# Handleiding ROS2



Namen: Marc de Jong, Julian Kamminga, Michiel Koetje

Klas: 1f

Datum: 10-07-2023

Inhoud

[Handleiding ROS2 1](#_Toc140069459)

[1. Introductie ROS 2 4](#_Toc140069460)

[1.1 Wat is ROS2? 4](#_Toc140069461)

[2. Benodigdheden voor het werken met ROS2: 5](#_Toc140069462)

[3. Het installatieproces 6](#_Toc140069463)

[3.1 Het installeren van Ubuntu server op de Raspberry pi. 6](#_Toc140069464)

[3.2 Het opstarten en activeren van SSH (wifi-optie) 8](#_Toc140069465)

[3.2.1 De voorbereiding voor het opstarten als wifi niet is ingesteld: 8](#_Toc140069466)

[3.2.2 De voorbereiding voor het opstarten als wifi is ingesteld: 8](#_Toc140069467)

[3.2.3 Opstarten met behulp van een beeldscherm: 9](#_Toc140069468)

[3.2.4 Opstarten met SSH zonder WIFI ingesteld te hebben: 9](#_Toc140069469)

[3.2.5 Opstarten met SSH met Wifi. 10](#_Toc140069470)

[3.3 Het installeren van ROS 2 10](#_Toc140069471)

[3.3.1 Setup locale 10](#_Toc140069472)

[3.3.2 Setup source 10](#_Toc140069473)

[3.3.3 Installeer de ROS 2 packages 11](#_Toc140069474)

[3.4 Een eigen repository toevoegen om code te kunnen runnen en testen 11](#_Toc140069475)

[3.4.1 Aanmaken van een git repository 12](#_Toc140069476)

[3.4.2 Repository clonen op de Raspberry Pi 12](#_Toc140069477)

[3.4.3 Repository pull 12](#_Toc140069478)

[5. Uitleg ROS Protocol 13](#_Toc140069479)

[4.1 Nodes 13](#_Toc140069480)

[4.2 Topics 13](#_Toc140069481)

[4.3 Messages 13](#_Toc140069482)

[4.4 Services 14](#_Toc140069483)

[4.5 Actions 14](#_Toc140069484)

[4.6 Parameters 15](#_Toc140069485)

[6. Voorbeelden ROS 2 16](#_Toc140069486)

[5.1 Creatie van een node 16](#_Toc140069487)

[5.2 simpele logica binnen een node 17](#_Toc140069488)

[5.2.1 Logica van een node 17](#_Toc140069489)

[5.2.2 creatie van een subscriber node 18](#_Toc140069490)

[5.2.3 Het beheren van services met ROS2 18](#_Toc140069491)

[5. Je robot laten bewegen. 20](#_Toc140069492)

[5.1 Instellen van de software. 20](#_Toc140069493)

[5.2 Simpele beweging voor een korte tijd 21](#_Toc140069494)

[5.3 Snelheid van de motoren aanpassen 23](#_Toc140069495)

[5.3.1 Toepassing op Python code 23](#_Toc140069496)

[5.3.2 voorbeeld gebruik PWM 24](#_Toc140069497)

[5.4 Implementatie in ROS2 25](#_Toc140069498)

[6. Implementatie sensor 27](#_Toc140069499)

[6 Bronnenlijst 29](#_Toc140069500)

# Introductie ROS 2

Hier wordt kort uitgelegd wat ROS2 is en waar het voor wordt gebruikt.

* 1. Wat is ROS2?

ROS2 is een tool die bestaat om ontwikkelaars te helpen bij het ontwikkelen en beheren van Robotica projecten en toepassingen. De tool zelf is een open source raamwerk met een verzameling aan softwarebibliotheken. De naam ROS is een afkorting voor ‘Robot Operating System’, ROS2 is dan ook de tweede versie van ROS. Het is ontworpen met het doel om de beperkingen van de originele ROS versie te omzeilen, en zo de ontwikkeling van complexere robotica systemen te ondersteunen.

* 1. Enkele belangrijke kenmerken van ROS 2

**Enkele belangrijke kenmerken van ROS 2 zijn:**

*Ondersteuning voor meerdere programmeertalen:*

ROS2 ondersteunt meerdere programmeertalen, waaronder C++ en Python.

Dit zorgt ervoor dat ROS2 in veel verschillende omstandigheden kan worden gebruikt, daarbovenop is python is een van de makkelijkere programmeertalen en kunnen beginners ook met ROS2 verder.

*De samenstelbaarheid/ modulariteit van ROS2:*

Met ROS2 kunnen de ontwikkelaars complexe systemen ontwerpen met behulp van herbruikbare componenten genaamd ‘nodes’. De aanpak die hierbij wordt gebruikt zorgt ervoor dat de code relatief makkelijk is te onderhouden en te hergebruiken is.

# Benodigdheden voor het werken met ROS2:

Om bezig te kunnen met ROS 2 zijn er een aantal dingen nodig. De meeste van deze dingen zijn handig om erbij te hebben als het niet naar wens verloopt. Enkele benodigdheden zijn:

* Raspberry Pi, met bijpassende voeding/kabel.
* Laptop of pc met beeldscherm
* Ubuntu of Windows OS.
* Beeldscherm met Hdmi aansluiting voor de Raspberry Pi (extern/niet de primaire monitor).
* Hdmi kabel/micro Hdmi.
* Toetsenbord.
* Ethernet kabel (optioneel maar aangeraden).
* Usb kabel (verschilt per Raspberry Pi).
* Micro SD kaart (minimaal 8 gb).
* Camjam edukit (optioneel maar word gebruikt voor het voorbeeld).
* Een stabiele internet verbinding voor zowel je pc en de Raspberry pi.

ROS 2 kan op verschillende modellen van de Raspberry Pi gedraaid worden, waaronder de Raspberry Pi 2, Pi 3, Pi 3B, Pi 3B+, Pi 4 en de Raspberry Pi Zero. Het belangrijk om te weten met welk type Raspberry je aan de slag gaat. De reden hiervoor is dat onderdelen op de Raspberry variëren en dus daarmee ook de benodigdheden anders kunnen zijn. De benodigdheden die in het voorgaande stuk zijn aangekaart werken gegarandeerd op de Raspberry Pi 2, Pi 3, Pi 3B en de 3B plus. Indien er een ouder model word gebruikt is het handig om van te voren te kijken hoe je bepaalde aansluitingen moet doen voordat er begonnen word met het werken met ROS2.

Verder is het aangeraden een losse computer erbij te hebben met een eigen beeldscherm. Deze computer zal in het proces nodig zijn om eventuele software te kunnen downloaden voor de Raspberry Pi en zoals later uitgelegd wordt kan de computer met behulp van SSH draadloze verbindingen maken zodat je op afstand ook kan werken aan de code en ROS 2.

De Software die nodig zal zijn zal stapsgewijs worden uitgelegd in deze handleiding.

# Het installatieproces

Het installatieproces Is ingedeeld in 4 stappen. Het volledige proces wordt uitgelegd in chronologische volgorde zodat de kans op fouten geminimaliseerd blijft.

## 3.1 Het installeren van Ubuntu server op de Raspberry pi.

Als eerste stap is het nodig om Ubuntu te installeren op de Raspberry Pi. Voor dit voorbeeld is gekozen voor Ubuntu Server 20.04.5 LTS (64-bit).

1. Download Raspberry pi Imager op je pc. Kies hierbij het besturingssysteem van de gebruikte pc voor de download. In het voorbeeld is het systeem uitgerust met Windows 11. De download link is te vinden op deze website: <https://www.raspberrypi.com/software/> .
2. Open het gedownloade programma. Hier kan een administrator melding bij komen, deze kan geaccepteerd worden. Doe de micro SD kaart die gebruikt gaat worden in de computer (als dat niet mogelijk is zorg voor een usb hub bijvoorbeeld deze: <https://www.bol.com/nl/nl/l/hubs-sd-kaartlezer/52299/66788/> . Nadat dit is gedaan, is het mogelijk in het programma de SD kaart te kiezen om daar het besturingssysteem op te zetten.
3. Selecteer bij “SELECTEER OS” het gewilde besturingssysteem. Voor deze handleiding wordt er rekening gehouden met het type Raspberry pi en wordt er gekozen voor Ubuntu server. Om Ubuntu te kunnen kiezen moet er genavigeerd worden naar: “Other general-purpose OS”. Daar kies je voor Ubuntu. Vervolgens is er een lijst met verschillende versies te vinden. Voor dit voorbeeld is gekozen voor: “Ubuntu Server 20.04.5 LTS (64-bit)” omdat deze handleiding gebruik maakt van de Raspberry pi 3 type b. Het is belangrijk dat bekend is wat voor CPU je hebt. Dit kan namelijk verschillen en dus is hierbij gekozen voor de 64-bit versie aangezien deze Raspberry Pi dat ondersteund.
4. Klik vervolgens op het tandwiel om nog wat aanpassingen te maken aan het besturingssysteem.
5. Voer de volgende stappen uit: 1. Schakel SSH in. Hierbij is belangrijk dat er een makkelijk te onthouden gebruikersnaam en wachtwoord gekozen word aangezien dit later belangrijk is. 2. (Optioneel) Activeer de wifi. Voor draadloze SSH-verbindingen kan wifi handig zijn, maar dit wordt later nog uitgelegd wanneer het besturingssysteem opgestart is.
6. Klik op opslaan.
7. Klik op Schrijf om het besturingssysteem op de SD kaart te zetten. Hierbij kan een melding opkomen dat het programma alle bestanden weggooit en alleen het besturingssysteem er op download. Als er nog belangrijke bestanden op de SD kaart hebt is het aangeraden om deze eerst ergens op de pc op te slaan.
8. Laat de download zijn gang gaan. Als er halverwege een error komt dan zou de gebruiker de stappen opnieuw moeten proberen vanaf het begin. Anders zie stap 9 en 10 voor een andere manier van het installeren.
9. Download de Ubuntu server image op je pc. De link voor het vinden van de image is <https://ubuntu.com/download/raspberry-pi>. Je kan op de download pagina kiezen tussen de 32 bit versie en de 64 bit versie van de imae, welke je kiest hangt van een paar factoren. Het model van je Raspberry pi is in deze keuze belangrijk, de modellen Raspberry Pi 3B, 3B+ en 4 zijn in staat om 64-bits software te draaien. Echter, oudere modellen zoals de Raspberry Pi 1 of Raspberry Pi Zero ondersteunen alleen 32-bits software. Denk ook aan het gebruik van de RAM, de 32 bit software kan maar 4gb RAM maximaal gebruiken. Het is daarom slim om te kiezen voor de 64 bit versie als de Pi 4gb of meer ter beschikking heeft.
10. Vervolgens moet de software image op de SD kaart worden geïnstalleerd, om dit proces te voltooien kunnen verschillenden programma’s gebruikt worden. In deze handleiding zal gebruik worden gemaakt van de Raspberry Pi imager.

Afbeelding met tekst, schermopname, logo

Automatisch gegenereerde beschrijvingVervolgens moet je in het programma zelf, de knop “choose OS” klikken en de ”use-custom” optie selecteren, dan selecteer je de software image die je gedownload hebt en laat de installer zijn gang gaan. Wanneer het proces compleet is staat Ubuntu server op de SD kaart.

Figuur 1 Raspberry Pi imager met OS en schijf geselecteerd

## 3.2 Het opstarten en activeren van SSH (wifi-optie)

Voor het opstarten van de Raspberry pi zijn er 3 mogelijkheden. Er is een manier om het op te starten met behulp van een beeldscherm en er is een manier om het met behulp van SSH te doen. Aangeraden is om eerst het te proberen met een beeldscherm en later met SSH, maar voor de duidelijkheid zijn er 2 lijsten met stappen gemaakt.

### 3.2.1 De voorbereiding voor het opstarten als wifi niet is ingesteld:

1. Wanneer de download klaar is, verwijder de sd kaart van de pc en druk de sd kaart in de Raspberry Pi.
2. Open op de pc de instellingen.
3. Navigeer naar netwerk en internet -> Geavanceerde netwerkinstellingen -> meer opties voor netwerkadapter.
4. Open de status van de Wi-Fi en klik vervolgens op eigenschappen.
5. Navigeer naar het kopje “Delen” en zet “Andere netwerkgebruikers mogen verbinding maken via de internetverbinding van deze computer” aan. Kies vervolgens eronder Ethernet en klik op ok.
6. Sluit een Ethernet kabel aan op de pc en verbind diezelfde kabel vervolgens met de Raspberry Pi.

### 3.2.2 De voorbereiding voor het opstarten als wifi is ingesteld:

1. Wanneer de download klaar is, verwijder de sd kaart van de pc en druk de sd kaart in de Raspberry Pi.
2. Afbeelding met tekst, schermopname, Lettertype, diagram

   Automatisch gegenereerde beschrijvingBekijk op de router of de Raspberry Pi automatisch een connectie maakt met de router. Daar kan je ook vinden wat het ip adres is. Als dit niet is gelukt probeer dan de voorbereiding voor het opstarten als wifi niet is ingesteld. Je kunt met de interface van de router verbinden door in een browser het IP adres van de router te plakken. Het IP adres van de PI zal te vinden zijn in de clients.

Figuur 2 voorbeeld van een router interface

### 3.2.3 Opstarten met behulp van een beeldscherm:

1. Sluit eerst de Raspberry Pi aan op een toetsenbord en een beeldscherm voordat je stroom op de Raspberry Pi zet.
2. Doe vervolgens de micro usb kabel in de Raspberry Pi om het apparaat op te starten.
3. Als dit correct gaat valt er te zien op het beeldscherm dat de Raspberry Pi aan gaat.
4. Log in met de gebruikersnaam en wachtwoord die bij de download is gekozen.
5. De Pi is nu gestart.

### 3.2.4 Opstarten met SSH zonder WIFI ingesteld te hebben:

1. Sluit een micro usb kabel aan om de Raspberry Pi op te starten.
2. Na een minuut geduld zou de Raspberry Pi opgestart moeten zijn en kan er met SSH verbonden kunnen worden.
3. Open op de pc een gekozen opdrachtpropt zoals cmd of windowns powershell.
4. Typ in de prompt:”arp -a”. Dit laat alle ip adressen zien die verbonden zijn met de pc.
5. Kijk bij “Interface {ip adress} --- 0x6 voor de ip adressen die verbonden zijn met behulp van Ethernet. Zoek hier dan het ip adres dat anders eindigt dan de veel voorkomende ip adressen.
6. Vul in de prompt in: “ssh (gekozen gebruikersnaam) @ (gevonden ip adres). Als dit lukt dan komt er een melding die geaccepteerd kan worden.
7. Als de connectie geweigerd wordt is er waarschijnlijk het verkeerde ip adres ingevuld. Om dit probleem op te lossen moeten de volgende stappen uitgevoerd worden
8. : “Opstarten met behulp van een beeldscherm”. Hierbij kan je dan wanneer je succesvol bent ingelogd het commando: hostname –I doen om het ip adress te kunnen achterhalen.
9. Vul nu het bij de download gekozen wachtwoord in om in te loggen op de Raspberry pi.
10. De Raspberry Pi is nu opgestart via SSH.

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

Figuur 3 ssh connectie prompt met fingerprint

### 3.2.5 Opstarten met SSH met Wifi.

1. Sluit een micro usb kabel aan om de Raspberry pi op te starten, na ongeveer een minuut zou het op moeten zijn gestart.
2. Op een opdrachtenprompt, bijvoorbeeld Windows powershell.
3. Aannemend dat het IP-adres al bekend is, vul de prompt: “ssh (gekozen gebruikersnaam) @ (ip adres) in. Als dit lukt dan komt er een melding waarbij ‘yes’ ingevuld kan worden.
4. Vul nu het gekozen wachtwoord in, als dit goed gaat is de Raspberry pi gestart via ssh.

## 3.3 Het installeren van ROS 2

Voor het installeren van ROS2 is het belangrijk dat alle stappen ervoor goed zijn uitgevoerd. Het belangrijkste is dat de Raspberry Pi internet heeft. Zonder internet kan de Raspberry pi niks downloaden. Voor dit voorbeeld wordt ROS2 Humble gedownload. Bij deze download wordt er gebruik gemaakt van deze website: <https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html> . Deze installatie kan even duren.

### 3.3.1 Setup locale

1. Check eerst of je internet hebt met behulp van: hostname –I of ip a.
2. Als je bij hostname een ip adres terugkrijgt dan heb je dus internet en kan je ROS 2 downloaden. Als je bij ip a een ip adres kan vinden onder die niet vergelijkbaar is met 127.0.0.1 dan ben je ook goed verbonden.
3. Als eerste moet je checken of de Raspberry de goede locale support(UTF-8). Het zou kunnen voorkomen dat je iets minimaals zoals POSIX. Om dit te kunnen checken vul je de volgende code in de terminal in:”locale”. Als er en\_US.UTF-8 staat kan de volgende stap overgeslagen worden.
4. Het instellen van locale word gedaan met behulp van de volgende code: sudo apt update && sudo apt install locales  
   sudo locale-gen en\_US en\_US.UTF-8  
   sudo update-locale LC\_ALL=en\_US.UTF-8 LANG=en\_US.UTF-8  
   export LANG=en\_US.UTF-8.
5. Check nu wederom of locale nu goed is ingesteld met “locale”.

### 3.3.2 Setup source

1. Voordat we de source kunnen instellen moeten we zeker weten dat de Ubuntu Universe repository is ingeschakeld. Hierbij word de volgende code gebruikt: sudo apt install software-properties-common  
   sudo add-apt-repository universe
2. Voeg vervolgens de ROS 2 GPG key toe met apt: sudo apt update && sudo apt install curl -y  
   sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
3. Voeg nu de repository aan je source lijst toe: echo "deb [arch=**$(**dpkg --print-architecture**)** signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu **$(**. /etc/os-release && echo $UBUNTU\_CODENAME**)** main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

### 3.3.3 Installeer de ROS 2 packages

1. Update je apt repository cache: sudo apt update
2. Voor het doorgaan is het handig eerst alles te updaten: sudo apt upgrade
3. Voor dit voorbeeld Word de meest kale variant van ROS geïnstalleerd genaamd ROS-Base. De code hiervoor is: sudo apt install ros-humble-ros-base
4. Als dit alles gedaan is ben je klaar om ros te gebruiken.

Het is belangrijk om voor de installatie te bedenken wat voor het desbetreffende project belangrijk is, ros base bevat alleen de basis van ROS2, ros desktop bevat daarentegen meer libraries en tools die handig kunnen zijn. De desktop versie neemt daarentegen meer ruimte en resources in.

Als het ROS weer verwijderd moet worden zie dan het kopje uninstall op deze website: <https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html>

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijvingAls de stappen juist gevolgd zijn zal ROS2 beginnen met de installatie, dit kan 10-15 minuten duren, er zal zoals in het onderstaande voorbeeld een voortgangsmeter zijn.

Figuur 4 ROS2 aan het installeren

## 3.4 Een eigen repository toevoegen om code te kunnen runnen en testen

Aangeraden is om een eigen git repository te maken. Dit kan gedaan worden via een platform zoals GitHub. Dit wordt aangeraden omdat dan op afstand makkelijk gewerkt kan worden. De code kan dan op een computer geschreven worden terwijl de robot eventuele een taak uitvoert. De manier waarop deze wordt gemaakt en toegevoegd aan de robot word uitgelegd in de volgende stappen.

### 3.4.1 Aanmaken van een git repository

Er zijn vele manieren om een git repository aan te maken maar voor dit voorbeeld wordt gebruik gemaakt van GitHub Desktop:

1. Open GitHub Desktop. (Als dit nog niet op de computer staat download het dan op de website van GitHub).

2. Navigeer naar file en maak daar een nieuwe repository aan.

3. Geeft het een toepasselijke naam die niet te moeilijk is en geef het een makkelijke locatie op de computer zodat deze snel gevonden kan worden.

4. Er is nu een repository gemaakt en deze kan gevonden worden op de website van GitHub.

### 3.4.2 Repository clonen op de Raspberry Pi

Voor het clonen van de Repository is het van uiterst belang dat bij de instellingen van de repository de repository op openbaar of gedeeld staat. Als dit niet het geval is kan de Raspberry Pi de Repository niet clonen. Als dit allemaal oké staat doe volg dan de volgende stappen:

1. Maak als eerst een map aan waar de repository op moet komen. Dit doet men met behulp van: mkdir (naam map).

2. Wanneer dat gedaan is navigeer je in de map met behulp van: cd (naam map). Dit doet men zodat de clone ook daadwerkelijk in de map gebeurd waar je de bestanden wilt hebben.

3. Als het goed is staat er nu: ~/naam map$. Als er behoefte is om meer mappen aan te maken doe dan de voorgaande 2 stappen overnieuw.

4. Als de map is aangemaakt kan de git repository gekopieerd worden. Voor het kopiëren van de repository is het van belang dat de repository link gevonden word. Deze word gevonden op de website van GitHub waar bij de aangemaakte repository gekeken worde en deze link gekopieerd word. Voor het daadwerkelijke klonen is de volgende code nodig: git clone (repository locatie link).

5. Als dit proces gedaan is dan is het handig om te kijken of het is gelukt door middel van “ls” te typen in de console. Als in de repository al mappen had aangemaakt zijn zouden dezen in een lijst te zien kunnen zijn door ls.

### 3.4.3 Repository pull

Als laatste is het belangrijk dat wanneer je de net geschreven code die dan in de repository zit dan ook gepulled wordt. Dit wordt simpelweg gedaan met behulp van git pull origin main (main kan ook master zijn).

# 4. Uitleg ROS Protocol

Voordat er verder op het programmeren van de robot in wordt gegaan wordt de werking achter ROS2 uitgelegd. Het ROS-protocol (ROS-communicatieprotocol) is een set van regels en conventies die zorgen voor communicatie tussen de verschillende onderdelen, ook wel nodes.

4.1 Nodes  
Iedere node voert een bepaalde functie uit, zoals bijvoorbeeld het besturen van een robotarm, verwerken van gegevens binnen een sensor, etc. Nodes communiceren met elkaar door gebruik van messages. Wat deze precies zijn wordt later behandeld.

Elke node heeft een unieke naam binnen het ROS2-systeem, zodat andere nodes verwijzingen kunnen doen naar andere specifieke nodes, waarmee ze willen communiceren. Nodes kunnen informatie verzenden en ontvangen door messages te publiceren/abonneren op specifieke ROS2-topics.

Nodes kunnen worden geschreven in verschillende soorten programmeertalen, zoals C+ of Python. Er zijn ook meer talen, maar deze zijn de meest populaire. De keuze hangt af van de ontwikkelaar. Het is erg gemakkelijk om bestaande code te integreren of nieuwe code toe te voegen.

4.2 Topics  
Een topic is een soort virtuele postbus waar nodes messages in kunnen plaatsen of uit kunnen halen. Een node die gericht is op een bepaald stuk informatie kan zich subscriben op een topic om die messages te ontvangen. Weer een andere node kan naar dat specifieke topic publiceren, waardoor de nodes die messages kunnen ontvangen.

Elke message dat gestuurd wordt naar een topic, heeft een bepaald formaat en inhoud dat wordt gedefinieerd door het gebruikte ROS2 message type. Bijvoorbeeld, als een robot zijn huidig positie wil delen, kan een node de positiegegevens publiceren naar een topic genaamd ‘’robot\_positie” met behulp van een specifieke message type, zoals ‘’geometry\_msg/Pose’’.

Nodes kunnen meerdere topics gebruike om allerlei soorten informatie te kunnen verzenden en ontvangen. Een node kan een message publiceren naar een topic, zonder dat het zorgen hoeft te maken welke andere nodes ze ontvangen en nodes kunnen zich subscriben op topics zonder dat zich zorgen maakt welke andere nodes de messages publiceren.

## 4.3 Messages

Nodes communiceren met elkaar door berichten naar topics te publiceren. Een bericht is een eenvoudige datastructuur, bestaande uit getypte velden. Standaard primitieve typen (geheel getal, drijvende komma, boolean, enz.) worden ondersteund, evenals arrays van primitieve typen. Messages kunnen willekeurig geneste structuren en arrays.

msg- bestanden zijn eenvoudige tekstbestanden voor het specificeren van de gegevensstructuur van een bericht. Deze bestanden worden opgeslagen in de subdirectory msg van een pakket.

1. **std\_msgs/String:** String van data.  
2. **sensor\_msgs/Imu:** Dit is een bericht om gegevens van een IMU (Inertial Measurement Unit) vast te houden.  
3. **geometry\_msgs/Pose:** Een weergave van pose in de vrije ruimte, samengesteld uit positie en oriëntatie.  
4. **nav\_msgs/Odometry:** Dit is een schatting van een positie en snelheid in de vrije ruimte.  
5. **sensor\_msgs/Image:** Dit bericht bevat een niet-gecomprimeerde afbeelding.

4.4 Services  
Een service is een soort vraag-antwoord systeem tussen nodes. Een node kan een service aanbieden door te definiëren welke aanvragen het zou kunnen verwerken en welke antwoorden het kan teruggeven. Andere nodes kunnen weer een aanvraag indienen bij die bepaalde service en vervolgens wachten op een reactie.

Net als bij topics hebben services een specifiek formaat en inhoud voor het aanvragen en antwoorden, wat wordt gedefinieerd door het gebruikte ROS2 service type. Het service type bepaalt welke soort gegevens moeten worden verstuurd in de aanvraag van die service en welke gegevens terug moeten worden gegeven in het antwoord.

Het gebruik van services maakt synchrone communicatie mogelijk tussen nodes. Dit houdt in dat de aanvrager wacht op een reactie van de antwoorder voordat hij verder gaat met de andere taken. Dit kan zeer nuttig zijn wanneer er een directe interactie of resultaat nodig is tussen de nodes.

Net als een topic hebben services een bijbehorend servicetype dat de pakketresourcenaam is van het .srv- bestand. Net als bij andere op het ROS-bestandssysteem gebaseerde typen, is het servicetype de pakketnaam + de naam van het .srv- bestand. my\_srvs/srv/PolledImage.srv heeft bijvoorbeeld het servicetype my\_srvs/PolledImage.

**rossrv:** geeft informatie weer over .srv-gegevensstructuren. Zie rossrv voor documentatie over het gebruik van deze tool.

**rosservice:** lijsten en query's ROS Services.

4.5 Actions  
Om de client en server te laten communiceren, moeten we een paar messages definiëren waarop ze communiceren. Dit is met een *Action specification*. Dit definieert de doel-, feedback- en resultaatberichten waarmee clients en servers communiceren:

**Doel**  
Om taken uit te voeren met behulp van acties, introduceren we het idee van een doel dat door een ActionClient naar een ActionServer kan worden gestuurd. In het geval van het verplaatsen van de basis, zou het doel een PoseStamped-bericht zijn dat informatie bevat over waar de robot naartoe moet gaan in de wereld. Voor het besturen van de kantelbare laserscanner zou het doel de scanparameters bevatten (min. hoek, max. hoek, snelheid, enz.).

**Feedback**  
Feedback biedt serverimplementeerders een manier om een ​​ActionClient te vertellen over de incrementele voortgang van een doel. Voor het verplaatsen van de basis kan dit de huidige houding van de robot langs het pad zijn. Voor het besturen van de kantelbare laserscanner kan dit de resterende tijd zijn totdat de scan is voltooid.

**Resultaat**  
Een resultaat wordt verzonden van de ActionServer naar de ActionClient na voltooiing van het doel. Dit is anders dan feedback, omdat het precies één keer wordt verzonden. Dit is uiterst handig wanneer het doel van de actie is om informatie te verstrekken. Voor verplaatsingsbasis is het resultaat niet erg belangrijk, maar het kan de uiteindelijke pose van de robot bevatten. Voor het besturen van de kantelbare laserscanner kan het resultaat een puntenwolk bevatten die is gegenereerd op basis van de gevraagde scan.

## 4.6 Parameters

Parameters in ROS zijn gekoppeld aan individuele nodes. Parameters worden gebruikt om nodes te configureren bij het opstarten (en tijdens runtime), zonder de code te wijzigen. De levensduur van een parameter is gekoppeld aan de levensduur van de node (hoewel de node een soort persistentie kan implementeren om waarden opnieuw te laden na opnieuw opstarten).

Parameters worden benoemd door node naam, node naamruimte, parameternaam en parameternaamruimte. Het opgeven van een parameternaamruimte is optioneel.

Elke parameter bestaat uit een sleutel, een waarde en een descriptor. De sleutel is een string en de waarde is een van de volgende typen: bool, int64, float64, string, byte[], bool[], int64[], float64[] of string[]. Standaard zijn alle descriptoren leeg, maar ze kunnen parameterbeschrijvingen, type-informatie en aanvullende beperkingen bevatten.

Standaard moet een node alle parameters declareren die het tijdens zijn levensduur zal accepteren. Dit is zodat het type en de naam van de parameters goed gedefinieerd zijn tijdens het opstarten van de node, wat de kans op verkeerde configuratie later verkleint.

# Voorbeelden ROS 2

Een paar simpele voorbeelden van de eerdergenoemde ROS2 protocollen zoals nodes om mee te beginnen.

## 5.1 Creatie van een node

Hieronder is beschreven is het grondwerk voor de basis van een ROS2 node beschreven in python, ROS2 werkt ook met c++. De logica van de node is nog leeg:

Import rclpy

def main():

rclpy.init()

Node = rclpy.create\_node('een\_node')

# Logica van de node gaat hier.

node.destroy\_node()

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

De manier waarop deze code werkt is als volgt:

1. De rclpy package wordt ge import, de package is nodig voor de werking van ROS2 met python. Het bevat de libraries en ROS2 bindings die nodig zijn om nodes te laten werken.
2. Binnen de main methode wordt de eerdergenoemde package gecalled, dit bereid het programma voor op het gebruik van de ROS2 infrastructuur.
3. De lijn :rclpy.create\_node('een\_node') is de creatie van de node zelf, tussen de haakjes komt de naam van de node zelf te staan.
4. De Logica van de node is het belangrijkste gedeelte, hierin komt te staan wat de node moet doen. Je kunt bijvoorbeeld van deze node een publisher maken,deze herhaalt een string tot een requirement gehaald is. Een Subscriber node kan die informatie dan ontvangen. Dit zal later aan worden getoond.
5. De destroy node code wordt uitgevoerd wanneer de logica compleet uitgevoerd is, de destroy lijn maakt de resources die voor de node zijn gebruikt weer vrij.
6. Vervolgens wordt de ROS2 infrastructuur afgesloten met rclp.shutdown.

# 5.2 simpele logica binnen een node

Zoals eerder is beschreven kan een node van alles doen, bijvoorbeeld publishen om informatie te versturen en subscriben om informatie te ontvangen.

# 5.2.1 Logica van een node

De volgende code is code die in de lege "Logica van de node node" lijn kan om de node een publisher te maken (mits je bij het importen van rclpy aangeeft dat er een string wordt gebruikt als msg).

import rclpy

from std\_msgs.msg import String

def main():

rclpy.init()

node = rclpy.create\_node('een\_node')

publisher = node.create\_publisher(String, 'een\_topic', 10)

msg = String()

msg.data = 'Hallo, ROS 2!'

while rclpy.ok():

publisher.publish(msg)

rclpy.spin\_once(node)

node.destroy\_node()

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

De manier waarop deze code werkt is als volgt (alleen het logica gedeelte):

1. Het eerste verschil is 'std\_msgs.msg import string', deze code geeft aan dat de 'message' die verstuurt zal worden in de vorm zal zijn van een string, voor andere typen informatie kan dit ook iets anders zijn zoals een char of een int.
2. De 'publisher' lijn is waar de publisher wordt gemaakt, het geeft aan dat de message in de vorm van een string is en dat de message onder de topic 'een\_topic' wordt gemaakt. De 10 staat voor de maximumlengte van het bericht.
3. De msg.data bevat het bericht zelf in de vorm van een string, de publisher.publish(msg) methode verstuurt de message naar toepasselijke subscriber nodes.
4. De 'spin once' methode is nodig om het publishen te laten gebeuren en om binnenkomende berichten of informatie te ontvangen. Het ontvangt ook callbacks.

# 5.2.2 creatie van een subscriber node

De destroy node en shutdown zijn vergelijkbaar met de vorigen, een simpele subscriber node is afgezien van een paar veranderingen in de logica bijna vergelijkbaar met een publisher.

Hieronder de basiscode voor een subscriber node.

import rclpy

from std\_msgs.msg import String

def callback(msg):

print('Received message:', msg.data)

def main():

rclpy.init()

node = rclpy.create\_node('een\_node')

subscription = node.create\_subscription(String, 'een\_topic', callback, 10)

while rclpy.ok():

rclpy.spin\_once(node)

node.destroy\_node()

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

De uitleg van deze node:

1. Zoals de vorige staat er onder de import het type van het soort bericht waarmee de node om moet gaan, in dit geval een string.
2. De 'callback' functie print simpelweg de ontvangen informatie uit, deze methode niet cruciaal voor het functioneren van een subcriber method maar het kan handig zijn.
3. De methode 'subscription' is hetgeen wat de subscription aanmaakt, het 'subscribted' aan een node met het type string, de 10 staat voor de maximum hoeveelheid aan data die kan worden gecached als er callback processing lag is.
4. De spin methode wordt gebruikt om de methode mogelijke callbacks te laten processen.

# 5.2.3 Het beheren van services met ROS2

Het aanbieden en oproepen van services in ROS 2 werkt als het doen van een verzoek en het ontvangen van een reactie. Nodes kunnen services aanbieden met specifieke mogelijkheden, en andere nodes kunnen die services oproepen om informatie op te vragen of acties uit te voeren. Het is een manier voor nodes om met elkaar te communiceren.

Onderstaande code is een voorbeelden van een node met de mogelijkheid om services te beheren:

import rclpy

from std\_srvs.srv import SetBool

def callback(request, response):

response.success = True

response.message = 'Service called successfully'

return response

def main():

rclpy.init()

node = rclpy.create\_node('my\_service\_node')

service = node.create\_service(SetBool, 'my\_service', callback)

while rclpy.ok():

rclpy.spin\_once(node)

node.destroy\_node()

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Afbeelding met tekst, schermopname, Lettertype, boek

Automatisch gegenereerde beschrijving

Figuur 5 voorbeeld van een talker die een bericht verstuurt naar een listener

# Je robot laten bewegen.

Om de motor te laten bewegen moet er interactie komen tussen de software en de motors van de robot. Hiervoor zal gebruik worden gemaakt van Python code. Na het opstellen van de code kan die informatie toe worden gepast op ROS2 om het interactief te maken.

De voorbeeld code die voor het bewegen van de motoren zal worden gebruikt is afkomstig van een Github repository van Sid Faber voor de Camjam Edukit 3. (link naar de Github: <https://github.com/SidFaber/robots/tree/master/poolcity>)

## Instellen van de software.

Aannemend dat de robot gebouwd is en dat de bekabeling juist is kun je meteen aan de slag met het instellen van je workspace.

1. Maak een directory aan met een toepasselijke naam. dit doe je in de Ubuntu omgeving van je raspberry pi. Het commando is als volgt: mkdir roboticafolder

Vervolgens om in het nieuwe bestand te komen : cd robitcafolder/

De naam van het bestand hoeft niet roboticafolder te zijn, het is de keuze van de gebruiker.

2. Vervolgens zal een source directory aan worden gemaakt voor het klonen van de Github directory. Mkdir src à cd src/ à git clone à <https://github.com/SidFaber/robots.git>

3. Het controleren van de bestanden kan met de list commando: ls.

Als het klonen juist is verlopen zal in je main workspace met het commando git status te zien zijn, als het gelukt is, is het volgende te zien.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Figuur 6 voorbeeld van een gelukte git clone

## 5.2 Simpele beweging voor een korte tijd

Om de motoren van de robot voor een korte tijd te activeren moeten de pinnen op de raspberry pi eerst ingesteld worden. Deze code is Python 3 maar wordt later gelinkt aan ROS2. De volgende code is specifiek voor de Camjam Edukit 3 dus als de robot een andere bekabelingsvolgorde heeft dan zal het anders zijn. De onderstaande code is een korte test die de robot voor 1 seconde naar voren laat rijden. Als deze code gebruikt wil worden is het niet nodig om het over te nemen (mits je gebruik maakt van Sid Fabers repository), als de git clone goed is gegaan staat kan het gevonden worden in de directory: ~/robot\_ws/src/robots/poolcity/tests$.

De desbetreffende code wordt onderstaand toegelicht.

﻿#!/usr/bin/python3

import RP1.GPIO as GPIO

import time

**De libraries die nodig zijn om de pinnen aan te sturen worden hier geïmporteerd, maar voordat dit kan werken moet de source hiervoor in worden gesteld met: sudo apt-get -y install python3-rpi.gpio.**

# GPIO initialization

GPIO.setmode (GPIO.BCM)

GPIO.setwarnings (False)

**Het uitschakelen van waarschuwingen en het selecteren van de juiste instelling voor het selecteren van de pinnetjes.**

# set GPIO pin mode

GPIO.setup (7, GPIO.OUT) GPIO.setup (8, GPIO.OUT) GPIO.setup (9, GPIO.OUT) GPIO.setup (10, GPIO.OUT)

#turn motors off

GPIO.output (7,0)

GPIO.output (8,0)

GPIO.output (9,0)

GPIO.output (10,0)

**Pin 7, 8 , 9 en 10 zijn de pinnen die worden gebruikt. Hier worden ze in de “output” modus gezet zodat ze gebruikt kunnen worden om de robot aan te sturen. Deze pinnen zijn verbonden aan de motoren, deze indeling van pinnen is specifiek voor de Camjam kit, een eigen robot kan andere pinnen gebruiken.**

﻿

# turn right motor forward

GPIO.output (7,0)

GPIO.output (8,1)

**Pin 7 wordt hier in een inactieve staat gezet door de 0, De 1 bij pin 8 staat voor actief, dus de rechter motor draait.**

#turn left motor forward

GPIO.output (9,0)

GPIO.output (10,1)

**Pin 9 wordt hier op inactief gezet en pin 10 op actief, dus de linker motor draait naar voren. Dit wordt geregeld door de motor controller die op de robot geplaatst is.**

time.sleep(1)

**Time sleep zorgt ervoor dat de code voor een bepaalde tijd een delay heeft voordat het verder gaat naar de volgende code, de 1 in dit geval is de duratie van de delay.**

# Reset GPIO and turn off motors

GPIO.cleanup()

**Reset, alle motoren worden op 0 gezet.**

om dit uit te voeren op een Camjam Edukit 3, moet je binnen je bestand naar de source gaan.

vanaf daar typ je cd poolcity/ in en de list commando ls, hier ze je alle testcode bestanden voor de robot. Om bijvoorbeeld uit te voeren wat net benoemd is moet de volgende commando uit worden gevoerd: pyton3 gpiotest.py, als het goed is rijdt de robot nu voor een seconde. Onderstaand is hoe het er uit moet komen te zien.

Afbeelding met tekst, schermopname, Lettertype

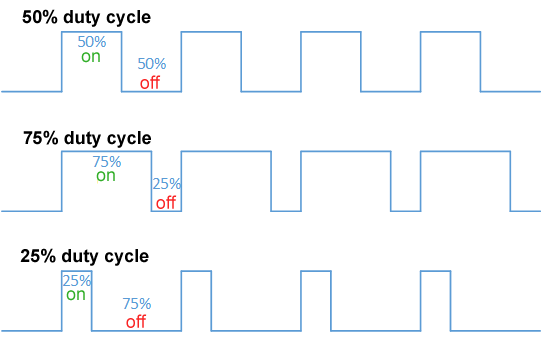
Automatisch gegenereerde beschrijving

Figuur 7 gpiotest uitvoering

Deze voorbeeldcode laat zien hoe de motor van een robot aan en uit kan worden gezet door de informatie door de pinnen te versturen, echter de motoren hebben zoals het staat alleen de optie om aan en uit te zijn. Dus vol gas of niets, hoe de snelheid aan kan worden gepast wordt in het volgende hoofdstuk uitgelegd.

## Snelheid van de motoren aanpassen

Het is niet de bedoeling dat de motoren op de robot altijd op de maximale snelheid draaien, daarom is het van belang om de snelheid naar eigen voorkeur te kunnen veranderen. Dit wordt gedaan met “pulse width modulation”, dit is een techniek om de gemiddelde kracht en intensiteit van een elektrisch signaal te beïnvloeden door het heel snel in en uit te schakelen. Om het op een simpele manier te beschrijven, het is een digitale manier om een analoog signaal na te boosten. PWM is de afkorting voor deze techniek, het kan op veel verschillende manieren worden gebruikt, waaronder het beheren van de snelheid van een elektrische motor en audiovolume. Het razendsnelle in en uitschakelen van de motor staat lagere snelheden toe, dit concept wordt in de volgende afbeelding weergegeven. De “Duty cycle” is de operatiecyclus van de motor, zoals in de afbeelding te zien is, is er meer activiteit hoe hoger de Duty Cycle is.



Figuur 8 visualisatie van PWDM

## Toepassing op Python code

Aannemend dat RPi.GPIO al geimporteerd is kun je de duty cycle van de pins als volgt in Python aanpassen.

GPIO.setmode(GPIO.BOARD)

GPIO.setup(pin\_number, GPIO.OUT)

De GPIO.setmode past de gpio instelling aan zodat het nummerschema van de pins op de PI worden gebruikt. De GPIO.setup(pin\_number, GPIO.OUT)stelt de pin in als een output pin.

pwm = GPIO.PWM(pin\_number, frequency)

Hier word de PWM aangemaakt, het is gerelateerd aan de eerdergenoemde gpio pin en zal het signaal generen. Als je “pin\_number” vervangt met de pin die je kiest, en de “frequency” met een frequentie in Hertz is de basis klaar.

Als volgende moet de PWM output gestart worden, pwm.start(duty\_cycle), de tekst “duty\_cycle” moet vervangen worden met een waarde van 0 / 100 procent, dat is de duty cycle.

De gekozen Duty cycle moet ook veranderd kunnen worden, bijvoorbeeld als je wil dat een robot sneller gaat rijden of dat een robot arm sneller beweegt. Dit kan behaald worden met de volgende code: pwm.ChangeDutyCycle(new\_duty\_cycle)de oude duty cycle wordt vervangen met “new\_duty\_cycle”, een waarde van 0 tot 100. Dit staat toe dat de duty cycle op een dynamische manier gewijzigd kam worden.

Als er een pauze nodig is tussen de executie van code zonder dat de Duty cycle onderbroken wordt dan is de benodigde code: time.sleep(duration), de “duration” moet vervangen worden met de pauze in seconden. Dit kun je bijvoorbeeld gebruiken om de robot voor een bepaalde tijd te laten rijden, dit is in hoofdstuk 5.3 ook het geval geweest, de time sleep is de reden dat de robot voor één seconde bewoog.

Als het doel is om de PWM compleet stil te leggen is de volgende code van toepassing:

pwm.stop()

GPIO.cleanup()

## voorbeeld gebruik PWM

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Figuur 9 voorbeeld gebruik pwm

Zoals in het voorbeeld te zien is wordt PWM hier gebruikt om de robot voor 2 seconden naar voren te laten rijden met een duty cycle van 50 procent. Eerst worden de eenheden gedeclareerd, zoals “right motor pin =7”, dan wordt de duty cycle van 0 procent naar 50 procent veranderd. Als laatste wordt de robot stilgelegd met een duty cycle van 0 procent en worden de gebruikte resources door gipo cleanup vrijgemaakt.

## Implementatie in ROS2

Om de robot meer functioneel te laten zijn, zal er ROS2 worden gebruikt voor de beweging van de robot. Hiervoor zal de code voor het bewegen aangepast worden en in classes worden verdeeld, er zal gebruik worden gemaakt van onder andere nodes topics services en subscribers.

Er zal nu een stappenplan worden gegeven over hoe de basis hiervan zou kunnen worden gedaan:  
**stap 1:** importeer de benodigde modules, nu zullen er naast de GPIO module ook de ROS2 node en messenger modules geimporteerd moeten worden.

import RPi.GPIO as GPIO

import rclpy

from rclpy.node import Node

from std\_msgs.msg import String

Afbeelding met tekst, schermopname, ontwerp

Automatisch gegenereerde beschrijving

Figuur 10 voorbeeld beweging node met subscriber

**Stap 2:** Er moet een ROS2 node worden gedefinieerd voor control over de motors, in dit geval zal de node de naam “Motorcontrolnode” krijgen. Deze node bevat de eerder besproken code, waaronder de code voor PWM. De callback methode ontvangt berichten van de ‘cmd\_vel’ topic en de voert de bijbehorende motor controles uit gebaseerd op de ontvangen string commandos. De functies die hiervoor worden gebruikt zijn onder andere de ‘move\_forward’ en ‘move backward’ functies, dit werkt met behulp van PWM. Het formaat van ROS2 functies is in een eerder hoofdstuk uitgelegd.

**Stap 3:** Maak een instantie van de gemaakte node in ROS2.

ROS2 wordt hierbij geïnitialiseerd door rclpy.init(), er wordt een instantie aangemaakt van de motorcontrol class. Om de node steeds actief te houden wordt er gebruik gemaakt van de commando spin.

def main(args=None):

rclpy.init(args=args)

motor\_control\_node = MotorControlNode()

rclpy.spin(motor\_control\_node)

motor\_control\_node.destroy\_node()

rclpy.shutdown()

GPIO.cleanup()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Stap 4:** stuur de robot commando’s door te publishen op de ‘cmd\_vel’ topic, dit moet in een specifiek formaat, namelijk:

ros2 topic pub /cmd\_vel std\_msgs/String "{data: 'forward'}"

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

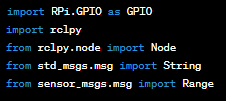
Figuur 11 Werkende publisher, hoe het eruit zou moeten zien

**Samenvatting:** De modules moeten geimporteerd worden à er moet een node aangemaakt worden met motor controls à binnen die node moet een subscriber komen met een topic àbinnen de node moet een callback methode komen voor de uitvoering van de commando’s

# Implementatie sensor

Een Sensor aan de robot koppelen kan veel toevoegen, sensors kunnen bijvoorbeeld gebruikt worden voor obstakel detectie en veiligheidsfuncties. Om een sensor toe te voegen en in gebruik te nemen met Python en ROS2 moeten de volgende stappen worden ondernomen. In deze toepassing gaan we er van uit dat de sensor zal worden gebruikt om de robot veilig te laten stoppen als er een obstakel voor de robot is.

**Stap 1:** importeer de Range message module, deze is nodig voor de afstand sensors. De code hiervoor is: from sensor\_msgs.msg import Range. Hier een voorbeeld van hoe dit er uit zou zien in de eerder gebruikte code:



Figuur 12 voorbeeld correcte import Range

**Stap 2:** Definieer de pin(s) die door de GPIO gebruikt zullen worden voor de sensor, dit komt voor de GPIO ingesteld is. De code hiervoor is: self.distance\_sensor\_pin = 13. De 13 hier is het getal dat vervangen moet worden met de eigen pin nummer.

**Stap 3:** Voeg de variabele toe die bepaalt wat de veilige afstand is, de code hiervoor is self.safe\_distance = 0.3, deze code komt net als de pin voor de sensor in de motor control node te staan (in dit geval). De 0.3 verschilt per sensor, het hangt af van welke sensor je hebt en hoe gevoelig de gebruikte sensor is. De waarden die de sensor doorgeeft kan zoals eerder uitgelegd worden getest.

**Stap 4:** De distance sensor pin moet nu worden geconfigureerd als een input, dit wordt gedaan met GPIO.setup(). Dit kan met de volgende code: GPIO.setup(self.distance\_sensor\_pin, GPIO.IN). De pins van de motoren staan in de GPIO geconfigureerd als een output, omdat ze data ontvangen, de sensor daarentegen voert informatie terug naar de robot en moet dus als een input in worden gesteld. De afbeelding toont aan hoe de GPIO voor de sensor pin in kan worden gesteld als een input.

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

Figuur 13 GPIO setup

**Stap 5:** Er moet een subscriber gemaakt worden voor de afstand sensor, deze subscriber moet messages van het type “Range” kunnen ontvangen. De subscriber zal binnen de initilizatie methode in de motor control klasse komen te staan. Hieronder de creatie van een subscriber als voorbeeld.

self.range\_subscriber = self.create\_subscription(

Range,

'distance\_sensor',

self.distance\_callback,

10

)

Deze code maakt een subscriber voor de topic ‘distance\_sensor’, de callback methode zal aan worden geroepen wanneer een message van het type “Range” op de topic is ontvangen.

Het getal 10 onderaan staat voor de wachtrij grootte voor deze subscriber, dus er kunnen maximaal 10 berichten op worden geslagen en in de wachtrij gezet worden.

**Stap 6:** Implementeer de distance\_callback methode, wanneer er een bericht wordt ontvangen kan deze methode uit het bericht de waarde van de afstand ontleden. Als de berekening zegt dat de afstand te klein is worden de motoren stilgezet met de stop.motors method. Dus de duty cycle voor de twee motors verandert naar 0 als de afstand te klein is.

def distance\_callback(self, msg):

distance = msg.range

if distance < self.safe\_distance:

self.stop\_motors()

**Stap 7:** definieer de veilige afstand methode.

def is\_safe\_distance(self):

distance = GPIO.input(self.distance\_sensor\_pin)

return distance >= self.safe\_distance

Deze methode leest de afstand met de GPIO-input en vergelijkt de ontvangen afstand met de veilige afstand. Als de afstand groter is dan de veilige afstand returnt de methode ´true´ als indicatie dat het veilig is om weer normale snelheid te rijden. Als het False returnt betekent het dat de robot de dicht bij een obstakel is. Deze informatie word vervolgens in de volgende stap op de motors toegepast zodat de robot

**Stap 8:** Pas de veilige afstand toe op de motoren, doe dit in de move\_forward methode. Deze stap en stap 7 zijn niet cruciaal maar er kan voor gekozen worden om ze toch uit te voeren.

def move\_forward(self):

if self.is\_safe\_distance():

self.right\_motor\_pwm.ChangeDutyCycle(50)

self.left\_motor\_pwm.ChangeDutyCycle(50)

else:

self.right\_motor\_pwm.ChangeDutyCycle(30)

self.left\_motor\_pwm.ChangeDutyCycle(30)

Door deze code verlaagt de Duty cycle als je te dicht bent bij een obstakel, wanneer de robot de self.safe distance verbreekt komen de motoren compleet tot een stop. Verwar de methoden ‘is\_safe\_ distance’ niet met de methode ‘safe\_distance.’ De waarden die gekozen zijn voor deze twee zullen aangepast worden naar de specificaties van de eigen sensor.

**Stap 9:** Initialisatie van de gemaakte code.

def main(args=None):

rclpy.init(args=args)

motor\_control\_node = MotorControlNode()

rclpy.spin(motor\_control\_node)

motor\_control\_node.destroy\_node()

rclpy.shutdown()

GPIO.cleanup()

if \_\_name\_\_ == '\_\_main\_\_':

main()

De code voor het eigen project zal hoogstwaarschijnlijk geheel anders zijn, maar deze tekst biedt een voorbeeld van hoe het eruit zou kunnen komen te zien.

Het wordt zeer aangeraden om de videos van Sid Faber te kijken als je een ROS2 robotje gaat bouwen, hij heeft ook een playlist specifiek voor de Camjam Edukit 3.

Deze zijn te vinden in de playlist:

<https://www.youtube.com/watch?v=xHw2WFe6GBs&t=31s>

# Bronnenlijst

Figuur 8: <https://en.wikipedia.org/wiki/Pulse-width_modulation>

Bronnen kennis ROS2:

<https://www.youtube.com/watch?v=xHw2WFe6GBs&t=31s>

<https://www.youtube.com/watch?v=96XsJ7xfsS8&t=6s>

<https://www.youtube.com/watch?v=7TVWlADXwRw>

<https://www.youtube.com/watch?v=3HuV1M1NMB8>

<https://www.youtube.com/watch?v=dxcU-_PGZdw&t=97s>

Toepassingen ROS2: AutomaticAddison.

Boek: Via de bibliotheek / ebook ROS: introduction ROS2.

ROS2 for beginners, video series via de bibliotheek.

Ebook: ‘A Concise Introduction to Robot Programming with ROS2’.

Bronnen ROS2:

<https://docs.ros.org/en/dashing/Installation/Ubuntu-Install-Debians.html>

<https://docs.ros.org/en/humble/Installation.html>