_position_request.left_right = 0.0
292         self.move_position_request.front_back = 0.0
293         self.move_position_request.angle = 0.0
294         self.send_move_position_request()
295         future = self.land_drone_client.call_async(self.land_drone_request)
296         future.add_done_callback(partial(self.land_service_callback))
297
298     def arm_disarm(self):
299         """Sends an arm or disarm request to the PX4Controller"""
300         if self.armed:
301             self.get_logger().info('Disarm command received')
302             future = self.disarm_drone_client.call_async(
303                 self.disarm_drone_request)
304             future.add_done_callback(partial(self.disarm_service_callback))
305         else:
306             self.get_logger().info('Arm command received')
307             future = self.ready_drone_client.call_async(
308                 self.ready_drone_request)
309             future.add_done_callback(partial(self.ready_drone_callback))
310
311     def ready_drone_callback(self, future):
312         try:
313             result = future.result()
314             if result.success:
315                 self.get_logger().info('Ready service call success')
316             else:
317                 self.get_logger().error('Ready service call failed')
318         except Exception as e:
319             self.get_logger().error('Something went wrong while calling the ready service!\n' + str(e))
320
321     def arm_service_callback(self, future):
322         try:
323             result = future.result()
324             if result.success:
325                 self.get_logger().info('Arm service call success')
326             else:
327                 self.get_logger().error('Arm service call failed')
328         except Exception as e:
329             self.get_logger().error('Something went wrong while calling the arm service!\n' + str(e))
330
331     def disarm_service_callback(self, future):
332         try:
333             result = future.result()
334             if result.success:
335                 self.get_logger().info('Disarm service call success')
336                 self.armed = False
337             else:
338                 self.get_logger().error('Disarm service call failed')
339         except Exception as e:
340             self.get_logger().error('Something went wrong while calling the disarm service!\n' + str(e))
341
342     def land_service_callback(self, future):
343         try:
344             result = future.result()
345             if result.is_landing:
346                 self.get_logger().info('Land service call success')
347             else:
348                 self.get_logger().error('Land service call failed')
349         except Exception as e:
350             self.get_logger().error('Something went wrong while calling the land service!\n' + str(e))
351
352     def consume_message(self, message):
353         """Consumes a message from the client"""
354         self.get_logger().info(f'Consuming message: {message}')
355         try:
356             message_json = json.loads(str(message))
357             if not "command" in message_json:
358                 self.get_logger().error('Received message without command')
359                 self.send_available_commands()
360             else:
361                 self.get_logger().info(
362                     f'Received command: {message_json["command"]}')
363                 if message_json['command'] == RequestCommand.LAND.value:
364                     self.get_logger().info('Land command received')
365                     self.land()
366                 elif message_json['command'] == RequestCommand.ARM_DISARM.value:
367                     self.get_logger().info('Arm/disarm command received')
368                     self.arm_disarm()
369                 elif message_json['command'] == RequestCommand.MOVE_DIRECTION.value:
370                     self.get_logger().info('Move direction command received')
371                     self.handle_direction_message(message_json)
372                 elif message_json['command'] == RequestCommand.GET_COMMANDS_TYPES.value:
373                     self.get_logger().info('Get command received')
374                     self.send_available_commands()
375                 elif message_json['command'] == RequestCommand.EMERGENCY_STOP.value:
376                     self.get_logger().info('Emergency stop command received')
377                     self.emergency_stop()
378             else:
379                 self.get_logger().error('Received unknown command ' + str(message_json['command']))
380             self.send_available_commands()
381         except TypeError:
382             self.get_logger().error('Received unknown type: ' + str(type(message)) + " " + str(message))
383             self.send_available_commands()

```

```

384         except json.JSONDecodeError:
385             self.get_logger().error('Received invalid JSON')
386             self.send_available_commands()
387         except Exception as e:
388             self.get_logger().error('Something went wrong!')
389             self.get_logger().error(str(type(e)))
390             self.get_logger().error(str(e.with_traceback()))
391
392     async def api_handler(self, websocket):
393         """Handles the websocket connection
394         Args:
395             websocket (websockets object): the websocket connection
396         """
397         self.get_logger().info('New connection')
398
399         self.websocket = websocket
400         try:
401             async for message in websocket:
402                 self.get_logger().info(f"Received message: {message}")
403                 self.consume_message(message)
404
405             except websockets.exceptions.ConnectionClosed:
406                 self.get_logger().info('Connection closed')
407                 self.websocket = None
408             except Exception as e:
409                 self.get_logger().error('Something went wrong!')
410                 self.get_logger().error(str(e))
411                 self.websocket = None
412
413 def main(args=None):
414     """Main function"""
415     rclpy.init(args=args)
416
417     api_listener = ApiListener()
418     multithreaded_executor = rclpy.executors.MultiThreadedExecutor()
419     multithreaded_executor.add_node(api_listener)
420     multithreaded_executor.spin()
421
422     api_listener.destroy_node()
423     rclpy.shutdown()
424
425 if __name__ == '__main__':
426     main()

```

A.13: GUI op edge computer

Dit is de code die op de webpagina van de edge computer voor de functionaliteit zorgt. Het zorgt ook voor de communicatie met de API en de videostream van de camera. Hier wordt afgehandeld wanneer een gebruiker op een bepaalde knop drukt en een bijbehorend commando wordt dan naar de API gestuurd. Ook wordt afgehandeld wanneer de failsafe geactiveerd wordt en dit wordt dan weergegeven aan de gebruiker.

```
1  var ws;
2  assign_button_callbacks();
3  openSocket = () => {
4
5      document.getElementById("cameraview").innerHTML = "Camera view: Connecting...";
6      socket = new WebSocket("ws://10.100.0.40:9002/");
7      let msg = document.getElementById("result-video");
8      socket.addEventListener('open', (e) => {
9          console.log("Connected to video")
10         document.getElementById("cameraview").innerHTML = "Camera view: Connected";
11     });
12     socket.addEventListener('message', (e) => {
13         let ctx = msg.getContext("2d");
14         let image = new Image();
15         image.src = URL.createObjectURL(e.data);
16         image.addEventListener("load", (e) => {
17             ctx.drawImage(image, 0, 0, 800, 600);
18         });
19     });
20     socket.addEventListener('close', (e) => {
21         console.log("Disconnected from video")
22         document.getElementById("cameraview").innerHTML = "Camera view: Disconnected. Reload the page to reconnect";
23     });
24     socket.addEventListener('error', (e) => {
25         console.log("Error in video connection")
26         document.getElementById("cameraview").innerHTML = "Camera view: Error. Reload the page to reconnect";
27     });
28 }
29
30 function assign_button_callbacks() {
31     var buttons = document.getElementsByClassName("movebutton");
32     for (var i = 0; i < buttons.length; i++) {
33         buttons[i].addEventListener("mouseup", function () {
34             stop();
35         });
36     }
37     document.getElementById("button_forward").addEventListener("mousedown", function () { forward(); });
38     document.getElementById("button_backward").addEventListener("mousedown", function () { backward(); });
39     document.getElementById("button_left").addEventListener("mousedown", function () { left(); });
40     document.getElementById("button_right").addEventListener("mousedown", function () { right(); });
41     document.getElementById("button_turnleft").addEventListener("mousedown", function () { turn_left(); });
42     document.getElementById("button_turnright").addEventListener("mousedown", function () { turn_right(); });
43     document.getElementById("button_up").addEventListener("mousedown", function () { up(); });
44     document.getElementById("button_down").addEventListener("mousedown", function () { down(); });
45 }
46
47 function send_move_request(data) {
48     console.log("sending move request");
49     if (ws != null)
50         ws.send(data);
51 }
52
53 function turn_left() {
54     console.log("turnleft");
55     send_move_request(JSON.stringify({ "command": 3, "up_down": 0.0, "forward_backward": 0.0, "left_right": 0.0, "yaw": -10.0 }));
56 }
57 function turn_right() {
58     console.log("turnright");
59     send_move_request(JSON.stringify({ "command": 3, "up_down": 0.0, "forward_backward": 0.0, "left_right": 0.0, "yaw": 10.0 }));
60 }
61 function up() {
62     console.log("up");
63     send_move_request(JSON.stringify({ "command": 3, "up_down": 1.0, "forward_backward": 0.0, "left_right": 0.0, "yaw": 0.0 }));
64 }
65 function down() {
66     console.log("down");
67     send_move_request(JSON.stringify({ "command": 3, "up_down": -1.0, "forward_backward": 0.0, "left_right": 0.0, "yaw": 0.0 }));
68 }
69
70 function forward() {
71     console.log("forward");
72     send_move_request(JSON.stringify({ "command": 3, "up_down": 0.0, "forward_backward": 1.0, "left_right": 0.0, "yaw": 0.0 }));
73 }
74 function backward() {
75     console.log("backward");
76     send_move_request(JSON.stringify({ "command": 3, "up_down": 0.0, "forward_backward": -1.0, "left_right": 0.0, "yaw": 0.0 }));
77 }
78
79 function left() {
```

```

80     console.log("left");
81     send_move_request(JSON.stringify({ "command": 3, "up_down": 0.0, "forward_backward": 0.0, "left_right": -1.0, "yaw": 0.0
    }));
82 }
83 function right() {
84     console.log("right");
85     send_move_request(JSON.stringify({ "command": 3, "up_down": 0.0, "forward_backward": 0.0, "left_right": 1.0, "yaw": 0.0 }));
86 }
87 function stop() {
88     console.log("stop");
89     send_move_request(JSON.stringify({ "command": 3, "up_down": 0.0, "forward_backward": 0.0, "left_right": 0.0, "yaw": 0.0 }));
90 }
91
92 function land() {
93     console.log("land");
94     var request = JSON.stringify({ command: 0 });
95     ws.send(request);
96 }
97
98
99 function estop() {
100     console.log("estop");
101     var request = JSON.stringify({
102         command: 6,
103     });
104     ws.send(request);
105 }
106
107 function take_picture() {
108     console.log("take picture");
109     var image = new Image();
110     image.src = document.getElementById("result-video").toDataURL();
111     image.width = 400;
112     document.getElementById("lastimg").innerHTML = "";
113     document.getElementById("lastimg").appendChild(image);
114 }
115
116 function arm_disarm() {
117     console.log("arm/disarm");
118     var request = JSON.stringify({ command: 1 });
119     ws.send(request);
120 }
121
122 function handle_ws_message(data) {
123     set_timeout = false;
124     if (data.type == "STATUS") {
125         document.getElementById("batterypercentage").innerHTML = "Battery percentage: " +
data.data.battery_percentage.toString().substring(0, 4) + "%";
126         document.getElementById("cpuload").innerHTML = "CPU load: " + data.data.cpu_usage.toString().substring(0,
6).substring(2, 4) + "%";
127         document.getElementById("armed").innerHTML = "Armed: " + data.data.armed;
128         document.getElementById("control_mode").innerHTML = "Control mode: " + data.data.control_mode;
129         document.getElementById("speed").innerHTML = "Current speed (m/s): x: " +
data.data.velocity[0].toString().substring(0,4) + " y: " + data.data.velocity[1].toString().substring(0,4) + " z: " +
data.data.velocity[2].toString().substring(0,4);
130         document.getElementById("position").innerHTML = "Current position (m): x: " +
data.data.position[0].toString().substring(0,4) + " y: " + data.data.position[1].toString().substring(0,4) + " z: " +
data.data.position[2].toString().substring(0,4);
131     } else if (data.type == "FAILSAFE") {
132         document.getElementById("failsafe").innerHTML = "Failsafe: ACTIVATED";
133         document.getElementById("failsafe").style.backgroundColor = "red";
134         document.getElementById("failsafe").style.color = "white";
135         document.getElementById("failsafe").style.textDecoration = "bold";
136         alert("Failsafe enabled! Drone is landing. The failsafe message is:\n" + data.message);
137     } else if (data.type == "API_HEARTBEAT") {
138         console.log("Got heartbeat from API")
139         clearTimeout(api_timeout);
140     }
141 }
142
143 function local_connect() {
144     console.log("Connecting to API");
145     ws = new WebSocket("ws://10.100.0.40:9001/");
146
147     ws.addEventListener("open", function open() {
148         console.log("connected with websockets to API!");
149         document.getElementById("connectedlabel").innerHTML = "Connected to drone";
150         document.getElementById("connectbutton").disabled = true;
151         openSocket();
152     });
153
154     ws.addEventListener("message", function message(message) {
155         try {
156             var msg = JSON.parse(message.data);
157             handle_ws_message(msg);
158         } catch (error) {
159             console.log("could not parse as json");
160             console.error(error)
161         }
162     });
163
164     ws.addEventListener("error", function error(err) {
165         console.log("there was an error! " + err);
166         console.log("Showing alert!");
167         alert("There was an error connecting to the API!");
168         document.getElementById("connectedlabel").innerHTML = "Not connected to drone";
169         document.getElementById("connectbutton").disabled = false;
170     });
171 }

```

```

172
173     ws.addEventListener("close", function close(event) {
174         console.log("connection closed");
175         alert("Connection to API closed!");
176         document.getElementById("connectedlabel").innerHTML = "Not connected to drone";
177         document.getElementById("connectbutton").disabled = false;
178     });
179 }
180
181 function connect() {
182     var received = false;
183     var xhr = new XMLHttpRequest();
184     xhr.open("GET", "/connect", true);
185     xhr.onreadystatechange = function () {
186         if (this.status == 200) {
187             console.log(this.responseText);
188             if (this.responseText.length > 0) {
189                 var status = JSON.parse(this.responseText);
190                 document.getElementById("connectedlabel").innerHTML = "Connected to drone";
191                 document.getElementById("connectbutton").disabled = true;
192             }
193         } else {
194             console.log("error");
195             document.getElementById("connectedlabel").innerHTML = "Not connected to drone";
196             if (!received) {
197                 alert("Could not connect to API!");
198                 received = true;
199             }
200         }
201     }
202 };
203 xhr.send();
204 }

```

Bijlage B: Code en output tests

B.1: Camera unittest

De code voor test CAM.U.1 is hieronder te zien. Het resultaat van deze test is te zien in Figuur B.1. In deze figuur is te zien dat het bestand `build/camera/pytest.xml` één uitgevoerde test en geen gefaalde tests bevat.

```
1  import os
2  import sys
3  import unittest
4  import time
5
6  import launch
7  import launch_ros
8  import launch_ros.actions
9  import launch_testing.actions
10 import pytest
11 import rclpy
12
13 from drone_services.srv import TakePicture
14
15 @pytest.mark.rostest
16 def generate_test_description():
17     file_path = os.path.dirname(__file__)
18     camera_node = launch_ros.actions.Node(
19         executable=sys.executable,
20         arguments=[os.path.join(file_path, '..', 'camera', 'camera_controller.py')],
21         additional_env={'PYTHONUNBUFFERED': '1'}
22     )
23
24     return (
25         launch.LaunchDescription([
26             camera_node,
27             launch_testing.actions.ReadyToTest(),
28         ]),
29         {
30             'camera_node': camera_node,
31         }
32     )
33
34 class CameraUnitTest(unittest.TestCase):
35
36     @classmethod
37     def setUpClass(cls):
38         rclpy.init()
39
40     @classmethod
41     def tearDownClass(cls):
42         rclpy.shutdown()
43
44     def setUp(self):
45         self.node = rclpy.create_node('test_camera')
46         self.service_called = False
47
48     def tearDown(self):
49         self.node.destroy_node()
50
51     def service_call_callback(self, future):
52         self.assertIsNotNone(future.result())
53         self.assertEqual(future.result().filename, "/home/ubuntu/test_image.jpg")
54         self.assertTrue(os.path.exists("/home/ubuntu/test_image.jpg"))
55         self.service_called = True
56
57     def test_camera_save_image(self, camera_node, proc_output):
58         # call camera service
59         camera_client = self.node.create_client(TakePicture, '/drone/picture')
60         while not camera_client.wait_for_service(timeout_sec=1.0):
61             self.node.get_logger().info('camera service not available, waiting again...')
62         request = TakePicture.Request()
63         request.input_name = "/home/ubuntu/test_image.jpg"
64         try:
65             while True:
66                 rclpy.spin_once(self.node, timeout_sec=0.1)
67                 if not self.service_called:
68                     camera_client.call_async(request).add_done_callback(self.service_call_callback)
69                 else:
67                     break
68         finally:
69             self.node.destroy_client(camera_client)
```

```

ubuntu@ubuntu:~/ros2_ws$ colcon test --packages-select camera && colcon test-result --verbose --all
Starting >>> camera
/usr/local/lib/python3.8/dist-packages/setuptools/dist.py:788: UserWarning: Usage of dash-separated 'script_dir' instead
the underscore name 'script_dir' instead
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/setuptools/dist.py:788: UserWarning: Usage of dash-separated 'install_scripts' instead
use the underscore name 'install_scripts' instead
  warnings.warn(
Finished <<< camera [14.3s]

Summary: 1 package finished [16.1s]
[1.439s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/height/Testing/TAG': could not find
'
[1.467s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/drone_services/Testing/TAG': could not find
'20230607-1405/Test.xml'
[1.511s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/object_detection/Testing/TAG': could not find
'230607-1405/Test.xml'
[1.532s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/px4_msgs/Testing/TAG': could not find
'.xml'
build/camera/pytest.xml: 1 test, 0 errors, 0 failures, 0 skipped
build/failsafe/pytest.xml: 1 test, 0 errors, 0 failures, 0 skipped
build/px4_connection/Testing/20230607-1539/Test.xml: 1 test, 0 errors, 0 failures, 0 skipped
Summary: 3 tests, 0 errors, 0 failures, 0 skipped

```

Figuur B.1: Output camera unit test

B.2: Failsafe unittest

De code voor test FAIL.U.1 is hieronder te zien. Het resultaat van de test is te zien in Figuur B.2. Hierin is te zien dat de test is geslaagd.

```
1 import os
2 import sys
3 import unittest
4 import time
5
6 import launch
7 import launch_ros
8 import launch_ros.actions
9 import launch_testing.actions
10 import pytest
11 import rclpy
12
13 from drone_services.srv import EnableFailsafe
14 from drone_services.msg import FailsafeMsg
15
16 # launch node
17 @pytest.mark.rostest
18 def generate_test_description():
19     file_path = os.path.dirname(__file__)
20     failsafe_node = launch_ros.actions.Node(
21         executable=sys.executable,
22         arguments=[os.path.join(file_path, '..', 'failsafe', 'failsafe.py')],
23         additional_env={'PYTHONUNBUFFERED': '1'})
24
25
26     return (
27         launch.LaunchDescription([
28             failsafe_node,
29             launch_testing.actions.ReadyToTest(),
30         ]),
31         {
32             'failsafe_node': failsafe_node,
33         }
34     )
35
36 class FailsafeUnitTest(unittest.TestCase):
37
38     @classmethod
39     def setUpClass(cls):
40         rclpy.init()
41
42     @classmethod
43     def tearDownClass(cls):
44         rclpy.shutdown()
45
46     def setUp(self):
47         self.node = rclpy.create_node('test_failsafe')
48         self.service_called = False
49
50     def tearDown(self):
51         self.node.destroy_node()
52
53     def service_call_callback(self, future):
54         self.assertIsNotNone(future.result())
55         self.assertTrue(future.result().enabled)
56         self.assertEqual(future.result().message, "test")
57         self.service_called = True
58
59     def test_failsafe_node_enables(self, failsafe_node, proc_output):
60         failsafe_msgs = []
61         failsafe_subscription = self.node.create_subscription(FailsafeMsg, "/drone/failsafe", lambda msg: failsafe_msgs.append(msg),
62 10)
63         failsafe_client = self.node.create_client(EnableFailsafe, "/drone/enable_failsafe")
64         while not failsafe_client.wait_for_service(timeout_sec=1.0):
65             self.node.get_logger().info("service not available, waiting again...")
66         request = EnableFailsafe.Request()
67         request.message = "test"
68
69         end_time = time.time() + 10.0
70
71         try:
72             while time.time() < end_time:
73                 rclpy.spin_once(self.node, timeout_sec=0.1)
74                 if (not self.service_called):
75                     future = failsafe_client.call_async(request)
76                     future.add_done_callback(self.service_call_callback)
77                 else:
78                     break
79             self.assertTrue(failsafe_msgs[0].enabled)
80             self.assertEqual(failsafe_msgs[0].msg, "test")
81             self.assertTrue(self.service_called)
82         finally:
83             self.node.destroy_subscription(failsafe_subscription)
84             self.node.destroy_client(failsafe_client)
```

```

ubuntu@ubuntu:~/ros2_ws$ colcon test --packages-select failsafe && colcon test-result --all --verbose
Starting >>> failsafe
/usr/local/lib/python3.8/dist-packages/setuptools/dist.py:788: UserWarning: Usage of dash-separated 'sc
warnings.warn(
/usr/local/lib/python3.8/dist-packages/setuptools/dist.py:788: UserWarning: Usage of dash-separated 'in
warnings.warn(
Finished <<< failsafe [6.34s]

Summary: 1 package finished [8.96s]
[1.637s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/height/Testing/TAG': could not f
[1.670s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/drone_services/Testing/TAG': cou
[1.725s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/object_detection/Testing/TAG': c
[1.751s] WARNING:colcon.colcon_cmake.test_result.ctest:Skipping 'build/px4_msgs/Testing/TAG': could not
build/failsafe/pytest.xml: 1 test, 0 errors, 0 failures, 0 skipped

Summary: 1 test, 0 errors, 0 failures, 0 skipped

```

Figuur B.2: Resultaat test FAIL.U.1

B.3: Heartbeat control mode unittest

De code voor test PX4.U.1 is hieronder te zien. De output van de test is te zien in Figuur B.3. Hierin is te zien dat de test is geslaagd. Dit is te zien door de tekst `OK` die voor elke test case is aangegeven en de `OK` aan het einde van de test.

```
1  import os
2  import sys
3  import unittest
4
5  import launch
6  import launch_ros
7  import launch_ros.actions
8  import launch_testing.actions
9  import pytest
10 import rclpy
11
12 from drone_services.srv import SetVehicleControl
13 from drone_services.msg import DroneControlMode
14
15 @pytest.mark.rostest
16 def generate_test_description():
17     heartbeat_node = launch_ros.actions.Node(
18         package='px4_connection', executable='heartbeat')
19
20     return (
21         launch.LaunchDescription([
22             heartbeat_node,
23             launch_testing.actions.ReadyToTest(),
24         ]),
25         {
26             'heartbeat_node': heartbeat_node,
27         }
28     )
29
30 class TestHeartbeatControlMode(unittest.TestCase):
31
32     @classmethod
33     def setUpClass(cls):
34         rclpy.init()
35
36     @classmethod
37     def tearDownClass(cls):
38         rclpy.shutdown()
39
40     def setUp(self):
41         self.node = rclpy.create_node('test_heartbeat_control_mode')
42         self.called_control_mode_service = False
43         self.received_control_mode = False
44
45     def tearDown(self):
46         self.node.destroy_node()
47
48     def control_mode_callback(self, msg):
49         self.received_control_mode = True
50         self.assertEqual(msg.control_mode, 1)
51
52     def heartbeat_service_callback_correct(self, future):
53         self.called_control_mode_service = True
54         self.assertIsNotNone(future.result())
55         self.assertTrue(future.result().success)
56
57     def heartbeat_service_callback_wrong(self, future):
58         self.called_control_mode_service = True
59         self.assertIsNotNone(future.result())
60         self.assertFalse(future.result().success)
61
62     def test_set_vehicle_control_mode_proper_value(self, heartbeat_node, proc_output):
63         self.called_control_mode_service = False
64         heartbeat_client = self.node.create_client(SetVehicleControl, '/drone/set_vehicle_control')
65         while not heartbeat_client.wait_for_service(timeout_sec=1.0):
66             self.node.get_logger().info('heartbeat service not available, waiting again...')
67         heartbeat_sub = self.node.create_subscription(DroneControlMode, "/drone/control_mode", self.control_mode_callback, 10)
68         request = SetVehicleControl.Request()
69         request.control = 4 # attitude control
70
71         try:
72             while True:
73                 rclpy.spin_once(self.node, timeout_sec=0.1)
74                 if not self.called_control_mode_service:
75                     vehicle_control_future = heartbeat_client.call_async(request)
76                     vehicle_control_future.add_done_callback(self.heartbeat_service_callback_correct)
77                 elif not self.received_control_mode:
78                     continue
79                 else:
80                     break
81
82         finally:
83             self.node.destroy_client(heartbeat_client)
84
85     def test_set_vehicle_control_mode_wrong_value(self, heartbeat_node, proc_output):
86         self.called_control_mode_service = False
87         heartbeat_client = self.node.create_client(SetVehicleControl, '/drone/set_vehicle_control')
88         while not heartbeat_client.wait_for_service(timeout_sec=1.0):
```

```

89         self.node.get_logger().info('heartbeat service not available, waiting again...')
90         request = SetVehicleControl.Request()
91         request.control = 33 # wrong control mode
92
93         try:
94             while True:
95                 rclpy.spin_once(self.node, timeout_sec=0.1)
96                 if not self.called_control_mode_service:
97                     vehicle_control_future = heartbeat_client.call_async(request)
98                     vehicle_control_future.add_done_callback(self.heartbeat_service_callback_wrong)
99                 else:
100                     break
101         finally:
102             self.node.destroy_client(heartbeat_client)

```

```

ubuntu@ubuntu:~/ros2_ws$ launch_test src/px4_connection/test/test_heartbeat_control_mode.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2023-06-07-18-39-21-828863-ubuntu-14127
[INFO] [launch]: Default logging verbosity is set to INFO
test_set_vehicle_control_mode_proper_value (test_heartbeat_control_mode.TestHeartbeatControlMode) ... [INFO] [he
[heartbeat-1] [INFO] [1686155961.940406352] [heartbeat]: initialized service /drone/set_vehicle_control
[heartbeat-1] [INFO] [1686155961.945902271] [heartbeat]: done initializing at -1422020400 seconds. Sending heart
[heartbeat-1] [INFO] [1686155962.085139650] [heartbeat]: set control mode to 4
[heartbeat-1] [INFO] [1686155962.085411941] [heartbeat]: set control mode to attitude
[heartbeat-1] [INFO] [1686155962.085477976] [heartbeat]: publishing new control mode 1
ok
test_set_vehicle_control_mode_wrong_value (test_heartbeat_control_mode.TestHeartbeatControlMode) ... [heartbeat-
ol mode to 4
[heartbeat-1] [INFO] [1686155962.089118842] [heartbeat]: set control mode to attitude
[heartbeat-1] [INFO] [1686155962.089176878] [heartbeat]: publishing new control mode 1
ok
-----
[heartbeat-1] [INFO] [1686155962.091448774] [heartbeat]: set control mode to 4
Ran 2 tests in 0.329s
OK

```

Figuur B.3: Resultaat test PX4.U.1

B.4: Integratietest failsafe en PX4Controller

De code voor test PX4.I.1 is hieronder te zien. Het resultaat van deze test is te zien in Figuur B.4. Hierin is te zien dat de test slaagt doordat er drie keer `OK` in de output verschijnt (één voor iedere testcase). Ook is `OK` onderaan de output te zien. Dit geeft aan dat de gehele test geslaagd is.

```
1  import os
2  import unittest
3
4  import launch
5  import launch_ros
6  import launch_ros.actions
7  import launch_testing.actions
8  import pytest
9  import rclpy
10 import time
11
12 from drone_services.srv import EnableFailsafe
13 from drone_services.srv import SetTrajectory
14 from drone_services.srv import SetAttitude
15 from drone_services.srv import ArmDrone
16
17 @pytest.mark.rostest
18 def generate_test_description():
19     file_path = os.path.dirname(__file__)
20     px4_controller_node = launch_ros.actions.Node(
21         package='px4_connection', executable='px4_controller')
22     failsafe_node = launch_ros.actions.Node(
23         package='failsafe', executable='failsafe')
24
25     return (
26         launch.LaunchDescription([
27             px4_controller_node,
28             failsafe_node,
29             launch_testing.actions.ReadyToTest(),
30         ]),
31         {
32             'px4_controller_node': px4_controller_node,
33             'failsafe_node': failsafe_node,
34         }
35     )
36
37 class TestPx4Failsafe(unittest.TestCase):
38
39     @classmethod
40     def setUpClass(cls):
41         rclpy.init()
42
43     @classmethod
44     def tearDownClass(cls):
45         rclpy.shutdown()
46
47     def setUp(self):
48         self.node = rclpy.create_node('test_px4_failsafe')
49         self.called_failsafe_service = False
50         self.called_trajectory_service = False
51         self.called_arm_service = False
52         self.called_attitude_service = False
53
54     def tearDown(self):
55         self.node.destroy_node()
56
57     def failsafe_service_callback(self, future):
58         self.called_failsafe_service = True
59
60     def trajectory_service_callback(self, future):
61         self.called_trajectory_service = True
62         self.assertFalse(future.result().success)
63
64     def attitude_service_callback(self, future):
65         self.called_attitude_service = True
66         self.assertFalse(future.result().success)
67
68     def arm_service_callback(self, future):
69         self.called_arm_service = True
70         self.assertFalse(future.result().success)
71
72     def test_failsafe_enabled_set_trajectory_returns_false(self, px4_controller_node, proc_output):
73         self.called_failsafe_service = False
74         failsafe_client = self.node.create_client(EnableFailsafe, '/drone/enable_failsafe')
75         set_trajectory_client = self.node.create_client(SetTrajectory, '/drone/set_trajectory')
76         while not failsafe_client.wait_for_service(timeout_sec=1.0):
77             self.node.get_logger().info('failsafe service not available, waiting again...')
78         while not set_trajectory_client.wait_for_service(timeout_sec=1.0):
79             self.node.get_logger().info('set_trajectory service not available, waiting again...')
80         failsafe_request = EnableFailsafe.Request()
81         failsafe_request.message = "test"
82         set_trajectory_request = SetTrajectory.Request()
83         set_trajectory_request.control_mode = 2
84         set_trajectory_request.values = [1.0, 1.0, 1.0]
85         set_trajectory_request.yaw = 0.0
86
87         end_time = time.time() + 10.0
88         try:
```

```

89         while time.time() < end_time:
90             rclpy.spin_once(self.node, timeout_sec=0.1)
91             if not self.called_failsafe_service:
92                 failsafe_future = failsafe_client.call_async(failsafe_request)
93                 failsafe_future.add_done_callback(self.failsafe_service_callback)
94             elif not self.called_trajectory_service:
95                 trajectory_future = set_trajectory_client.call_async(set_trajectory_request)
96                 trajectory_future.add_done_callback(self.trajectory_service_callback)
97             else:
98                 break
99             self.assertTrue(self.called_failsafe_service, "Failsafe service was not called")
100             self.assertTrue(self.called_trajectory_service, "Trajectory service was not called")
101
102         finally:
103             self.node.destroy_client(failsafe_client)
104             self.node.destroy_client(set_trajectory_client)
105
106     def test_failsafe_enabled_set_attitude_returns_false(self, px4_controller_node, proc_output):
107         self.called_failsafe_service = False
108         failsafe_client = self.node.create_client(EnableFailsafe, '/drone/enable_failsafe')
109         set_attitude_client = self.node.create_client(SetAttitude, '/drone/set_attitude')
110         while not failsafe_client.wait_for_service(timeout_sec=1.0):
111             self.node.get_logger().info('failsafe not available, waiting again...')
112         while not set_attitude_client.wait_for_service(timeout_sec=1.0):
113             self.node.get_logger().info('attitude not available, waiting again...')
114         failsafe_request = EnableFailsafe.Request()
115         failsafe_request.message = "test"
116         set_attitude_request = SetAttitude.Request()
117         set_attitude_request.pitch = 1.0
118         set_attitude_request.yaw = 1.0
119         set_attitude_request.roll = 1.0
120         set_attitude_request.thrust = 0.5
121
122         end_time = time.time() + 10.0
123         try:
124             while time.time() < end_time:
125                 rclpy.spin_once(self.node, timeout_sec=0.1)
126                 if not self.called_failsafe_service:
127                     failsafe_future = failsafe_client.call_async(failsafe_request)
128                     failsafe_future.add_done_callback(self.failsafe_service_callback)
129                 elif not self.called_attitude_service:
130                     attitude_future = set_attitude_client.call_async(set_attitude_request)
131                     attitude_future.add_done_callback(self.attitude_service_callback)
132                 else:
133                     break
134                 self.assertTrue(self.called_failsafe_service, "Failsafe service was not called")
135                 self.assertTrue(self.called_attitude_service, "Attitude service was not called")
136
137         finally:
138             self.node.destroy_client(failsafe_client)
139             self.node.destroy_client(set_attitude_client)
140
141     def test_failsafe_enabled_arm_returns_false(self, px4_controller_node, proc_output):
142         self.called_failsafe_service = False
143         failsafe_client = self.node.create_client(EnableFailsafe, '/drone/enable_failsafe')
144         arm_client = self.node.create_client(ArmDrone, '/drone/arm')
145         while not failsafe_client.wait_for_service(timeout_sec=1.0):
146             self.node.get_logger().info('failsafe not available, waiting again...')
147         while not arm_client.wait_for_service(timeout_sec=1.0):
148             self.node.get_logger().info('arm not available, waiting again...')
149         failsafe_request = EnableFailsafe.Request()
150         failsafe_request.message = "test"
151         arm_request = ArmDrone.Request()
152
153         end_time = time.time() + 10.0
154         try:
155             while time.time() < end_time:
156                 rclpy.spin_once(self.node, timeout_sec=0.1)
157                 if not self.called_failsafe_service:
158                     failsafe_future = failsafe_client.call_async(failsafe_request)
159                     failsafe_future.add_done_callback(self.failsafe_service_callback)
160                 elif not self.called_arm_service:
161                     arm_future = arm_client.call_async(arm_request)
162                     arm_future.add_done_callback(self.arm_service_callback)
163                 else:
164                     break
165                 self.assertTrue(self.called_failsafe_service, "Failsafe service was not called")
166                 self.assertTrue(self.called_arm_service, "Arm service was not called")
167
168         finally:
169             self.node.destroy_client(failsafe_client)
170             self.node.destroy_client(arm_client)

```

```

ubuntu@ubuntu:~/ros2_ws$ launch_test src/px4_connection/test/test_failsafe_enabled.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2023-06-07-17-40-0
[INFO] [launch]: Default logging verbosity is set to INFO
test_failsafe_enabled_arm_returns_false (test_failsafe_enabled.TestPx4Failsafe) ... [INFO]
[INFO] [failsafe-2]: process started with pid [13363]
[px4_controller-1] [INFO] [1686152401.588029909] [px4_controller]: finished initializing a
[failsafe-2] [INFO] [1686152402.341466261] [failsafe]: Failsafe node started
[failsafe-2] [INFO] [1686152402.502310639] [failsafe]: Failsafe triggered! Message: test
[failsafe-2] [INFO] [1686152402.506461680] [failsafe]: Publishing failsafe message
[px4_controller-1] [INFO] [1686152402.509551373] [px4_controller]: Got arm request...
ok
test_failsafe_enabled_set_attitude_returns_false (test_failsafe_enabled.TestPx4Failsafe) .
request...
[px4_controller-1] [INFO] [1686152402.512206816] [px4_controller]: Got arm request...
[failsafe-2] [INFO] [1686152402.515142864] [failsafe]: Failsafe triggered! Message: test
[failsafe-2] [INFO] [1686152402.518922838] [failsafe]: Publishing failsafe message
[px4_controller-1] [INFO] [1686152402.588069760] [px4_controller]: Failsafe enabled, landi
[px4_controller-1] [INFO] [1686152402.688097334] [px4_controller]: Failsafe enabled, landi
[px4_controller-1] [INFO] [1686152402.788078704] [px4_controller]: Failsafe enabled, landi
[failsafe-2] [INFO] [1686152402.840730555] [failsafe]: Failsafe triggered! Message: test
[failsafe-2] [INFO] [1686152402.843997485] [failsafe]: Publishing failsafe message
ok
test_failsafe_enabled_set_trajectory_returns_false (test_failsafe_enabled.TestPx4Failsafe)
afe enabled, ignoring attitude setpoint
[px4_controller-1] [INFO] [1686152402.847036993] [px4_controller]: Failsafe enabled, ignor
[px4_controller-1] [INFO] [1686152402.848249042] [px4_controller]: Failsafe enabled, ignor
[failsafe-2] [INFO] [1686152402.848887491] [failsafe]: Failsafe triggered! Message: test
[failsafe-2] [INFO] [1686152402.852511413] [failsafe]: Publishing failsafe message
[failsafe-2] [INFO] [1686152402.856892467] [failsafe]: Failsafe triggered! Message: test
[failsafe-2] [INFO] [1686152402.860276931] [failsafe]: Publishing failsafe message
[px4_controller-1] [INFO] [1686152402.888454511] [px4_controller]: Failsafe enabled, landi
[failsafe-2] [INFO] [1686152402.926651765] [failsafe]: Failsafe triggered! Message: test
[failsafe-2] [INFO] [1686152402.929382725] [failsafe]: Publishing failsafe message
ok
[px4_controller-1] [INFO] [1686152402.931997631] [px4_controller]: Failsafe enabled, ignor
-----
Ran 3 tests in 1.517s

```

Figuur B.4: Output test PX4.I.1

B.5: Integratietest PositionChanger, LIDAR en Failsafe

De code voor test POS.I.1 en POS.I.2 is hieronder te zien. Hierin is de test POS.I.1 geïmplementeerd in regels 69 – 102, en test POS.I.2 is geïmplementeerd vanaf regel 104. Het resultaat van beide tests is te zien in Figuur B.5. Hierin is te zien dat de testen zijn geslaagd doordat er twee keer `ok` in de output verschijnt en de output bevat ook een regel met `OK` voor de hele code. De output in de figuur is ingekort om het beter in het verslag weer te kunnen geven.

```
1  import unittest
2
3  import launch
4  import launch_ros
5  import launch_ros.actions
6  import launch_testing.actions
7  import pytest
8  import rclpy
9  import time
10
11 from drone_services.srv import MovePosition
12 from drone_services.msg import FailsafeMsg
13 from drone_services.msg import LidarReading
14
15 @pytest.mark.rostest
16 def generate_test_description():
17     positionchanger_node = launch_ros.actions.Node(
18         package='drone_controls', executable='position_changer')
19     failsafe_node = launch_ros.actions.Node(
20         package='failsafe', executable='failsafe')
21     px4_controller_node = launch_ros.actions.Node(
22         package='px4_connection', executable='px4_controller')
23     heartbeat_node = launch_ros.actions.Node(
24         package='px4_connection', executable='heartbeat')
25
26     return (
27         launch.LaunchDescription([
28             positionchanger_node,
29             failsafe_node,
30             px4_controller_node,
31             heartbeat_node,
32             launch_testing.actions.ReadyToTest(),
33         ]),
34         {
35             'positionchanger_node': positionchanger_node,
36             'failsafe_node': failsafe_node,
37             'px4_controller_node': px4_controller_node,
38             'heartbeat_node': heartbeat_node
39         }
40     )
41
42 class TestPositionChanger(unittest.TestCase):
43
44     @classmethod
45     def setUpClass(cls):
46         rclpy.init()
47
48     @classmethod
49     def tearDownClass(cls):
50         rclpy.shutdown()
51
52     def setUp(self):
53         self.node = rclpy.create_node('test_positionchanger')
54         self.called_positionchanger_service = False
55         self.received_failsafe_callback = False
56
57     def tearDown(self):
58         self.node.destroy_node()
59
60     def failsafe_callback(self, msg):
61         self.assertTrue(msg.enabled, "Failsafe was not enabled!")
62         self.received_failsafe_callback = True
63
64     def move_position_callback(self, future):
65         self.assertFalse(future.result(
66             ).success, "MovePosition service call was successful, but should have failed!")
67         self.called_positionchanger_service = True
68
69     def test_positionchanger_no_lidar_data(self, positionchanger_node, proc_output):
70         self.received_failsafe_callback = False
71         self.called_positionchanger_service = False
72         failsafe_subscriber = self.node.create_subscription(
73             FailsafeMsg, '/drone/failsafe', self.failsafe_callback, 10)
74         move_position_client = self.node.create_client(
75             MovePosition, '/drone/move_position')
76         while not move_position_client.wait_for_service(timeout_sec=1.0):
77             self.node.get_logger().info('move_position service not available, waiting again...')
78
79         request = MovePosition.Request()
80         request.front_back = 1.0
81         request.left_right = 1.0
82         request.up_down = 1.0
83         request.angle = 1.0
84
```

```

85     end_time = time.time() + 10.0
86     try:
87         while time.time() < end_time:
88             rclpy.spin_once(self.node)
89             if not self.called_positionchanger_service:
90                 future = move_position_client.call_async(request)
91                 future.add_done_callback(self.move_position_callback)
92             elif not self.received_failsafe_callback:
93                 continue
94             else:
95                 break
96         self.assertTrue(self.received_failsafe_callback,
97                         "Failsafe callback was not received!")
98         self.assertTrue(self.called_positionchanger_service,
99                         "MovePosition service was not called!")
100     finally:
101         self.node.destroy_client(move_position_client)
102         self.node.destroy_subscription(failsafe_subscriber)
103
104     def test_positionchanger_lidar_stops(self, positionchanger_node, proc_output):
105         self.node.get_logger().info("STARTING TEST test_positionchanger_lidar_stops")
106         self.received_failsafe_callback = False
107         self.called_positionchanger_service = False
108         failsafe_subscriber = self.node.create_subscription(
109             FailsafeMsg, '/drone/failsafe', self.failsafe_callback, 10)
110         lidar_publisher = self.node.create_publisher(
111             LidarReading, '/drone/object_detection', 10)
112         move_position_client = self.node.create_client(
113             MovePosition, '/drone/move_position')
114
115         while not move_position_client.wait_for_service(timeout_sec=1.0):
116             self.node.get_logger().info('move_position service not available, waiting again...')
117
118         request = MovePosition.Request()
119         request.front_back = 1.0
120         request.left_right = 1.0
121         request.up_down = 1.0
122         request.angle = 1.0
123
124         lidar_msg = LidarReading()
125         lidar_msg.sensor_1 = 2.0
126         lidar_msg.sensor_2 = 2.0
127         lidar_msg.sensor_3 = 2.0
128         lidar_msg.sensor_4 = 2.0
129         lidar_msg.imu_data = [1.0, 1.0, 1.0, 1.0]
130         sent_lidar_msg = False
131
132         # wait for nodes to become active
133         time.sleep(3)
134
135         # wait 5 seconds for the failsafe to trigger
136         wait_time = time.time() + 5.0
137         end_time = time.time() + 10.0
138         try:
139             self.node.get_logger().info('STARTING WHILE LOOP')
140             while time.time() < end_time:
141                 rclpy.spin_once(self.node, timeout_sec=0.1)
142                 if not sent_lidar_msg:
143                     lidar_publisher.publish(lidar_msg)
144                     sent_lidar_msg = True
145                 # wait 5 seconds before sending the move_position request
146                 if not self.called_positionchanger_service and time.time() > wait_time:
147                     self.node.get_logger().info('Sending move_position request')
148                     future = move_position_client.call_async(request)
149                     future.add_done_callback(self.move_position_callback)
150                 elif not self.received_failsafe_callback:
151                     continue
152             self.node.get_logger().info('END OF WHILE LOOP')
153             self.assertTrue(self.called_positionchanger_service,
154                             "MovePosition service was not called!")
155             self.assertTrue(self.received_failsafe_callback,
156                             "Failsafe was not activated!")
157         finally:
158             self.node.get_logger().info('Cleaning up')
159             self.node.destroy_client(move_position_client)
160             self.node.destroy_subscription(failsafe_subscriber)
161             self.node.destroy_publisher(lidar_publisher)
162

```

```

ubuntu@ubuntu:~/ros2_ws$ launch test src/drone_controls/test/test_positionchanger.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2023-06-08-00-09-49-715475-ubuntu-4951
[INFO] [launch]: Default logging verbosity is set to INFO
test_positionchanger_lidar_stops (test_positionchanger.TestPositionChanger) ... [INFO] [position_changer-1]: proces
[INFO] [failsafe-2]: process started with pid [4959]
[INFO] [px4_controller-3]: process started with pid [4961]
[INFO] [heartbeat-4]: process started with pid [4963]
[heartbeat-4] [INFO] [1686175789.978004927] [heartbeat]: initialized service /drone/set vehicle control
[position_changer-1] [INFO] [1686175789.996583051] [position_changer]: waiting for trajectory service...
[heartbeat-4] [INFO] [1686175790.009789477] [heartbeat]: done initializing at -428441392 seconds. Sending heartbeat
[position_changer-1] [INFO] [1686175790.049612194] [position_changer]: waiting for attitude service...
[position_changer-1] [INFO] [1686175790.049838031] [position_changer]: waiting for vehicle control service...
[position_changer-1] [INFO] [1686175790.049955756] [position_changer]: waiting for failsafe service...
[px4_controller-3] [INFO] [1686175790.061102073] [px4_controller]: finished initializing at 1686175790.061096 secon
[INFO] [1686175790.353910840] [test_positionchanger]: STARTING TEST test positionchanger lidar stops
[px4_controller-3] [INFO] [1686175803.263535143] [px4_controller]: Failsafe enabled, landing...
[px4_controller-3] [INFO] [1686175803.363589509] [px4_controller]: Failsafe enabled, landing...
[INFO] [1686175803.437769257] [test_positionchanger]: END OF WHILE LOOP
[INFO] [1686175803.445883762] [test_positionchanger]: Cleaning up
ok
test_positionchanger_no_lidar_data (test_positionchanger.TestPositionChanger) ... [px4_controller-3]
landing...
[px4_controller-3] [INFO] [1686175803.563609818] [px4_controller]: Failsafe enabled, landing...
[px4_controller-3] [INFO] [1686175803.663595783] [px4_controller]: Failsafe enabled, landing...
[px4_controller-3] [INFO] [1686175803.763622330] [px4_controller]: Failsafe enabled, landing...
[position_changer-1] [WARN] [1686175803.817488940] [position_changer]: Lidar not sending messages,
[failsafe-2] [INFO] [1686175803.822068007] [failsafe]: Failsafe triggered! Message: No healthy contr
ok

[failsafe-2] [INFO] [1686175803.826091420] [failsafe]: Publishing failsafe message
-----
Ran 2 tests in 14.128s

```

Figuur B.5: Output integratietesten POS.I.1 en POS.I.2.

B.6: Integratietest PositionChanger en objectdetectie

De code voor test POS.I.3 is hieronder te zien. In de code wordt de output van de PositionChanger telkens gecontroleerd om te kijken of de drone bewogen wordt in de andere richting dan waar zich een object bevindt. De output wordt gecontroleerd omdat de drone niet daadwerkelijk aangestuurd kan worden voor deze test, omdat dat gevaarlijke situaties kan veroorzaken, mocht de test niet slagen. Het resultaat van de test is te zien in Figuur B.6. Hierin is te zien dat alle testcases geslaagd zijn door de OK onderaan de output.

```
1 import os
2 import unittest
3
4 import launch
5 import launch_ros
6 import launch_ros.actions
7 import launch_testing.actions
8 import pytest
9 import rclpy
10 import time
11
12 from drone_services.srv import MovePosition
13 from drone_services.msg import LidarReading
14
15 @pytest.mark.rostest
16 def generate_test_description():
17     file_path = os.path.dirname(__file__)
18     # device under test
19     positionchanger_node = launch_ros.actions.Node(
20         package='drone_controls', executable='position_changer')
21     failsafe_node = launch_ros.actions.Node(
22         package='failsafe', executable='failsafe')
23     px4_controller_node = launch_ros.actions.Node(
24         package='px4_connection', executable='px4_controller')
25     heartbeat_node = launch_ros.actions.Node(
26         package='px4_connection', executable='heartbeat')
27
28     return (
29         launch.LaunchDescription([
30             positionchanger_node,
31             failsafe_node,
32             px4_controller_node,
33             heartbeat_node,
34             launch_testing.actions.ReadyToTest(),
35         ]),
36         {
37             'positionchanger_node': positionchanger_node,
38             'failsafe_node': failsafe_node,
39             'px4_controller_node': px4_controller_node,
40             'heartbeat_node': heartbeat_node
41         }
42     )
43
44 class TestPositionChanger(unittest.TestCase):
45
46     @classmethod
47     def setUpClass(cls):
48         rclpy.init()
49
50     @classmethod
51     def tearDownClass(cls):
52         rclpy.shutdown()
53
54     def setUp(self):
55         self.node = rclpy.create_node('test_positionchanger')
56         self.called_positionchanger_service = False
57         self.received_failsafe_callback = False
58         self.lidar_publisher = self.node.create_publisher(
59             LidarReading, '/drone/object_detection', 10)
60         self.move_position_client = self.node.create_client(
61             MovePosition, '/drone/move_position')
62         while not self.move_position_client.wait_for_service(timeout_sec=1.0):
63             self.node.get_logger().info('move_position service not available, waiting again...')
64         self.request = MovePosition.Request()
65
66     def tearDown(self):
67         self.node.destroy_client(self.move_position_client)
68         self.node.destroy_publisher(self.lidar_publisher)
69         self.node.destroy_node()
70
71     def toRadians(self, degrees) -> float:
72         return degrees * 3.14159265359 / 180
73
74     def move_position_callback(self, future):
75         self.node.get_logger().info("Callback called")
76         self.called_positionchanger_service = True
77
78     def test_positionchanger_lidar_moves_away_front(self, proc_output):
79         self.node.get_logger().info("starting front test")
80         self.request.front_back = 1.0
81         self.request.left_right = 0.0
82         self.request.up_down = 0.0
```

```

83     self.request.angle = 0.0
84
85     lidar_msgs_sent = 0
86
87     lidar_msg = LidarReading()
88     lidar_msg.sensor_1 = 0.5 # front right
89     lidar_msg.sensor_2 = 2.0 # front left
90     lidar_msg.sensor_3 = 2.0 # rear left
91     lidar_msg.sensor_4 = 2.0 # rear right
92     lidar_msg.imu_data = [1.0, 1.0, 1.0, 1.0]
93     end_time = time.time() + 10.0
94
95     while time.time() < end_time:
96         rclpy.spin_once(self.node, timeout_sec=0.1)
97         self.lidar_publisher.publish(lidar_msg)
98         lidar_msgs_sent += 1
99         if (lidar_msgs_sent == 10):
100             lidar_msg.sensor_1 = 1.0
101             lidar_msg.sensor_2 = 0.3
102         elif (lidar_msgs_sent == 20):
103             break
104         if not self.called_positionchanger_service:
105             future = self.move_position_client.call_async(self.request)
106             future.add_done_callback(self.move_position_callback)
107     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention front: -0.5", 'position_changer-1')
108     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention front: -0.7", 'position_changer-1')
109
110 def test_positionchanger_lidar_moves_away_right(self, proc_output):
111     self.node.get_logger().info("starting right test")
112     self.request.front_back = 0.0
113     self.request.left_right = 1.0
114     self.request.up_down = 0.0
115     self.request.angle = 0.0
116
117     lidar_msgs_sent = 0
118
119     lidar_msg = LidarReading()
120     lidar_msg.sensor_1 = 0.4 # front right
121     lidar_msg.sensor_2 = 2.0 # front left
122     lidar_msg.sensor_3 = 2.0 # rear left
123     lidar_msg.sensor_4 = 2.0 # rear right
124     lidar_msg.imu_data = [1.0, 1.0, 1.0, 1.0]
125     end_time = time.time() + 10.0
126
127     while time.time() < end_time:
128         rclpy.spin_once(self.node, timeout_sec=0.1)
129         self.lidar_publisher.publish(lidar_msg)
130         lidar_msgs_sent += 1
131         if (lidar_msgs_sent == 10):
132             lidar_msg.sensor_1 = 2.0
133             lidar_msg.sensor_4 = 0.29
134         elif (lidar_msgs_sent == 20):
135             break
136         if not self.called_positionchanger_service:
137             future = self.move_position_client.call_async(self.request)
138             future.add_done_callback(self.move_position_callback)
139     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention right: -0.6", 'position_changer-1')
140     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention right: -0.71", 'position_changer-1')
141
142 def test_positionchanger_lidar_moves_away_left(self, proc_output):
143     self.node.get_logger().info("starting left test")
144     self.request.front_back = 0.0
145     self.request.left_right = -1.0
146     self.request.up_down = 0.0
147     self.request.angle = 0.0
148
149     lidar_msgs_sent = 0
150
151     lidar_msg = LidarReading()
152     lidar_msg.sensor_1 = 2.0 # front right
153     lidar_msg.sensor_2 = 0.11 # front left
154     lidar_msg.sensor_3 = 2.0 # rear left
155     lidar_msg.sensor_4 = 2.0 # rear right
156     lidar_msg.imu_data = [1.0, 1.0, 1.0, 1.0]
157     end_time = time.time() + 10.0
158
159     while time.time() < end_time:
160         rclpy.spin_once(self.node, timeout_sec=0.1)
161         self.lidar_publisher.publish(lidar_msg)
162         lidar_msgs_sent += 1
163         if (lidar_msgs_sent == 10):
164             lidar_msg.sensor_2 = 2.0
165             lidar_msg.sensor_3 = 0.78
166         elif (lidar_msgs_sent == 20):
167             break
168         if not self.called_positionchanger_service:
169             future = self.move_position_client.call_async(self.request)
170             future.add_done_callback(self.move_position_callback)
171     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention left: 0.89", 'position_changer-1')
172     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention left: 0.22", 'position_changer-1')
173
174 def test_positionchanger_lidar_moves_away_back(self, proc_output):
175     self.node.get_logger().info("starting back test")
176     self.request.front_back = -1.0
177     self.request.left_right = 0.0
178     self.request.up_down = 0.0
179     self.request.angle = 0.0
180
181     lidar_msgs_sent = 0

```

```

182
183     lidar_msg = LidarReading()
184     lidar_msg.sensor_1 = 2.0 # front right
185     lidar_msg.sensor_2 = 2.0 # front left
186     lidar_msg.sensor_3 = 0.36 # rear left
187     lidar_msg.sensor_4 = 2.0 # rear right
188     lidar_msg.imu_data = [1.0, 1.0, 1.0, 1.0]
189     #this test case is carried out first, so wait for nodes to start
190     time.sleep(5)
191     end_time = time.time() + 20.0
192
193     while time.time() < end_time:
194         rclpy.spin_once(self.node, timeout_sec=0.1)
195         self.lidar_publisher.publish(lidar_msg)
196         lidar_msgs_sent += 1
197         if (lidar_msgs_sent == 10):
198             lidar_msg.sensor_3 = 2.0
199             lidar_msg.sensor_4 = 0.12
200         elif (lidar_msgs_sent == 20):
201             break
202         if not self.called_positionchanger_service:
203             future = self.move_position_client.call_async(self.request)
204             future.add_done_callback(self.move_position_callback)
205     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention back: 0.64", 'position_changer-1')
206     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention back: 0.88", 'position_changer-1')
207
208     def test_positionchanger_lidar_moves_away_still(self, proc_output):
209         self.node.get_logger().info("starting back test")
210         self.request.front_back = 0.0
211         self.request.left_right = 0.0
212         self.request.up_down = 0.0
213         self.request.angle = 0.0
214
215         lidar_msgs_sent = 0
216
217         lidar_msg = LidarReading()
218         lidar_msg.sensor_1 = 2.0 # front right
219         lidar_msg.sensor_2 = 2.0 # front left
220         lidar_msg.sensor_3 = 2.0 # rear left
221         lidar_msg.sensor_4 = 0.12 # rear right
222         lidar_msg.imu_data = [1.0, 1.0, 1.0, 1.0]
223         end_time = time.time() + 10.0
224
225         while time.time() < end_time:
226             rclpy.spin_once(self.node, timeout_sec=0.1)
227             self.lidar_publisher.publish(lidar_msg)
228             lidar_msgs_sent += 1
229             if (lidar_msgs_sent == 10):
230                 lidar_msg.sensor_4 = 2.0
231                 lidar_msg.sensor_3 = 0.36
232             elif (lidar_msgs_sent == 20):
233                 break
234             if not self.called_positionchanger_service:
235                 future = self.move_position_client.call_async(self.request)
236                 future.add_done_callback(self.move_position_callback)
237     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention back: 0.64", 'position_changer-1')
238     launch_testing.asserts.assertInStderr(proc_output, "Collision prevention back: 0.88", 'position_changer-1')

```

```

ubuntu@ubuntu:~/ros2_ws$ launch test src/drone_controls/test/test_positionchanger_lidar.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2023-06-12-16-58-14-038291-ubuntu-9841
[INFO] [launch]: Default logging verbosity is set to INFO
test_positionchanger_lidar_moves_away_back (test_positionchanger_lidar.TestPositionChanger) ... [INFO] [position_changer-1]: process started with pid [9847]
[INFO] [failsafe-2]: process started with pid [9849]
[INFO] [px4_controller-3]: process started with pid [9851]
[INFO] [heartbeat-4]: process started with pid [9853]
[position_changer-1] [INFO] [1686581894.266166917] [position_changer]: waiting for trajectory service...
[heartbeat-4] [INFO] [1686581894.300080745] [heartbeat]: initialized service /drone/set_vehicle_control
[heartbeat-4] [INFO] [1686581894.312469790] [heartbeat]: done initializing at -658313008 seconds. Sending heartbeat...
[position_changer-1] [INFO] [1686581894.418805670] [position_changer]: waiting for attitude service...
[position_changer-1] [INFO] [1686581894.419666705] [position_changer]: waiting for vehicle control service...
[position_changer-1] [INFO] [1686581894.420359981] [position_changer]: waiting for failsafe service...
[px4_controller-3] [INFO] [1686581894.425314766] [px4_controller]: finished initializing at 1686581894.425308 seconds.
[INFO] [1686581894.593441464] [test_positionchanger]: starting back test
[failsafe-2] [INFO] [1686581895.110499435] [failsafe]: Failsafe node started
[position_changer-1] [INFO] [1686581895.117944565] [position_changer]: waiting for arm service...
[heartbeat-4] [INFO] [1686581895.118997119] [heartbeat]: set control mode to 16
[heartbeat-4] [INFO] [1686581895.119167026] [heartbeat]: set control mode to velocity
[heartbeat-4] [INFO] [1686581895.119223062] [heartbeat]: publishing new control mode 2
[px4_controller-3] [INFO] [1686581895.119761820] [px4_controller]: Got valid control mode
[px4_controller-3] [INFO] [1686581895.119888376] [px4_controller]: Control mode set to 2
[position_changer-1] [INFO] [1686581895.125127419] [position_changer]: Vehicle control mode set result: 1
[position_changer-1] [INFO] [1686581899.700933935] [position_changer]: Collision prevention back: 0.640000
[position_changer-1] [INFO] [1686581899.701150176] [position_changer]: Collision prevention left: 0.640000
[position_changer-1] [INFO] [1686581899.701438823] [position_changer]: Incoming request
[position_changer-1] front back: -1.000000
[position_changer-1] left right: 0.000000
[position_changer-1] up down: 0.000000
[position_changer-1] angle: 0.000000
[position_changer-1] [INFO] [1686581900.124000300] [position_changer]: Collision prevention back: 0.640000
[position_changer-1] [INFO] [1686581908.124294382] [position_changer]: Collision prevention left: 0.640000
[position_changer-1] [INFO] [1686581908.224981719] [position_changer]: Collision prevention back: 0.640000
ok
[position_changer-1] [INFO] [1686581908.225181571] [position_changer]: Collision prevention left: 0.640000
-----
Ran 5 tests in 14.202s
OK

```

Figuur B.6: Resultaat test POS.I.3

B.7: ApiListener integratietest

De code voor test API.I.1 is hieronder te zien. De output van deze test is te zien in Figuur B.7. Hierin is te zien dat de test slaagt door de OK aan het eind van de output.

```
239 import os
240 import sys
241 import unittest
242 import time
243
244 import launch
245 import launch_ros
246 import launch_ros.actions
247 import launch_testing.actions
248 import pytest
249 import rclpy
250
251 from drone_services.msg import FailsafeMsg
252 from drone_services.msg import DroneStatus
253
254 @pytest.mark.rostest
255 def generate_test_description():
256     file_path = os.path.dirname(__file__)
257     api_listener_node = launch_ros.actions.Node(
258         executable=sys.executable,
259         arguments=[os.path.join(
260             file_path, '..', 'api_communication', 'api_listener.py')],
261         additional_env={'PYTHONUNBUFFERED': '1'})
262
263     failsafe_node = launch_ros.actions.Node(
264         package='failsafe', executable='failsafe')
265     camera_node = launch_ros.actions.Node(
266         package='camera', executable='camera_controller')
267     position_changer_node = launch_ros.actions.Node(
268         package='drone_controls', executable='position_changer')
269     px4_controller_node = launch_ros.actions.Node(
270         package='px4_connection', executable='px4_controller')
271     heartbeat_node = launch_ros.actions.Node(
272         package='px4_connection', executable='heartbeat')
273
274     return (
275         launch.LaunchDescription([
276             api_listener_node,
277             failsafe_node,
278             camera_node,
279             position_changer_node,
280             px4_controller_node,
281             heartbeat_node,
282             launch_testing.actions.ReadyToTest(),
283         ]),
284         {
285             'api_listener_node': api_listener_node,
286             'failsafe_node': failsafe_node,
287             'camera_node': camera_node,
288             'position_changer_node': position_changer_node,
289             'px4_controller_node': px4_controller_node,
290             'heartbeat_node': heartbeat_node
291         }
292     )
293
294 class ApiListenerTest(unittest.TestCase):
295
296     @classmethod
297     def setUpClass(cls):
298         rclpy.init()
299
300     @classmethod
301     def tearDownClass(cls):
302         rclpy.shutdown()
303
304     def setUp(self):
305         self.node = rclpy.create_node('test_api_listener')
306         self.published_battery_status = False
307         self.received_failsafe_callback = False
308
309     def tearDown(self):
310         self.node.destroy_node()
311
312     def failsafe_callback(self, msg):
313         self.assertTrue(msg.enabled, "Failsafe was not enabled!")
314         self.assertTrue("Battery level too low! Failsafe enabled to prevent damage to battery" in msg.msg, "Failsafe message was not correct!")
315         self.received_failsafe_callback = True
316
317     def status_callback(self, msg):
318         self.node.get_logger().info("Received status callback " + str(msg))
319
320     def test_api_listener_battery(self, api_listener_node, proc_output):
321         failsafe_subscriber = self.node.create_subscription(FailsafeMsg, '/drone/failsafe', self.failsafe_callback, 10)
322         drone_status_publisher = self.node.create_publisher(DroneStatus, '/drone/status', 10)
323         msg = DroneStatus()
324         msg.battery_percentage = 10.0
325         msg.armed = True
326         msg.height = 10.0
327         msg.control_mode = "attitude"
```

```

328     msg.cpu_usage = 10.0
329     msg.route_setpoint = 0
330     msg.position = [0.0,0.0,0.0]
331     msg.velocity = [0.0,0.0,0.0]
332
333     time.sleep(5) # wait for nodes to start
334     self.node.get_logger().info("Starting publishing")
335     end_time = time.time() + 10.0
336
337     try:
338         while time.time() < end_time:
339             rclpy.spin_once(self.node, timeout_sec=0.1)
340             # self.node.get_logger().info("publishing message")
341             drone_status_publisher.publish(msg)
342             if self.received_failsafe_callback:
343                 break
344             self.assertTrue(self.received_failsafe_callback, "Failsafe was not enabled!")
345     finally:
346         self.node.destroy_subscription(failsafe_subscriber)
347         self.node.destroy_publisher(drone_status_publisher)

```

```

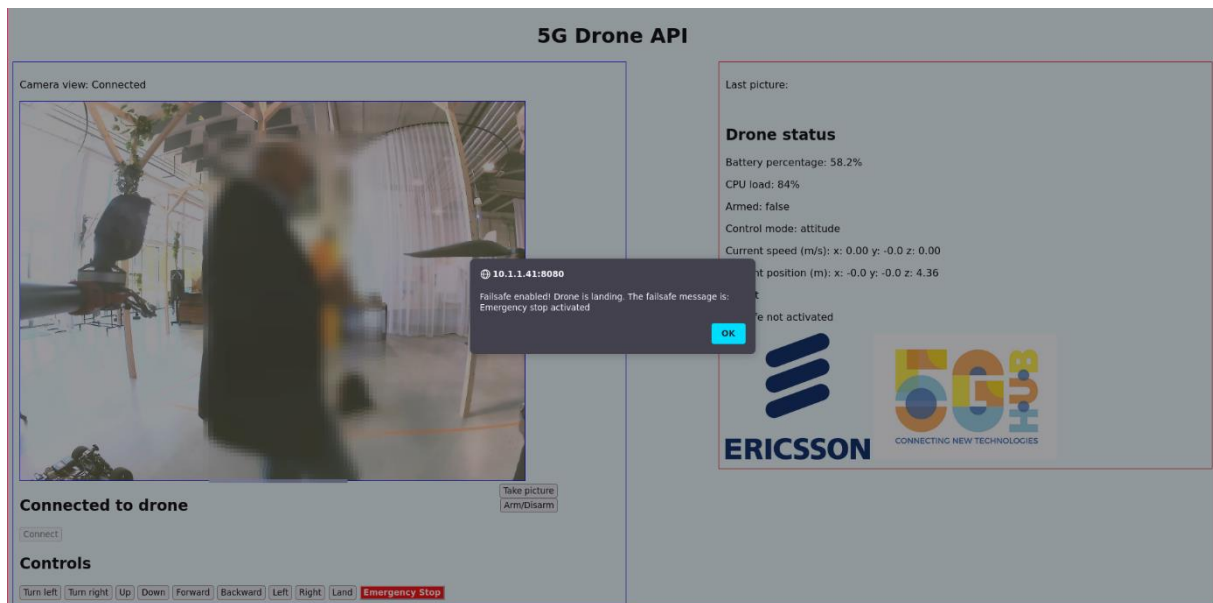
ubuntu@ubuntu:~/ros2_ws$ launch test src/api_communication/test/test_api_listener.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2023-06-09-19-40-48-667724-ubuntu-75129
[INFO] [launch]: Default logging verbosity is set to INFO
test_api_listener battery (test_api_listener.ApiListenerTest) ... [INFO] [python3-1]: process started with pid [75135]
[INFO] [failsafe-2]: process started with pid [75137]
[INFO] [camera_controller-3]: process started with pid [75139]
[INFO] [position_changer-4]: process started with pid [75141]
[INFO] [px4_controller-5]: process started with pid [75143]
[INFO] [heartbeat-6]: process started with pid [75145]
[heartbeat-6] [INFO] [1686332448.995950973] [heartbeat]: initialized service /drone/set_vehicle_control
[heartbeat-6] [INFO] [1686332449.014057489] [heartbeat]: done initializing at -688881456 seconds. Sending heartbeat...
[px4_controller-5] [INFO] [1686332449.038437724] [px4_controller]: finished initializing at 1686332449.038428 seconds.
[position_changer-4] [INFO] [1686332449.040053158] [position_changer]: waiting for trajectory service...
[position_changer-4] [INFO] [1686332449.040485200] [position_changer]: waiting for attitude service...
[position_changer-4] [INFO] [1686332449.040573513] [position_changer]: waiting for vehicle control service...
[position_changer-4] [INFO] [1686332449.040620641] [position_changer]: waiting for failsafe service...
[failsafe-2] [INFO] [1686332449.953763449] [failsafe]: Failsafe node started
[position_changer-4] [INFO] [1686332449.962553132] [position_changer]: waiting for arm service...
[heartbeat-6] [INFO] [1686332449.964194546] [heartbeat]: set control mode to 16
[heartbeat-6] [INFO] [1686332449.964323153] [heartbeat]: set control mode to velocity
[heartbeat-6] [INFO] [1686332449.964366281] [heartbeat]: publishing new control mode 2
[px4_controller-5] [INFO] [1686332449.964874322] [px4_controller]: Got valid control mode
[px4_controller-5] [INFO] [1686332449.965000374] [px4_controller]: Control mode set to 2
[position_changer-4] [INFO] [1686332449.965066538] [position_changer]: Vehicle control mode set result: 1
[python3-1] [INFO] [1686332450.196349434] [api_listener]: ApiListener node started
[camera_controller-3] [INFO] [1686332450.790454486] [camera_controller]: Camera controller started. Waiting for service call...
[python3-1] [INFO] [1686332451.268000251] [api_listener]: Starting API
[python3-1] [INFO] [1686332451.551773593] [api_listener]: API started on port 9001
[INFO] [1686332454.245218184] [test_api_listener]: Starting publishing
[python3-1] [INFO] [1686332454.624465895] [api_listener]: Enabling failsafe
[failsafe-2] [INFO] [1686332454.627615616] [failsafe]: Failsafe triggered! Message: Battery level too low! Failsafe enabled to prevent damage to battery (10.0%)
ok
[failsafe-2] [INFO] [1686332454.630384903] [failsafe]: Publishing failsafe message
-----
Ran 1 test in 5.968s
[px4_controller-5] [INFO] [1686332454.638381738] [px4_controller]: Failsafe enabled, landing...
OK

```

Figur B.7: Output ApiListener test

B.8: Output failsafe aanzetten door noodstop op GUI

Het resultaat van test API.F.5 voor de melding over dat de failsafe is aangezet is te zien in Figuur B.8. In Figuur B.9 is te zien dat hierna de failsafe status is geüpdatet naar *ENABLED*.



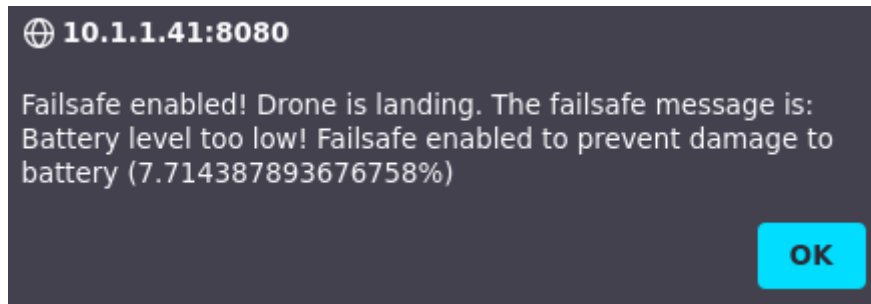
Figuur B.8: melding failsafe aanzetten door noodstop



Figuur B.9: failsafe aangezet nadat noodstop is ingedrukt

B.9: Output failsafe door te lage batterij

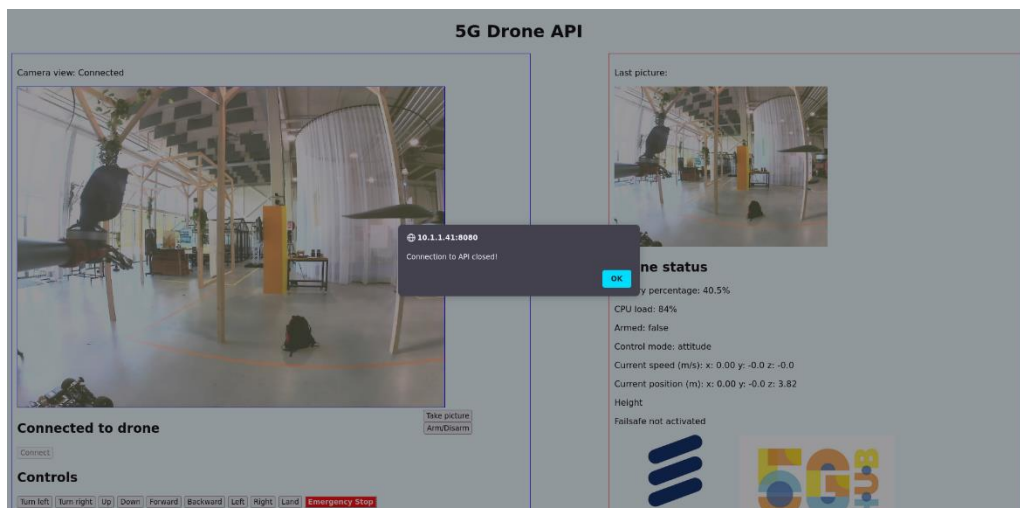
Het resultaat van test API.F.3 voor de weergegeven melding is te zien in Figuur B.10.



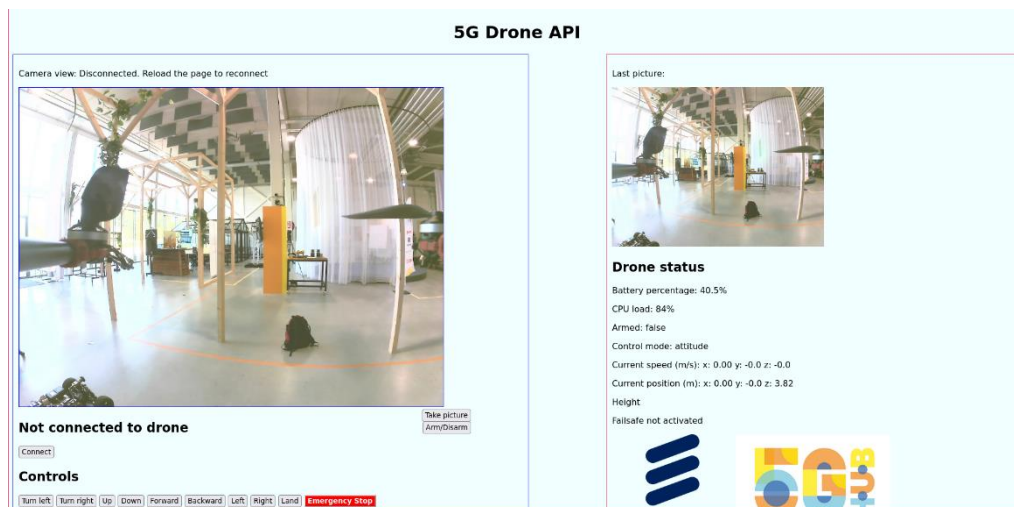
Figuur B.10: melding voor te lage batterij op GUI

B.10: Output wegvallen verbinding met API

Het resultaat van test API.F.2 voor de weergegeven melding is te zien in Figuur B.11. In Figuur B.12 is de veranderde status over de verbinding voor deze test te zien.



Figuur B.11: Weergegeven melding voor wegvallen van communicatie



Figuur B.12: Status voor wegvallen verbinding met API

Bijlage C: Overzicht video's

Omdat dit verslag een aantal video's bevat die verschillende functionaliteiten van de drone laten zien, is in Tabel C.1 een overzicht van deze video's opgenomen.

Tabel C.1: Overzicht gemaakte video's

Video	Beschrijving	Link
1	Testvlucht in de 5G Hub met besturing op basis van houding	MyMedia
2	Testvlucht thuis met besturing op basis van houding	MyMedia
3	Marvelmind beacons positionering test	MyMedia
4	Werking van aansturen drone met PositionChanger in simulator.	MyMedia
5	Demonstratie lage framerate bij versturen video	MyMedia
6	Demonstratie vermogen van vliegen met 5 propellers	MyMedia
7	Demonstratie vliegen met GUI	MyMedia
8	Functionele test opstarten drone en verbinden met GUI	MyMedia

Register

- ArduPilot
 - Een autopilot software pakket voor drones. 15, 16, 17, 21
- Beamforming
 - Een techniek van signaalbewerking waarbij een draadloos signaal directer gericht kan worden op een apparaat. 45, 46
- Beyond Visual Line of Sight
 - Vliegen zonder dat de drone zichtbaar is voor de bestuurder. 8
- DDS
 - Data Distribution Service. Dit is een standaard voor data-georiënteerde verbindingen. 16, 17, 19, 21, 26, 32, 44
- EEPROM
 - Een vorm van geheugen dat gebruikt wordt om kleine hoeveelheden data op te slaan. 13
- ESC
 - Electronic Speed Controller 9, 13, 21, 27, 28
- Fast-DDS
 - Een versie van DDS die met RTPS gebruikt kan worden. 26
- FastRTPS-Gen
 - Een programma om code te genereren voor de koppeling tussen Fast-DDS en microRTPS. 27
- flight computer
 - Computer die op een drone verschillende sensoren uitleest en op basis hiervan de flight controller aanstuurt. 7, 9, 10, 11, 12, 13, 14, 15, 16, 18, 21, 22, 26, 27, 30, 31, 32, 42, 44, 45, 56, 57
- flight controller
 - Controller op een drone die zorgt dat de drone stabiel vliegt door de motoren aan te sturen ... 7, 9, 10, 11, 12, 13, 15, 16, 18, 21, 22, 24, 27, 28, 29, 30, 31, 32, 34, 37, 38, 39, 42, 44, 45, 56
- FMU
 - Flight Management Unit. Dit is de microprocessor van de flight controller. 13
- GPIO
 - General Purpose Input/Output. Dit is een hardwareverbinding voor communicatie. 16, 18, 20, 21, 22, 44, 45
- Ground Control Station
 - Een computer of stuk software waarmee een flight controller geconfigureerd kan worden. 15
- IMU
 - Inertial Measurement Unit. Dit zijn sensoren die de snelheid en positie van een drone kunnen meten. 13, 14, 42
- Long polling
 - Vergelijkbaar met polling, maar hierbij stuurt de server pas een antwoord naar de client als data beschikbaar is 56
- MAVLink
 - Protocol voor communicatie tussen de flight computer en flight controller. . 15, 16, 17, 32, 35, 71
- MAVROS
 - Een package voor ROS 1 die ROS topics om kan zetten naar MAVLink berichten en andersom. .. 16, 17, 21, 71
- Micro XRCE-DDS
 - Een implementatie van XRCE-DDS voor apparaten met weinig resources. 70
- microRTPS
 - Een versie van RTPS die in de 1.13 versie van PX4 gebruikt wordt. 17, 21, 26
- OpenSSL
 - Een verzameling programma's voor cryptografie en beveiligde communicatie. 26, 27
- OS
 - Operating System 15, 18, 19, 20, 21
- PDB
 - Power Distribution Board 21, 22
- PID-regelaar
 - Een regelaar die een systeem soepel naar een bepaald punt kan bewegen. 14
- Pixhawk 4 Mini
 - De flight controller die voor de drone wordt gebruikt 9, 13
- Polling
 - Een manier van HTTP communicatie waarbij de client regelmatig een bericht naar de server stuurt om te vragen of nieuwe data beschikbaar is. 56, 73
- PWM
 - Pulse Width Modulation 13
- PX4
 - Autopilot pakket voor PixHawk flight controllers 15, 16, 17, 21, 22, 24, 25, 26, 27, 30, 31, 32, 36, 41
- PX4-ROS 2 Bridge
 - Een verzameling ROS 2 packages om communicatie tussen PX4 en ROS 2 mogelijk te maken. 17, 21, 22, 27
- QGroundControl
 - Ground Control software om de flight controller te configureren. 27, 32, 70
- quaternion
 - Een manier om 3D vectoren weer te geven met imaginaire getallen. Wordt gebruikt voor het aanpassen van de houding in de flight controller. 38
- RCID
 - Remote Controller Indoor Drone. De drone die tijdens dit project ontwikkeld wordt. 2, 28, 70
- ROS
 - Robot Operating System. 12, 16, 17, 19, 20, 21, 22, 23, 27, 32, 33, 43, 44, 46, 55, 74
- ROS 2
 - De nieuwere, verbeterde versie van ROS 16, 17, 19, 20, 21, 22, 27, 32, 33, 43, 44, 55, 74
- RTPS
 - Real-Time Publish/Subscribe. Dit is een interface waardoor ROS 2 topics omgezet kunnen naar uORB berichten. 17, 71
- SSE
 - Server-Sent Events. Dit is een HTTP standaard die het mogelijk maakt voor een client om automatisch updates van de server te ontvangen. 56, 70, 72
- UART
 - Universal Asynchronous Receiver/Transmitter. Dit is een serieel communicatieprotocol. 16, 22, 29, 45, 70

uORB

Een specificatie voor berichten die door de flight controller gebruikt wordt voor interne communicatie. 15, 16, 17, 72

VM

Virtual Machine 58, 67

Websockets

Een communicatieprotocol die tweewegcommunicatie over één TCP/IP verbinding mogelijk maakt. 56, 57, 59, 67

XRCE-DDS

DDS voor apparaten met weinig resources ...17, 21, 32