Autonomous emergency robot

Afstudeerverslag

HBO ICT Software Engineering

Pieter Zeilstra

436342@student.saxion.nl

06 398 573 64

Versie 2.0
2019/2020

SAXION HOGESCHOOL

ICT

VERSIE	DATUM	OPMERKING
0.1	09/03/2020	Opzet verslag
0.2	10/04/2020	Uitbreiding kopjes
0.3	20/04/2020	Toevoeging andere documenten
0.4	05/05/2020	Uitbreiding kopjes
0.5	11/05/2020	Feedback verbeteren
0.6	12/05/2020	Onderzoek samengevat
0.7	13/05/2020	Verbetering setup feedback Wouter
0.8	18/05/2020	Verbetering feedback Shashidhar
0.9	22/05/2020	Testen en reflectie.
1.0	25/05/2020	Verbetering feedback Alfons
1.1	29/05/2020	Verbetering feedback van Wouter.
1.2	02/06/2020	Puntjes op de i. Concept verslag inleveren
1.3	08/06/2020	Verwerking feedback concept verslag Peter.
1.4	09/06/2020	Verwerking feedback Wouter en Peter + toevoeging planning
1.5	10/06/2020	Verwerking feedback + toevoeging autonoom rijden
1.6	11/06/2020	Spelling verbetering
1.7	12/06/2020	Grote check en verbetering.
2.0	14/06/2020	Toevoeging implementatie userstories

Inhoudsopgave

Lijst met afkortingen

Afkorting	Betekenis
BLE	Bluetooth low energy
Gazebo	Een simulatie programma om robots te simuleren in virtuele omgevingen
LIDAR	Light Detection and Ranging
POC	Proof of concept
ROS	Robot Operating System
RVIZ	Een data visualiseer programma van ROS
RSSI	Received Signal Strength Indicator
SLAM	Simultaneous localization and mapping

Tabel 1 Lijst met afkortingen

Bijlagen

Naam document	Inhoud
Development omgeving	Uitleg wat voor software er is gebruikt.
Functioneel ontwerp	Uitleg ontwerp en functies.
Handleiding	Handleiding over gebruik van het product.
Onderzoek App	Dit document beantwoordt alle hoofd- en deelvragen van de app.
Onderzoek Robot	Dit document beantwoordt alle hoofd- en deelvragen van de robot.
Plan van aanpak	Plan van aanpak.
Technisch document	In dit document staan alle keuze verantwoordingen en technische aspecten uitgelegd.
Userstories	Hierin staan alle userstories.

Tabel 2 Lijst met bijlagen

Lijst van figuren

Communicatie

Afstudeerder

Pieter Zeilstra	436342@student.saxion.nl	+31 6 3985 7364

Stagebegeleider

Wouter van Kleunen	w.a.p.vankleunen@saxion.nl	+31 880195897

Bedrijfsbegeleider

Alfons Muil	Alfons.muil@ict.nl	+31 6 2573 3531

Product owner

Bart Lamot	bart.lamot@ict.nl	+31 6 2708 7469

Gegevens ICT Group

Adres	Munsterstraat 7
Postcode	7418 EV
Plaats	Deventer
Telefoonnummer	+31 (0)88 908 2000
Email	info@ict.nl
Website	www.ict.eu

1. Samenvatting

In dit afstudeerverslag wordt het proces en resultaat besproken van de opdracht “Autonomous emergency robot”. De opdracht is gedaan bij het bedrijf ICT Group in Deventer. ICT Group focust zich voornamelijk op embedded systemen en wil meer te weten komen over het gebruik van autonome voertuigen in distributiecentra. De opdrachtgever wil een eerste hulp robot die een hulp signaal kan ontvangen van een mobiele telefoon. Nadat een hulpsignaal is verstuurd moet de robot autonoom navigeren naar de dichtstbijzijnde bluetooth beacon bij een ongeluk. De opdracht bestaat uit twee delen:

- Het maken van een kaart en opslaan van bluetooth beacons.
- Rijden naar een bluetooth beacon na een hulpsignaal van een mobiele telefoon.

Als eerste is er een plan van aanpak en requirements opgesteld met de opdrachtgever. Ook is hier besloten om van de agile methode ‘Kanban’[23] gebruik te maken. Hierna is er onderzoek gedaan met behulp van ‘Community research’[24] en ‘Literature study’[25]. Hieruit kwam dat er een mobiele app, robot software en een externe api gemaakt moesten worden. Ook kwamen er hier weer nieuwe randvoorwaarden en requirements uit. Alle functionele requirements zijn omgezet in userstories. In de realisatie fase is alles gerealiseerd. In de planning hebben we alles in drie verschillende realisatie fases gepland: Robot 1(Alles met betrekking tot de robot), App(Alle musts van de app) en Robot 2 (Het autonoom maken van een kaart en uitloop). In het kopje realisatie is de implementatie uitgelegd en zijn keuzes onderbouwd. De robot die gebruikt is een Turtlebot3[1] en werkt met ROS Kinetic[5]. De Turtlebot3 heeft een LIDAR sensor[7] die een kaart van de omgeving kan maken. De Turtlebot kan autonoom rondrijden doormiddel van het Frontier Exploration algoritme. Ondertussen wordt er een kaart gemaakt met het SLAM algoritme[8] Gmapping[6] en worden er bluetooth beacons opgeslagen door naar de positie van de robot te kijken en de RSSI van de bluetooth beacons. De app kan ook de dichtstbijzijnde bluetooth beacon vinden met de RSSI. Wanneer er op de hulp knop van de app wordt gedrukt, wordt er een hulp signaal met UUID van de dichtstbijzijnde bluetooth beacon naar een externe api gestuurd die het doorstuurt naar de robot. De robot kijkt in de eerder gemaakte kaart met de posities van de bluetooth beacons en navigeert naar de meegestuurde bluetooth beacon. Uiteindelijk zijn alle gemaakte userstories getest met behulp van de geschreven ‘how to test’ en is het hele project een succesvol in het kantoor getest. Als laatste zijn er een paar aanbevelingen gedaan zodat het project beter gaat werken en aan minder randvoorwaarden hoeft te houden. Om dit te doen moet er een bijvoorbeeld gebruik worden gemaakt van een andere robot met meerdere sensors.

2. Inleiding

ICT Group heeft in haar 40-jarig bestaan een groot en divers portfolio van klanten opgebouwd. De klanten bevinden zich in uiteenlopende markten zoals bijvoorbeeld energie, water & infra, healthcare en logistiek. De overkoepelende factor is technologie die versterkt wordt door software.

ICT Group ziet steeds meer een intrede van connected technologie in de distributiecentra van klanten en wil meer ervaring opdoen met het "zo eenvoudig mogelijk maken" van deze technologie. Daarom is er een afstudeeropdracht van gemaakt om een verse blik hierop te krijgen.

ICT Group wil een ‘’eerste hulp’’ robot rond laten rijden in een warehouse. Deze robot moet autonoom naar ongelukken kunnen navigeren door het gebruik van bluetooth beacons en een mobiele telefoon app. De robot die gebruikt wordt is een Turtlebot3.

In dit document is de uitvoering van de opdracht te vinden. Hoe het project is aangepakt, de planning, welke onderzoeken er zijn uitgevoerd, hoe het project is gerealiseerd en een reflectie.

3. Het bedrijf

ICT Group is een groot bedrijf met verschillende vestigingen in het land, met in totaal meer dan duizend medewerkers. De voornaamste focus van ICT Group ligt in embedded systemen. Omdat ICT Group een groot, dit ook terug te zien in de opdrachtgevers van ICT Group. ICT Group maakt bijvoorbeeld auto dashboards voor grote autofabrikanten zoals Volkswagen en DAF, elektronische ingang poorten voor Rotterdam haven, elektronische voedersystemen voor boerderijen maar ook een bagage check-in systeem voor een vliegveld in Engeland.

De afstudeeropdracht is uitgevoerd in het kantoor in Deventer. Elk pand van ICT Group heeft vaak verschillende afdelingen die zich op andere sectoren focussen. In Deventer zijn drie afdelingen gesitueerd, alle met een focus op embedded technologie.

- *Machine & Systems;* de afdeling met klanten die apparatuur maken voor de industrie. Voor deze klanten ontwikkelen zij firmware, embedded software en Cloud toepassingen.
- *Automotive;* deze afdeling heeft klanten in de automobielindustrie, waarvoor zij software maken voor klanten zoals Volkswagen en DAF.
- *Industries;* deze afdeling richt zich op veevoer fabrieken. Hier ontwikkelen ze machines voor bijvoorbeeld het produceren van veevoer.
- *Interne IT-afdeling;* deze afdeling behandelt alle zaken rond laptops, telefoons en software voor alle medewerkers van ICT Group.

Voor meer informatie over ICT Group kan er op de website gekeken worden: https://ict.eu/nl/

4. Opdracht

4.1. Probleemstelling

Distributiecentra worden steeds groter en vaker voorzien van autonome voertuigen. Door alle extra bewegingen in het magazijn kan het gebeuren dat er een ongeval gebeurt. Er moet een autonome robot met een EHBO-voorziening ontwikkelt worden die met indoor positionering zelf door het pand kan navigeren. De indoor positionering zal plaatsvinden met behulp van bluetooth beacons.
Bij een calamiteit kan een medewerker via een app het voertuig oproepen. De robot navigeert vervolgens zelf naar het dichtstbijzijnde beacon van het ongeval. De eerste keer kan het voertuig zelf rondrijden om een kaart te maken en daarin beacons te ontdekken zodat de robot daardoor later in die kaart kan navigeren. De focus bij de opdracht ligt op de software, de hardware is een gegeven (autonoom voertuig en bluetooth beacons).

Figuur 1 Voorbeeld van de opdracht

4.2. Gewenste eindsituatie

- Robot software die autonoom een kaart van de omgeving kan maken, bluetooth beacons kan ontdekken en in een kaart kan zetten.
- Een app waarmee een signaal met de dichtstbijzijnde bluetooth beacon naar de robot gestuurd kan worden.
- Robot software die een signaal binnen kan krijgen en hierna autonoom naar de juiste beacon kan rijden door middel van de eerder gemaakte kaart.

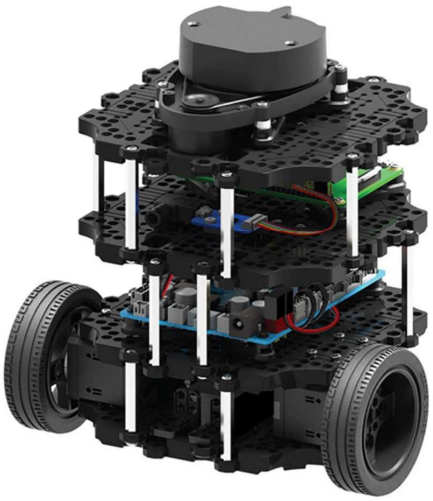
4.3. Gegeven hardware en software

4.3.1. Hoe werkt de Turtlebot3?

Turtlebot3 [1] is de meest bekende en populaire open source robot van nu. De Turtlebot heeft een snel draaiende 360 graden lasersensor waarmee ruimtes makkelijk in kaart gebracht kunnen worden. Er kan python of c++ code geschreven worden om de robot acties uit te laten voeren. Ook kunnen er allerlei fysieke modules aan de robot toegevoegd worden, zoals een extra sensor of camera. De Turtlebot3 werkt met het framework ‘ROS’ [4] wat veel functionaliteit en programma’s biedt. ROS is een soort tussenlaag die kan praten tussen de hardware en de software. Het voordeel

van de Turtlebot is dat er standaard al handige programma's voor bijvoorbeeld het maken van een kaart al op de robot werken. De data van de robot, motoren en sensors worden al goed doorgezonden naar ROS. De doorgezonde data heb je voor veel algoritmes nodig. Zo kan je direct zelf aan de slag en werken andere packages van mensen ook snel.

De Turtlebot3 die voor dit project is gebruikt is een Burger. De burger heeft dezelfde functionaliteit als de andere Turtlebot alleen mist die een camera en is hij wat kleiner. Achteraf is er alsnog een camera op de burger gezet om het testen van de robot makkelijker te maken.



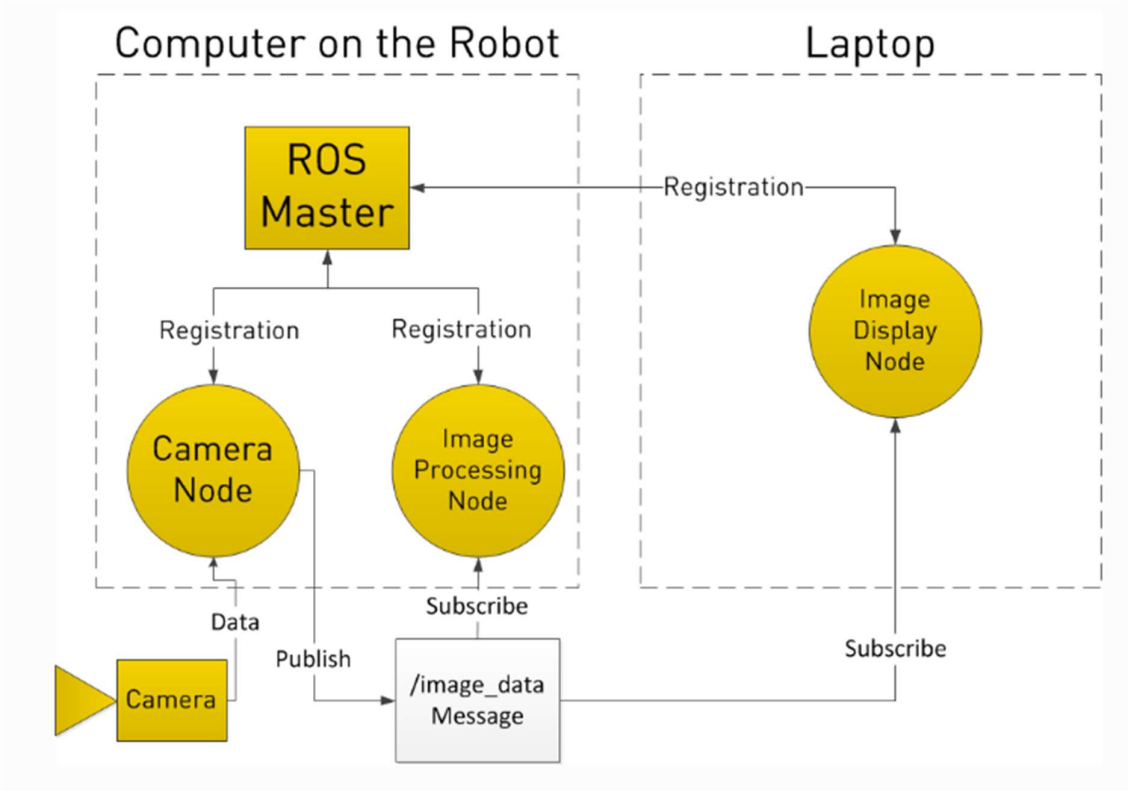
Figuur 2 Turtlebot3 Burger

De Turtlebot heeft een LIDAR laser sensor [7] en beschikt over een Raspberry pi 3b+ [3] en een openCR Arduino. De Raspberry pi is de computer die ook verbinding maakt met de server. De Raspberry pi staat in verbinding met de openCR Arduino die de wielen aansturen. Hierdoor kan er op de Raspberry pi een programma draaien, die de wielen via de Arduino besturen of data van de wielen uitlezen. Ook staat de Raspberry pi in verbinding met de LIDAR laser sensor. De Raspberry pi leest de data uit en stuurt dit door doormiddel van een programma die al op de robot staat. Op de Raspberry pi staan alle programma's die voor de robot geschreven worden. De openCR beschikt verder nog over een paar knoppen, een buzzer en een paar LED lampjes die eventueel ook door de Raspberry pi aangestuurd kunnen worden.

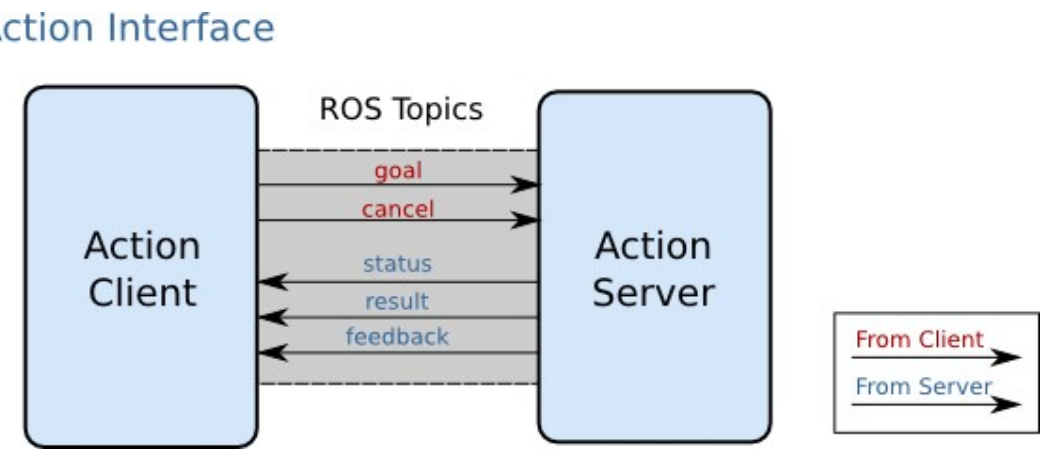
Om ROS op de Raspberry pi van de Turtlebot3 werkend te krijgen moet er een Rasbian stretch op de website van Turtlebot gedownload worden en op de Raspberry pi gezet worden (http://emanual.robotis.com/docs/en/platform/turtlebot3/raspberry_pi_3_setup/#install-linux-based-on-raspbian).
In deze stretch staat ROS klaar en zo staan er Nodes en launch bestanden voor het bewegen, uitlezen en gebruik van de robot.

4.3.2. Hoe communiceert ROS met de Turtlebot3

ROS[5] installeer je op Linux Ubuntu 16.04 met de bijbehorende Turtlebot3 dependencies. Hierna kan je de Turtlebot aansturen, uitlezen en simuleren. ROS werkt met Nodes. Nodes zijn niets meer dan code dat draait, data uitleest van Hardware of ontvangt van andere Nodes, hier iets mee doet en data als een 'Topic' kan publiceren. Voor meer uitleg kijk hiervoor naar bron [4]. Denk bijvoorbeeld aan een node die de positie van de Turtlebot teruggeeft of een Node die de beweging regelt. Er zijn Nodes die hardware uitlezen maar ook Nodes die andere Nodes uitlezen en hiermee wat doen. In de terminal of in de code kan je op een 'Topic' abonneren om uit te lezen wat de Node teruggeeft. Zelf kan je ook een Node maken en naar een Topic publiceren of abonneren. Er staat standaard een Node op de Turtlebot die zorgt dat de robot kan bewegen. Deze node luistert naar een bepaald Topic, als de correcte data naar de Topic wordt gepubliceerd/gestuurd, zal de Node die ontvangen en gaan bewegen. Er is ook een Node die de positie van de Turtlebot teruggeeft. Deze Node kan geen data ontvangen maar stuurt alleen data. Het handige van de Turtlebot is dat het al veel Nodes bezit voor de sensoren en motoren. Dit scheelt veel tijd.



Figuur 3 Werking Nodes, Publishers en Subscribers. Bron [4]



Figuur 4 Voorbeeld Action client en server Bron [33]

Ook heb je een Action Client en een Action Server. Als er data van Nodes heen en weer gestuurd moet worden kan je in plaats van Publisher en Subscribers, van een Actionserver en Action client gebruik maken. Deze staan in koppeling met elkaar en zijn asynchroon. Dit is handig als je bijvoorbeeld een actie van een andere Node wilt uitvoeren en daarna wil weten of de actie is gelukt.

Het soort message moet wel van tevoren bekend zijn voordat het gestuurd wordt. Je moet bijvoorbeeld meegeven dat je een string gaat sturen of een videostream. Je kan dus niet zomaar wat meesturen en aan de andere kant ontvangen. Er kunnen ook custom messages worden gemaakt. Dit wordt bijvoorbeeld gedaan voor het doorsturen van een lijst met bluetooth beacons. Beide machines (Robot en Server) moeten wel de custom messages een keer gecompileerd hebben anders kan de data alsnog niet uitgelezen worden omdat de data onbekend is.

Nodes staan los van elkaar en moeten los van elkaar opgestart worden. Ook moet de simulatie en andere programma's allemaal via de command line gestart worden. Bij het uiteindelijke project willen we alles zoveel mogelijk automatiseren. Gelukkig zijn hier launch files voor. Launch files zijn Xml-bestanden die door ROS uitgevoerd kunnen worden. Je kan hier aangeven wat voor Node en programma's er opgestart moeten worden en met wat voor parameters, dit automatiseert veel en scheelt uiteindelijk veel tijd.

5. Aanpak

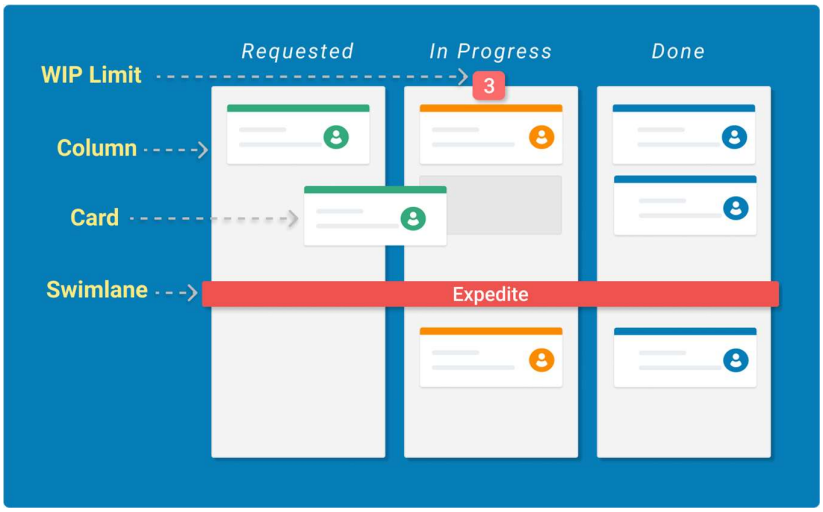
Hierna is er begonnen met een plan van aanpak opstellen. Er is begonnen met een agile methodiek te kiezen dat paste bij dit project. Daarna zijn kwaliteit eisen en een planning vastgesteld.

Omdat er nog niet veel kennis over ROS en de Turtlebot was, is ervoor gekozen om het opstellen van de eisen en het uitvoeren van de onderzoeken tegelijkertijd te doen. Hierdoor konden de requirements en voorwaarden beter opgesteld worden omdat er vaak nieuwe kennis aan het licht kwam dat een requirement veranderde. Zo is er bij het onderzoek de vraag: "Wat is de beste manier om de robot te laten navigeren zonder wifi?" en bij de requirements de eis: "De robot na een signaal van de server te hebben ontvangen, navigeren en een kaart kunnen maken zonder wifi." bijgekomen omdat er tijdens het onderzoeken de ontdekking werd gedaan dat je wel een wifi verbinding nodig hebt om correct te kunnen navigeren.

De volledige aanpak staat in bijlage "Plan van aanpak"

5.1. Project methodiek – Kanban

Er is gekozen om de Kanban methodiek[23] te gebruiken. Een grote reden hiervoor is omdat dit project maar wordt uitgevoerd door één persoon. Hierdoor vallen veel andere technieken zoals scrum af. Scrum werkt met rollen, standups en sprints, dit kan niet met een team van maar één persoon worden gedaan. Kanban lijkt in veel opzichten op scrum maar is meer open en flexibeler. Kanban werkt met een Kanban bord wat vergelijkbaar is met een scrumbord. Een Kanban bord maakt gebruik van swimming lanes met userstories. Deze swimming lanes mag je zelf inrichten aan de hand van je werkproces, bijvoorbeeld backlog, doing en done. Sommige swimming lanes hebben een maximaal aantal userstories die er tegelijk in mogen staan. Dit is gedaan om ervoor te zorgen dat er niet te veel taken tegelijkertijd opgepakt kunnen worden.
Ook kunnen er horizontale swimming lanes zijn, dit is vaak op basis van prioriteit.
De belangrijkste userstories komen in de bovenste horizontale swimming lanes geplaats zodat ze sneller opgepakt worden(Dit word vaak de expedite genoemd). Kanban werkt goed als je individueel werkt. Userstories worden aangemaakt aan de hand van de requirements. Er word gekeken welke taken er gedaan moeten worden om een requirement af te ronden. Voor deze taken worden userstories aangemaakt.
Kanban heeft geen vaste rollen en werkt niet met sprints maar is één doorlopend proces.
Met Kanban heb je zelf een idee wat er nog gedaan moet worden en kan je niet te veel tegelijkertijd doen. Je bent vrij met Kanban omdat je niet vastzit aan sprints.



Figuur 5 Voorbeeld van een Kanban bord

5.2. Development omgeving

Alle software die gebruikt is staat in de bijlage: 'Development omgeving'.

De belangrijkste software waar gebruik van is gemaakt:

- ROS Kinetic omdat dit de meest gebruikte ROS versie is en omdat de Turtlebot3 hiermee het beste werkt.
- Op de robot is Rasbian Stretch gebruikt met ROS Kinetic. Dit is gedaan omdat ROS Kinetic is getest en werkt op Rasbian Stretch.
- Een computer met Ubuntu 16.04 omdat dit de Ubuntu versie is die ROS Kinetic ondersteund.
- Als planning is er voor Trello gekozen omdat het een Kanban plugin heeft en gemakkelijk in gebruik is.
- Voor GIT is er gebruik gemaakt van Git Cola als Ubuntu git gui en GitHub als repo. Git Cola en Github gratis en makkelijk op te zetten.

5.3. Planning

In dit hoofdstuk ga ik uitleggen wat mijn planning was en hoe het uiteindelijk van week tot week ging. De afstudeerperiode bestond uit 20 weken. Elke maandag, woensdag en vrijdag was er een standup met de bedrijfsbegeleider. Om de week was er ook een gesprek met de opdrachtgever waar er vragen gesteld konden worden en demo's werden gegeven. De eerste 5 weken zijn gebruikt voor het maken van een plan van aanpak, het opstellen van requirements en is er al een beetje onderzoek gedaan naar bijvoorbeeld ROS. Hierna zijn ongeveer 5 weken gebruikt voor het uitvoeren van onderzoeken en ontwerpen van de realisatie. Daarna zijn 8 weken gebruikt voor het realiseren van de robot. De overige weken zijn gebruikt als afronding en het bijwerken van het afstudeerverslag. Het hele project bestond vooral uit onderzoeken omdat er vaak nieuwe problemen kwamen uit onderzoeken en tijdens het realiseren.
Alle userstories zijn uiteindelijk in drie fases geplaatst. Dit is gedaan om agile te blijven en niet lang vast te zitten. In de bijlage "Userstories" is te zien welke userstories in welke planning zijn geplaatst. Door het in drie fases te plaatsen weet je ongeveer waar je moet zijn in welke week zonder ergens aan vast te zitten. Er is ervoor gekozen om het autonoom maken van de kaart als laatste fase te doen omdat dan alle andere functionaliteit af en getest is. Hierdoor werkt het hele project en heb je wat te laten zien en kan je de laatste weken bezig met het autonoom maken van de kaart.

Robot 1: Alle userstories met betrekking tot de server en het maken van een kaart met bluetooth beacons en autonoom navigeren in de kaart naar een bluetooth beacon.

App: Alle userstories met betrekking tot de app en het vinden en sturen van bluetooth beacons in de app.

Robot 2: Alle userstories met betrekking tot het autonoom kaart maken met behulp van de robot. Ook zit hier eventuele uitloop in. Extra functionaliteit kan hier toegevoegd worden en alle puntjes kunnen op de i worden gezet.

Weeknummers	Planning
Week 10 t/m Week 13	Robot 1
Week 14 t/m Week 15	App
Week 16 t/m Week 18	Robot 2
Week 19 t/m Week 21	Uitloop

Tabel 3 Projectplanning

Week 1 t/m 4:

De eerste 4 weken ben ik bezig geweest met het maken van een plan van aanpak en het opzetten van het onderzoek. In deze weken is er veel contact gezocht met de opdrachtgever. Dit lukte niet altijd omdat de opdrachtgever niet antwoorde op mails of telefoontjes. Daarom kon er in week 3 pas de requirement goedgekeurd worden. Ondertussen is er veel onderzoek naar het platform ROS gedaan om zo snel mogelijk een idee voor een mogelijke oplossing te vinden. Er is Ubuntu met ROS op mijn laptop gezet. Hierdoor kon ik al snel een virtuele Turtlebot laten rijden en programma's uitvoeren om te beginnen met het onderzoeken van ROS en de werking van een Turtlebot.

Week 5 t/m 8:

Tijdens week 5 is mijn plan van aanpak goedgekeurd door de opdrachtgever en school. Hierna is tot en met week 8 druk bezig geweest met onderzoeken. Het onderzoeken bestond voornamelijk uit informatie van het internet opzoeken, de informatie proberen werkend te krijgen op mijn computer, het testen door bijvoorbeeld in code te zetten en hierna in het onderzoek te zetten. Hierdoor ben ik

snel gewend geraakt aan ROS omdat ik al snel kleine programma's had gemaakt waarmee ik de virtuele robot aan kon sturen of uit kon lezen. Het uitvoeren van de onderzoeken was erg leerzaam omdat ik nog helemaal geen voorkennis had of een idee hoe ik de opdracht uit zou voeren. Na de onderzoeken had ik een ontwerp hoe ik de opdracht kon gaan realiseren. Er is uiteindelijk wel meer dagen aan onderzoek besteedt dan gedacht omdat er zo veel dingen uitgezocht en getest moesten worden. Uit veel onderzoeken kwamen weer nieuwe deelvragen. De weken hierna zijn er ook veel dingen toegevoegd aan de onderzoeken omdat er vaak nieuwe kennis bij kwam tijdens de realisatie dat ook invloed had op de onderzoeken.

Ook is er contact geweest met de opdrachtgever over het kopen van de Turtlebot3 en het krijgen van bluetooth beacons. De communicatie met de opdrachtgever liep nog steeds stroef omdat er lang op antwoord gewacht moest worden. Hierdoor is er afgesproken om iedere twee weken een gesprek te hebben over de vooruitgang. Dit verbeterde uiteindelijk de communicatie.

Week 9 t/m 12:

Door corona was week 9 de eerste week dat er vanaf thuis gewerkt werd. In deze week heb ik ook de robot en bluetooth beacons ontvangen. Ik ben eerst bezig geweest om op de robot een programma te maken die bluetooth low energy apparaten doorstuurt via ROS. Hierna zijn ook de correcte programma's geïnstalleerd zodat er handmatig rondgereden kon worden om een kaart te maken. Daarna heb ik op de computer een programma gemaakt die de gevonden bluetooth low energy apparaten van de robot uitleest, de beacons herkend, hierbij een positie pakt, in een lijst zet en opslaat. De positie van de beacons bleek verkeerd te zijn omdat de positie van de wielen zijn gebruikt in plaats van de positie van de robot in de kaart. Er is een programma gemaakt die deze positie uitzendt zodat de positie wel klopte. Alle programma's zijn in launch bestanden gezet zodat je met één command kan rijden, een kaart kan maken en beacons kan vinden.

Nu er een kaart met beacons gemaakt kon worden is er bezig geweest om een action client te maken die een UUID van een bluetooth beacon kan ontvangen, hierna de positie van de beacon uit de eerder gemaakte lijst kan halen en dit naar de robot kan sturen zodat de robot erna toe navigeert.

Toen dit eenmaal werkte is er een api op mijn computer gemaakt die een http call kan ontvangen en hierna via ROS een actie naar de action client kan sturen met bluetooth beacon.

Uiteindelijk moesten er nog veel kleine dingen veranderd worden om bijvoorbeeld de navigatie of het opslaan van de beacons echt goed werkend te krijgen daarom is er ook veel tijd besteed aan het testen van het hele proces. Er zijn ook demo's gegeven aan de opdrachtgever door filmpjes te maken van de werking en navigeren van de robot en het systeem.

Week 13:

Toen het hele proces naar behoren werkte ben ik in week 13 bezig geweest met het maken van de app. Dit ging aardig snel omdat hier al veel ervaring mee was. De app kon bluetooth beacons vinden en de dichtstbijzijnde doorsturen naar de api. Hierna reed de robot naar de positie van de doorgestuurde beacon. Er is een demo gegeven aan de opdrachtgever waar het hele proces van de robot, de app en de server op te zien was. Er is veel getest en in deze week kwam ik erachter dat de robot niet goed kon navigeren of bluetooth beacon kon opslaan op mijn kamer. Na dit uitgezocht te hebben bleek dit aan de wifi te liggen. Mijn kamer heeft geen goede wifi verbinding waardoor de robot soms connectie verliest. Hierdoor kan de robot niet data versturen of ontvangen van de computer waar alle programma's opdraaien waardoor de robot niet meer goed werkt. In de demo is

dit ook gezegd dat dit een restrictie van de robot bleek te zijn. De opdrachtgever heeft hierdoor een nieuwe requirement gesteld dat de robot na een signaal te hebben ontvangen moet kunnen werken zonder wifi.

Week 14/15:

Deze weken is deze nieuwe requirement van de opdrachtgever onderzocht en geïmplementeerd. Dit duurde langer dan verwacht omdat de structuur van de hele opdracht veranderd moest worden en er veel op de robot geïnstalleerd en ingesteld moest worden. Om alles zonder wifi te laten werken moeten bijna alle programma's op de robot draaien. Deze programma's kosten vaak veel rekenkracht waardoor veel parameters veranderd moesten worden zodat de computer op de robot het aankon en bij kon houden.

In week 15 ben ik ook drie keer naar het kantoor geweest om te testen. De eerste twee keer is het niet gelukt om te testen omdat de wifi van het bedrijf te streng beveiligd was en omdat er gekeken moest worden hoe er een nieuw netwerk aangemaakt kon worden. Dit is uiteindelijk gedaan met een Ubuntu hotspot. Tijdens deze testen is de wifi van de robot ook kapot gegaan. Dit duurde ook een dag op het bedrijf om te repareren. De derde keer kon er echt getest worden en werkte de robot zeer goed en konden er een paar parameters veranderd worden om de navigatie van de robot te verbeteren. De kaart die de robot had gemaakt zag er goed uit en de robot kon goed bluetooth beacons vinden er hierna toe navigeren. Hiervan is een filmpje gemaakt en laten zien tijdens een demo. De opdrachtgever was tevreden met het resultaat.

Week 16/17:
Deze weken ben ik bezig gegaan met het autonoom maken van een kaart. Omdat het programma zonder wifi moest werken moesten er wel weer veel parameters veranderd worden omdat het navigeren en maken van een kaart beide veel rekenkracht kost. Gelukkig werkte alles aardig snel en moest er vooral veel getest worden. Er is een programma gemaakt die naar de robot een signaal kon sturen zodat de robot autonoom de hele kamer in kaart begint te brengen. Tijdens het testen kwamen wel dingen na voren die verbeterd konden worden aan de navigatie van de robot in kleine ruimtes. Hierdoor heb ik gelijk alle benodigde parameters lokaal gezet zodat alles op één plek staat en makkelijker veranderd kan worden.

In week 17 ben ik vooral bezig geweest met het verbeteren en maken van mijn afstudeerverslag. De weken hiervoor heb ik veel feedback van het verslag gekregen van mijn school en bedrijfsbegeleider. Origineel stond in de planning dat ik deze week ook nog bezig zou zijn met de robot maar dit is verschoven naar week 19 en 20.

Week 18:

Deze week heb ik te horen gekregen dat ik groenlicht heb en dus mijn project mag verdedigen. Wel had ik veel feedback waar ik bijna de hele week bezig ben geweest om te verwerken. Eind deze week heb ik mijn definitieve afstudeerverslag bij school ingeleverd.

Week 19 en 20:

Deze weken ga ik bezig met testen. Ook ga ik bezig met het maken van een presentatie en oefenen voor de verdediging.

6. Afbakening

Nadat de opdracht, probleemstelling, gewenste eindsituatie en aanpak duidelijk en goedgekeurd zijn door de opdrachtgever is er een lijst opgesteld met afbakeningen, requirements en randvoorwaarden om voor alle betrokken partijen duidelijk te maken te wat de opdracht was, wat er gemaakt zou worden en of alle partijen wel dezelfde visie hadden. De afbakeningen maken duidelijk wat er binnen en buiten het project valt en welke eisen belangrijker zijn dan andere eisen. Het maken van de randvoorwaarden en eisen is tegelijkertijd gedaan als het uitvoeren van de onderzoeken. Dit is gedaan omdat tijdens het onderzoeken snel nieuwe eisen en problemen naar voren kwamen. Dit kon dan snel met de opdrachtgever besproken worden en in requirements en randvoorwaarden veranderd worden.

Er is ook met de opdrachtgever besproken dat er voor een geslaagd project minimaal alle musts afgerond moeten zijn. Verder moest er ook aan alle randvoorwaarden voldaan worden voor zowel het project als de betrokken partijen.

6.1. Requirements

Er is in het begin van het project veel met de opdrachtgever gezeten voor het opstellen van requirements voor zowel de app, de server, als de robot die gemaakt moest worden. Nadat de requirement door de opdrachtgever zijn goedgekeurd is dit ook in het een plan van aanpak gezet en naar school gestuurd.

In week 13 werd er ook een nieuwe eis bekend bij de opdrachtgever. “De robot moet kunnen navigeren zonder wifi”. De robot moest na een signaal te hebben ontvangen niet meer afhankelijk zijn van wifi en autonoom kunnen navigeren naar een bluetooth beacon. Hier is eerst weer onderzoek naar gedaan en later opgenomen in de requirements.

Er moet uiteindelijk software voor de robot, een server en een app ontwikkeld worden.
Daarom is ervoor gekozen om de requirements hiervan op te splitsen. Dit is gedaan omdat het ook drie verschillende technieken en hardware betreft.

5.1.1 App

Functionele requirements

Must

- De app moet bluetooth beacons via bluetooth kunnen vinden die binnen de acceptabele bluetooth radius vallen van de hardware. Bij de bluetooth beacon is dat 15 meter indien het signaal onverstoord is.
- De app moet de dichtstbijzijnde bluetooth beacon kunnen tonen indien die er is.
- De applicatie moet via wifi een bericht met de dichtstbijzijnde bluetooth beacon kunnen sturen naar de server.

Should

- Via de app moet het zichtbaar zijn of de robot onderweg is.
- De gevonden bluetooth beacons moeten zichtbaar zijn in de app.

Niet functionele requirements

Must

- De app moet werken op Android met tenminste minimum api level 26.

Could

- De app moet dezelfde functionaliteit bieden als Android op IOS 13.

5.1.2. Robot

Functionele requirements

Must

- De robot moet bluetooth beacons kunnen lokaliseren doormiddel van bluetooth binnen de acceptabele bluetooth radius van de hardware. De bluetooth radius bij de bluetooth beacon is 15 meter indien het signaal onverstoord is.
- Bij het instellen van de robot moet de robot rond kunnen rijden en een kaart kunnen maken van de omgeving.
- De robot moet de eerste verdieping van het kantoor van ICT Group in Deventer in één nacht ik kaart kunnen brengen.

Should

- De robot na een signaal van de server te hebben ontvangen, navigeren en een kaart kunnen maken zonder wifi.
- Bij het instellen van de robot moet de robot rond kunnen rijden en bluetooth beacons ontdekken en hiervan de positie opslaan.
- De robot moet obstakels kunnen ontwijken terwijl hij een kaart aan het maken is.
- De robot moet obstakels kunnen ontwijken terwijl hij navigeert naar een bluetooth beacon.

Could

- De robot moet bij het instellen zelf stoppen wanneer de kaart compleet is en alle bluetooth beacons bevat.

Won't

- Bij een ongeval moet de dichtstbijzijnde robot naar het ongeluk toerijden ingeval dat er meerdere robots zijn.

Niet functionele requirements

Must

- De robot moet instelbaar zijn met een SSH verbinding.
- De robot moet gebruik maken van ROS Kinetic.

Should

- De robot moet de positie van bluetooth beacons binnen 3 meter nauwkeurig bepalen.
- In geval van een defect moet een nieuwe robot niet opnieuw een kaart maken of bluetooth beacons vinden.

Could

- De robot kan signalen krijgen via een webpagina.

Won't

- Het bedrijf moet meerdere robots rond kunnen laten rijden.

5.1.3. Server

Functionele requirements

Must

- De server moet signalen van een mobiel kunnen ontvangen via wifi.
- De server moet na een wifisignaal gekregen te hebben, de robot automatisch kunnen opstarten en navigeren naar de correcte bluetooth beacon doormiddel van een eerder gemaakt kaart met opgeslagen bluetooth beacons.

- De server moet doormiddel van ROS, signalen naar de robot kunnen sturen.

Niet functionele requirements

Must

- De server moet gebruik maken van Ubuntu LTS 16.04 met ROS Kinetic.
- De server moet voorzien zijn van wifi.

5.2. Randvoorwaarden

Tijdens het uitvoeren van de onderzoeken is naar voren gekomen dat de robot goed werkt als er aan bepaalde voorwaarden voldaan wordt. Denk bijvoorbeeld aan de plaatsing van de beacons of obstakels en aan dat de robot alleen LIDAR bezit. Om de opdracht zo haalbaar mogelijk te maken is ervoor gekozen om eerst heel streng aan die voorwaarden te voldoen. Wanneer de POC werkt kan er gekeken worden of we de opdracht kunnen uitbreiden zodat de robot in meer situaties werkt. In kopje 10 'Aanbevelingen' worden er wat aanbevelingen gedaan om niet meer afhankelijk te zijn van deze randvoorwaarden.

De project afspraken met stakeholders waar alle stakeholders zich moeten houden:

- De uitvoerder krijgt de turtlebot3 tot beschikking waarmee gewerkt kan worden.
- Er mag van worden uitgegaan dat de geschetste kaart/ situatie van de omgeving van de robot niet veranderd en dus statisch is.
- De uitvoerder krijgt bluetooth beacons tot beschikking (Minimaal twee).
- De product owner is elke week beschikbaar voor het geven van feedback.
- Er zijn genoeg bespreekmomenten met de aanspreekpunten (product owner, bedrijfsbegeleider en schoolbegeleider)
- De uitvoerder besteed de afgesproken tijd in het project zoals beschreven in het contract.
- Het distributiecentrum moet om de 7 meter een herkenbaar punt bevatten. Zodat de robot zijn positie kan blijven bepalen.
- De positie van de bluetooth beacons blijft statisch. De robot zal opnieuw ingesteld moeten worden als een positie verandert.

De bluetooth beacons:

- Moeten op dezelfde hoogte staan.
- Mobiel staat altijd in bereik van een bluetooth beacon.
- Moeten niet achter obstakels staan.
- Moet van hetzelfde type en merk zijn.

Het distributie centrum:

- Obstakels moeten allemaal op laserhoogte van de robot zichtbaar zijn. Dus geen kabels of opstapjes.
- De robot moet zich niet vast kunnen rijden op de vloer. De vloer moet dus redelijk vlak zijn. Dus geen kabels of opstapjes.

De robot:

- De start positie van de robot staat vast.
- De robot is opgeladen bij het krijgen van een signaal.

6. Onderzoeken

6.1. Aanpak

Over de turtlebot3 en het operating system ervan 'ROS' is veel informatie beschikbaar. Hierover moest nog veel onderzoek gedaan worden. Er was in het begin van dit afstudeerproject nog geen ervaring of kennis van ROS, het besturen van robots en het werken met embedded systemen. Hier is rekening gehouden tijdens het onderzoeken door ook over de werking van de systemen (wat voor de hand ligt voor mensen met ervaring) onderzoeksvragen te stellen.

Er was gelukkig al veel kennis en software beschikbaar omdat deze platformen worden gebruikt door veel open source bijdragers. Ook zijn er al veel onderzoeken op het internet te vinden die dezelfde vragen beantwoorden. Daarom heb ik ervoor gekozen om van de onderzoeksmethodes 'Community research'[24] en 'Literature study'[25] gebruik te maken. Dit betekent dat er bestaande data, onderzoeken en feiten zijn verzameld en op basis hiervan een conclusie is getrokken op de gestelde hoofd- en deelvragen. In dit onderzoek zijn antwoorden die van het internet kwamen ook altijd geprobeerd of geprogrammeerd. Wanneer het bijvoorbeeld duidelijk werd dat er een express server met rosnodejs gebruikt moest worden om met de robot te communiceren, is dit ook daadwerkelijk geprogrammeerd om te kijken of ik dit werkend kon krijgen. Hierdoor kwamen vaak nieuwe vragen naar boven of kon ik betere conclusies maken. Ook kon hier verder op worden doorgebouwd wanneer er echt gerealiseerd moest worden.

De hoofd- en deelvragen zijn opgesteld tijdens het maken van de afbakening (Requirements en randvoorwaarden) van het project. Hierdoor zijn er relevante hoofd- en deelvragen opgesteld en konden met de uitkomsten van de vragen ook de afbakening nog veranderd worden, als er bijvoorbeeld een requirements van de opdracht niet overeenkwam met de conclusie van een hoofd/deelvraag. Dit is zeer nuttig omdat er nog niet veel kennis was over de technologieën die gebruikt zouden worden en hierdoor een compleet overzicht is gemaakt.

6.2. Hoofd- en deelvragen

Voor het complete onderzoek kunnen de bijlages: "Onderzoek App" en "Onderzoek Robot" erbij gepakt worden.

Voor het maken van de vragen is er goed gekeken naar de aanwezige kennis. Er was weinig ervaring met ROS, Turtlebot en Bluetooth beacons. Daarom zijn er hier ook vragen voor gemaakt. Eerst zijn alle deelvragen onderzocht om hierna antwoord te kunnen geven op de hoofdvraag.

De hoofd- en deelvragen zijn:

App

1. Hoe kan de app het ID van de dichtstbijzijnde bluetooth beacon naar de robot sturen? (Kopje 6.3)
   1.1. Hoe kan de app gebruik maken van bluetooth? (Kopje 6.3.1.)
   1.2. Hoe kan de dichtstbijzijnde bluetooth beacon gevonden worden? (Kopje 6.3.2.)
   1.3. Hoe kan er het beste een hulpverzoek naar de robot verstuurd worden? (Kopje 6.3.2.)

Robot

De vragen 1.1 en 1.2 zijn in de opdracht omschrijving gezet zodat het hele document leesbaarder is.

1. Hoe laat ik een robot autonoom navigeren naar een bluetooth beacon? (Kopje 6.4)
   1.1. Hoe werkt de Turtlebot3? (Kopje 3.3.1)

1.2. Hoe communiceert ROS met de Turtlebot3? (Kopje 3.3.2)
1.3. Wat is de beste oplossing om de Turtlebot3 een kaart te laten maken van de omgeving? (Kopje 6.4.1)
    1.3.1. Hoe kan de Turtlebot3 het beste bluetooth beacons in deze kaart plaatsen? (Kopje 6.4.1.1)
    1.3.2. Hoe kan de Turtlebot3 het beste navigeren naar de bluetooth beacons in deze kaart? (Kopje 6.4.1.2)
1.4. Hoe kan de turtlebot3 het beste een commando van een mobiel ontvangen? (Kopje 6.4.2)
    1.4.1. Hoe kan de turtlebot3 acties ondernemen op basis van het ontvangen commando? (Kopje 6.4.2.1)
1.5. Hoe kunnen gemaakte kaarten gedeeld worden zodat de Turtlebot vervangen kan worden? (Kopje 6.4.3)
1.6. Wat is de beste manier om de software die geschreven wordt te testen? (Kopje 6.4.4)
    1.6.1. Wat zijn de beste oplossingen voor virtueel testen? (Kopje 6.4.4.1)
    1.6.2. Wat zijn de beste oplossingen voor fysiek testen? (Kopje 6.4.4.2)
1.7. Wat is de beste manier om de robot te laten navigeren zonder wifi? (Kopje 6.4.5)

6.3. Hoe kan de app het ID van de dichtstbijzijnde bluetooth beacon naar de robot sturen?

De belangrijkste functie van de app is het sturen van een hulpverzoek met een bluetooth beacon naar de robot. Hierom is dit ook de hoofdvraag geworden. De deelvragen zijn andere vragen die belangrijk zijn om te hoofdvraag te beantwoorden.

Het antwoord op de hoofdvraag staat onder het kopje 6.3.3 'Conclusie'

6.3.1. Hoe kan de app gebruik maken van bluetooth?

De app zal moeten werken op Android. Dit is een must. Een Could is dat de app ook moet werken op IOS. Omdat de app niet het belangrijkste onderdeel van het project is zal het handiger zijn om hier een hybride app voor te ontwikkelen. Een hybride app heeft toegang tot native functionaliteit zoals bluetooth en gps. De hybride app kan eerst klaar gemaakt worden voor Android maar kan zonder te veel veranderingen ook met IOS werken. Het voordeel van native apps is dat ze sneller draaien en een vertrouwde omgeving voor de gebruiker geven. Het zal dan wel twee keer zoveel tijd kosten omdat er twee apps gemaakt moeten worden. Ook heb ik geen ervaring in het maken van native iOS apps en hierdoor kost dit onnodig veel tijd voor een POC. Met iets zoals Cordova[15] kan gemakkelijk je web app omgezet worden in een echte app en van native functionaliteit in IOS en Android gebruik gemaakt worden.

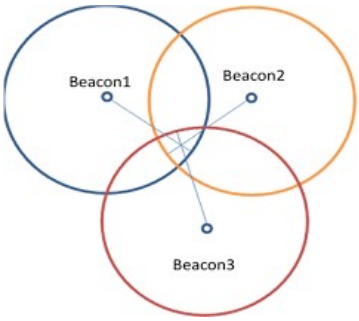
Er wordt ervoor gekozen om Vue Quasar te gebruiken om het prototype zo snel mogelijk af te hebben. De robot vereist veel nieuwe kennis. Hierdoor wordt het lastig om voor de voorkant ook nieuwe technologieën te leren. Quasar maakt gebruik van het design principe van google. Hierdoor blijft de blijft de app vertrouwd voor mensen omdat iedereen wel google apps heeft gebruikt. Er is ook gekeken naar een paar alternatieven. Ionic[26] maakt gebruik van React, heeft geen goede Vue ondersteuning en kost geld voor veel features. React native [27] maakt gebruik van React waar te weinig ervaring mee is en waar we ook niet te veel tijd aan willen besteden om te leren.

6.3.2. Hoe kan de dichtstbijzijnde bluetooth beacons gevonden worden?

Er zijn twee mogelijkheden om de dichtstbijzijnde bluetooth beacon te ontdekken: Op basis van de sterkte van het signaal of trilateration.

Als er een bluetooth beacon ontdekt wordt kan een mobieltje of robot de signaalsterkte/ de RSSI (Received Signal Strength Indicator) berekenen. De RSSI kan meestal tot 40-50 meter berekend worden. Hoe dichter bij de 0 dBm er wordt gemeten hoe sterker het signaal is. Hoe verder weg de mobiel van de bluetooth beacon is hoe onbetrouwbaarder de RSSI. Het is erg belangrijk dat de bluetooth beacons op plekken staan waar het signaal niet geblokkeerd kan worden, dit kan er namelijk voor zorgen dat niet de dichtstbijzijnde bluetooth beacon het beste signaal uitzendt. De bluetooth beacons moeten ook op gelijke hoogte staan omdat dit het uitzend signaal ook veranderd. Ook is het handig om bluetooth beacons vaker te meten om een gemiddelde te krijgen omdat individuele metingen soms uiteenlopen. Als hieraan wordt gehouden kan ervan uit worden gegaan dat het dichtstbijzijnde bluetooth beacon gevonden kan worden.

Met trilateration kan de positie van de robot of mobiel bepaald worden doormiddel van het gebruik van meerdere bluetooth beacons. Er worden dan met beacons meerdere overlappende zones met signalen gemaakt. Er kan dan worden gekeken in welke overlappende zone de telefoon zich bevindt. Als de posities van de bluetooth beacons bekend zijn kan precies de positie van een robot op mobiel bepaald worden.



Figuur 6 Werking van trilateration.

Een probleem is dat we de positie van de bluetooth beacons niet weten. De app kan dus niet op basis van trilateration de dichtstbijzijnde bluetooth beacon vinden. Hierdoor zou de robot eerst moeten rondrijden om de posities van de bluetooth beacons te bepalen en hiervoor kunnen we alleen van RSSI gebruik maken. Als we RSSI gebruiken om de positie van de beacons te bepalen hebben we trilateration ook niet nodig. Als dit wel bekend zou zijn zou het de opdracht onnodig complex maken omdat we voor het vinden van de positie van de bluetooth beacons ook alleen maar van RSSI gebruik hebben gemaakt.

6.3.3. Hoe kan er het beste een hulpverzoek naar de robot verstuurd worden?

Een bluetooth beacon zendt een signaal uit. Dit signaal bevat naast de RSSI ook een UUid van de bluetooth beacon. Om de dichtstbijzijnde bluetooth beacon te vinden kunnen we van de sterkste RSSI gebruik maken. De robot maakt ook een lijst met bluetooth beacons en de positie van het beste signaal. De mobiel hoeft alleen de beste bluetooth beacon door te sturen en de robot kan de positie van het beste signaal van de bluetooth beacon pakken. Hierdoor kan de robot altijd navigeren naar een goede plek omdat de robot dat eerder ook al heeft gedaan. In het onderzoek over de robot zijn we erachter gekomen dat we de robot het beste via een externe computer met een server kunnen aansturen. De server kan een http call ontvangen en berichten via ROS met de robot kunnen communiceren omdat ze in connectie met elkaar staan. De beste manier om via een mobiel een bericht te sturen naar een server is door een api call te sturen met hierin het UUid van de bluetooth beacon. De server kan dit doorsturen naar de robot en de robot kan hieruit de positie van de bluetooth beacon bepalen omdat hij alle bluetooth beacons op heeft geslagen met positie.

6.3.4. Conclusie

Omdat bluetooth beacons onbetrouwbaar zijn omdat signalen makkelijk verstoord worden is ervoor gekozen om hier voorwaarden aan te koppelen. Dit is gedaan om de opdracht haalbaar te maken in de tijd. Als de proof of concept werkt met deze voorwaarden kan er gekeken worden wat er uitgebreid en complexer kan worden gemaakt. De voorwaarden voor de bluetooth beacons:

- Moeten op dezelfde hoogte staan.
- Mobiel staat altijd in bereik van een bluetooth beacon.
- Mogen niet achter obstakels staan.
- Moeten hetzelfde signaal uitzenden. (zelfde type en merk)
- Moeten bij beide kanten van een stelling geplaatst worden om te voor komen dat de robot in het verkeerde gangpad rijdt.

Dit is gedaan zodat een mobiel makkelijk de dichtstbijzijnde bluetooth beacon kan vinden. Als dit niet wordt gedaan kan het zijn dat een bluetooth signaal van een beacons die verder weg is sterker is dan eentje die dichterbij staat. Er wordt een app gemaakt in Vue Quasar met Cordova. Hier is eerder mee gewerkt waardoor er snel een werkend prototype kan worden gemaakt. Met de bluetooth plugin van Cordova kan de dichtstbijzijnde bluetooth beacon gevonden worden. Elk bluetooth beacon heeft hetzelfde adres, waardoor ze herkenbaar zijn. De dichtstbijzijnde bluetooth beacon wordt gevonden door de gemiddelde signaalsterkte(RSSI) te gebruiken. Bluetooth beacons worden herkend aan hun naam, welke hetzelfde is voor elke beacon die we gebruiken. Het bluetooth beacon wordt met UUid naar een server gestuurd zodat de server weet waar hij de robot naartoe moet sturen.

6.4. Hoe laat ik een robot autonoom navigeren naar een bluetooth beacon?

De robot moet autonoom naar een bluetooth beacon kunnen navigeren. Dit is de belangrijkste functie van de robot en is ook de hoofdvraag geworden. Om dit te doen zijn extra stappen nodig zoals het maken van een kaart en het vinden van bluetooth beacons. De deelvragen bestaan uit de vragen die kwamen uit de extra stappen, de vragen die kwamen uit de eisen en vragen die beantwoord moeten worden om de hoofdvraag te kunnen beantwoorden.

Het antwoord op de hoofdvraag staat onder het kopje 6.4.2 ‘Conclusie’

6.4.1. Wat is de beste oplossing om de turtlebot3 een kaart te laten maken van de omgeving?

De Turtlebot3 beschikt over een 360 Laser Distance Sensor LDS-01 wat gebruikt maakt van LIDAR[7], wat een technologie is waarmee een afstand tot een object of oppervlakte bepaald kan worden doormiddel van het gebruik van laserpulsen. Er wordt een laserpuls gestuurd, botst met een object en komt na een tijd weer binnen. Met die tijd kan de afstand tot het object berekend worden.

Met de 360 graden sensor en LIDAR kan er een kaart van de omgeving gemaakt worden met behulp van SLAM(Simultaneous localization and mapping)[8]. SLAM is een techniek waarbij een robot een kaart probeert te maken en zichzelf ondertussen op die kaart terug probeert te vinden.
Hierdoor kan de robot ook de onthouden waar hij zich bevindt. Als de kaart klaar is kan die kaart gebruikt worden om erin te navigeren. SLAM zoekt herkenbare punten op de kaart om de positie van de robot bij te houden. Zie link (https://nl.wikipedia.org/wiki/Simultaneous_localization_and_mapping#/media/Bestand:LIDAR-scanned-SICK-LMS-animation.gif) voor een geanimeerde afbeelding die de werking van SLAM met LIDAR laat zien.

SLAM gebruikt de metingen van LIDAR. De LIDAR-sensor draait rond en houdt alle gemeten afstanden bij met een punt in een x en y grafiek waarbij x en y de coördinaten zijn waar de laser terugkaatste. Als dit wordt gedaan is al snel de omtrek van een kamer te zien. Als de LIDAR zich verplaatst kan SLAM de positie blijven bepalen omdat SLAM de metingen die nu worden gemaakt kan vergelijken met de kaart. Denk bijvoorbeeld aan een gangpad van 2 meter breed. Als de LIDAR van positie verandert kan SLAM alsnog weten of hij midden in het gangpad ligt of niet.

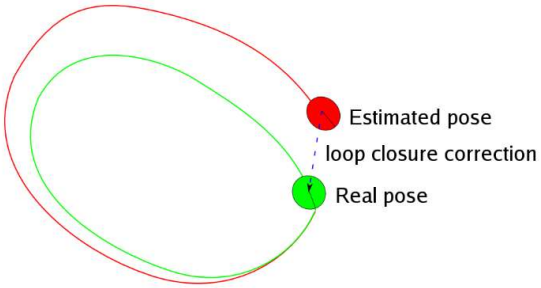
SLAM is een belangrijk onderdeel van dit project omdat de robot op basis van SLAM een kaart gaat maken, hier bluetooth beacons in positioneert en hier naartoe navigeert. Met de gemaakte kaart weet de robot de snelste route naar de beacons en weet de robot waar alle muren zijn die ontweken moeten worden. Er zijn verschillende SLAM-algoritmes/ programma's beschikbaar voor de Turtlebot3 en ROS. Hieronder een klein onderzoek en vergelijking van de algoritmes en naar wat het beste algoritme is voor dit project.

Gmapping VS Cartographer VS SLAM Hector

Er zijn drie veel gebruikte SLAM-algoritmes Gmapping, Cartographer en Hector SLAM[6]. Gmapping gebruikt odometry en laserscan voor het maken van een kaart. Odometry is data over de robot zoals bewegingen die hij maakt. Het gevolg hiervan is dat Gmapping vaak beter weet waar de robot is en hiermee een betere kaart kan maken. Gmapping kan de positie van de robot ook verbeteren doormiddel van AMCL[32] wanneer de robot bijvoorbeeld vast komt te zitten of wegglipt. De wielen bewegen dan wel waardoor een robot zonder AMCL kan denken dat de robot verder vooruit is gegaan. Door het bijstellen van de positie blijft de kaart en de positie van de robot hierin correct. Gmapping zit ook standaard bij ROS en werkt 'out of the box' goed met programma's als RVIZ en Gazebo. Hector SLAM gebruikt alleen de laser scan voor het maken van een kaart. Hierdoor kan de positie mogelijk slechter bepaald worden. Als de robot bijvoorbeeld vast komt te zitten stelt Hector de data niet bij en klopt de kaart niet meer omdat Hector de wielen gewoon zag draaien en denkt dat de robot vooruit is gegaan. Cartographer gebruikt ook AMCL, odometry en laserscan voor het maken van een kaart en werkt hetzelfde als Gmapping.

Wat het beste algoritme is ligt heel erg aan de situatie. Het ligt heel erg aan de robot die is gebruikt en de manier van bewegen van de robot. Een algoritme kan beter zijn als de robot een rondje probeert te rijden en de ander is beter als de robot zigzagt. In bron [9] is hier door een universiteit ook al een uitgebreid onderzoek naar gedaan. Daar zijn alle algoritmes getest met dezelfde bewegingen en posities. Hieruit kwam dat Gmapping vaak de mooiste en accurate kaarten produceerde en Cartographer en Hector SLAM op nummer twee. Om de beste oplossing voor de Turtlebot3 te vinden zijn de drie algoritmes getest met verschillende opties en situaties. Om de algoritmes zo goed mogelijk te testen is er gebruik gemaakt van rosbag[28]. Met rosbag kan alle data worden opgenomen en opnieuw worden afgespeeld. Zo is er bij elke test die er is gedaan dezelfde bewegingen gemaakt. Hierdoor hebben we een betrouwbaar een experiment wat reproduceerbaar is. De algoritmes zijn gesimuleerd en fysiek getest. Hierdoor weet je ook hoe goed de simulatie van de robot werkt. Met de gesimuleerde omgeving kan je ook elke omgeving creëren die je wilt, hierdoor kunnen er verschillende situaties getest worden.

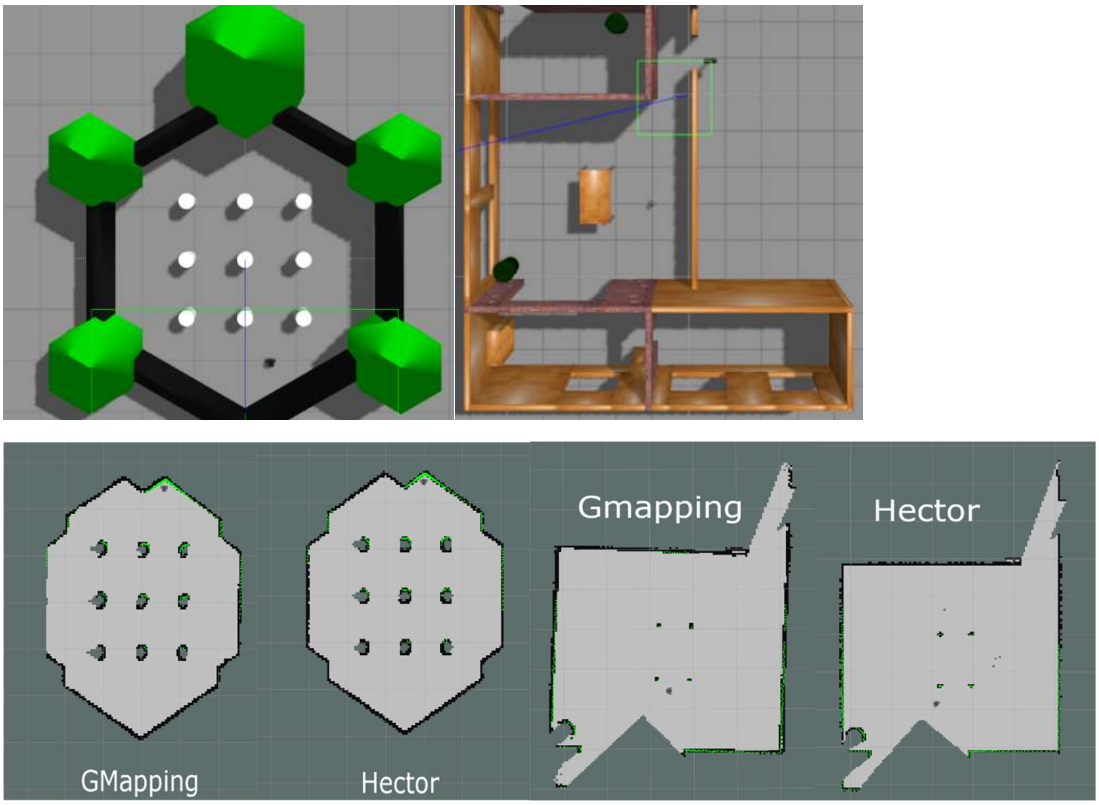
Een van de belangrijkste dingen die er moeten gebeuren tijdens het maken van een kaart is een 'loop closure'[12]. Een loop closure komt voor als je op hetzelfde punt uitkomst waar je begon. De robot ziet dan een bekende plek en kan hierdoor de kaart recht trekken en de fouten van de sensors kan verwijderen. Met het handmatig rijden kan je vaker een loop closure veroorzaken waardoor de kaart steeds preciezer wordt.



Figuur 7 Voorbeeld van een loop closure

Virtuele vergelijking

Met het programma Gazebo[29] is het mogelijk om werelden in te laden en een robot in deze wereld te plaatsen en simuleren. Gazebo zorgt zelfs voor wat speling in de wielen voor een realistischer resultaat. Jammer genoeg is het niet gelukt om het Cartographer algoritme aan de praat te krijgen. Er waren problemen bij het uitlezen van data wat wel goed getest is met de echte Turtlebot. De experimenten zijn goed gelukt omdat ze allemaal dezelfde uitkomst en de verwachte resultaten hebben. De algoritmes hebben allebei bruikbare kaarten opgeleverd met maar weinig verschil. Hector zet wel sneller muren neer en maakt ze rechter.



Figuur 8 Hector vs Gmapping uitkomst.

Fysieke vergelijking

De fysieke vergelijking is in twee verschillende ruimtes gedaan. De eerste ruimte is een kleine kamer met een gladde vloer. De tweede ruimte is een grote ruimte met veel verschillende objecten en

heeft een hobbelige vloer. Het Cartographer algoritme werkte deze keer wel. De algoritmes zijn out of the box gebruikt. Er zijn vele opties die er per algoritme veranderd kunnen worden. Maar om het testen makkelijk te maken zijn de standaard opties gebruikt. Alle algoritmes hebben alweer de kaart gemaakt zoals verwacht. In de fysieke omgeving kwam naar voren waarom Hector SLAM geen goeie oplossing is. Door de hobbelige vloer kwam de data van de sensoren en de positie van de robot niet meer overeen en is de kaart onbruikbaar geworden. Dit komt omdat de wielen vaak wegglippen, Gmapping en Cartographer hadden gelijkmatige resultaten.

Figuur 9 Kamer 1 met uitkomst vergelijkingen.

Figuur 10 Kamer 2 met uitkomst vergelijkingen

Conclusie fysiek en gesimuleerde omgeving

Gmapping	Hector SLAM	Cartographer
- Schiet vaker uit	+ Rechte lijnen	- Werkt niet in gesimuleerde omgeving
- Maakt kaart schever	+ Strakkere kaart	

Tabel 4 Plus en minpunten algoritmes gesimuleerde omgeving.

Gmapping	Hector SLAM	Cartographer
+ Werkte goed in beide omgevingen	- Werkt totaal niet bij kleine tegenslag	+ Werkte goed in beide omgevingen
+ Redelijke strakke kaart	+ Strakke kaart	+ Strakke kaart
+ Veel opties	+ Veel opties	+ Veel opties
+ Kost weinig computer kracht	+ Rechte lijnen	+ Zachte kaart
- Schiet vaker uit	- Gebruikt geen odometry	+ Zet snel obstakels als muur neer.
- Maakt kaart schever		- Kost veel computer kracht.
		- Werkt alleen op Ubuntu 16.04

Tabel 5 Plus en minpunten algoritmes fysieke omgeving.

In de gesimuleerde omgeving leek Hector SLAM de winnaar. Maar met de fysieke robot bleek het al snel dat Hector SLAM goed kaarten kan maken in een perfecte omgeving maar compleet de fout in gaat als er ook maar iets verkeerd gaat. Hierdoor valt Hector compleet af als algoritme. Gmapping en Cartographer hebben ongeveer dezelfde resultaten. Gmapping schiet vaker uit en zet niet hele rechte lijnen terwijl Cartographer niet uitschoot en zachte lijnen zetten tegenover hele rechte. Cartographer wacht met muren zetten. Hierdoor worden vooral grote ruimtes nauwkeurig in kaart gebracht omdat hij tijdens rijden de muren extra kan controleren. Hierdoor maakt Cartographer net wat mooier ogende kaarten als Gmapping. Uiteindelijk is er voor Gmapping gekozen omdat er uit onderstaand onderzoek is gebleken dat het SLAM algoritme op de robot moet draaien als we een kaart willen maken zonder wifi. Dit kan niet met Cartographer omdat het Rasbian niet ondersteund en te veel rekenkracht kost. Ook werkt Gmapping goed met Frontier Exploration wat hieronder wordt uitgelegd.

Automatisch een kaart maken

Voor het automatisch maken van een kaart kan het Frontier Exploration algoritme gebruikt worden. Het Frontier Exploration algoritme kan naar SLAM algoritmes luisteren om zo te kijken welk gebied nog niet verkend is. Het algoritme kan berichten sturen naar de move_base van de robot en kan zo de robot vertellen naar de nog niet verkende gebieden te navigeren waardoor ze in kaart worden gebracht door het SLAM algoritme. Frontier Exploration rijdt naar het gebied waar nog het meeste verkend kan worden. ‘frontier’ staat voor het gebied tussen bekend gebied en onbekend gebied. Wanneer er geen frontiers meer zijn is de kaart compleet. In Frontier Exploration kan je een oppervlakte meegeven wat helemaal verkend moet worden of een leeg bericht wat betekend dat het algoritme pas stopt als alles verkend is.

6.4.1.1. Hoe kan de turtlebot3 het beste bluetooth beacons in deze kaart plaatsen?

Er moet een Node geschreven worden die continue kijkt wat voor bluetooth signalen er binnen komen. Als er een bluetooth beacon gevonden wordt zal er een speciaal signaal uitgezonden moeten worden zodat de bluetooth beacon met de locatie opgeslagen kan worden.

Dit kan gedaan omdat er ook een RSSI (Received Signal Streng Indicator) wordt meegegeven van het signaal af. Met RSSI wordt de sterkte van het signaal meegegeven, hiermee kan de positie redelijk bepaald worden. Tijdens het maken van een kaart moet dit uiteindelijk gebeuren.

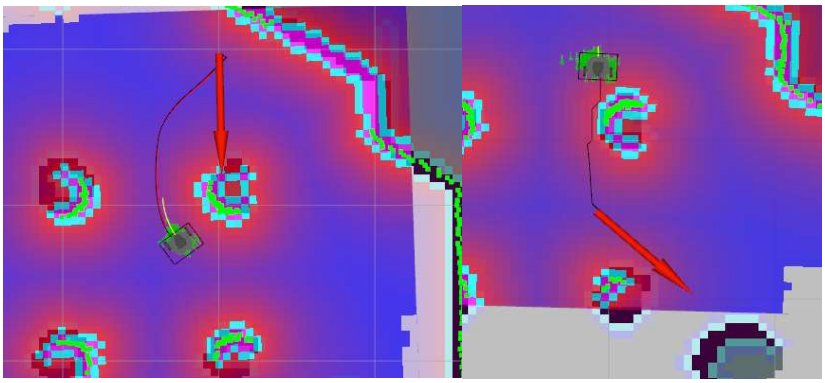
Om van Bluetooth low energy gebruik te maken op de Raspberry pi kan van de python package Bluepy gebruik gemaakt worden. Om hiermee te coderen moeten de Bluetooth rechten wel goed staan. Als je programma's niet als root wil uitvoeren moet je bluetooth permissies aan de pi gebruiker geven.

Tijdens het maken van de kaart staan er twee Nodes aan die bluetooth beacons met de locatie opslaan. Node 1 moet aanstaan op de Turtlebot omdat het de hardware van de robot gebruikt en alle ontvangen signalen van bluetooth doorsturen. Node 2 moet de gemiddelde signaalsterkte van bluetooth beacons bijhouden met de positie hiervan. De positie met het beste signaal wordt uiteindelijk opgeslagen. Zo kan je ervan uitgaan dat de robot dicht naar de beacon kan rijden.

Tijdens het rijden maakt het niet uit dat het RSSI per beacon verschilt. Als er een lijst wordt bijgehouden met alle bluetooth beacons en de beste RSSI en positie zal er per bluetooth beacon de kortste RSSI en dichtstbijzijnde positie worden opgeslagen. Wel is het belangrijk dat de beacons niet worden geblokkeerd door objecten omdat dit het signaal blokkeert. We hebben nu een lijst met beacons waarin de beste RSSI staat opgeslagen met de positie waar de robot toen reed. We willen niet de positie van de beacon maar van de robot omdat we zeker weten dat de robot naar de beacon toe kan rijden omdat de robot er tijdens het maken van de meting ook naartoe is genavigeerd.

6.4.1.2. Hoe kan de turtlebot3 het beste navigeren naar de bluetooth beacons in deze kaart?

Het is belangrijk dat de positie van de robot tijdens het beginnen van het maken van de kaart hetzelfde is als de positie tijdens beginnen van het rijden naar ongeluk. Dit scheelt het instellen en uitzoeken van de positie. De beste oplossing om te navigeren is om een Node te maken die naar waypoints kan navigeren. De waypoints zijn dan de positie van de eerder gemaakte bluetooth beacons. In deze Node kan een functie gemaakt worden die een bluetooth UUid kan ontvangen om vervolgens naar de correcte bluetooth beacon te navigeren met behulp van de Move Base functies die al in de Turtlebot zitten. De Move Base is een Action Server die een x en y coördinaat kan ontvangen om hier vervolgens in te navigeren. De Move Base kijkt naar de kaart of het wel mogelijk is hierna toe te navigeren. Er zijn veel opties voor navigeren die geconfigureerd kunnen worden. Een van de belangrijkste is de costmap. Met de costmap wordt er de efficiëntste route berekend. Zo kun je bijvoorbeeld instellen dat dichtbij muren rijden meer 'kost'. De robot kijkt bij het navigeren naar de route die het minste kost. Hierdoor zal de Turtlebot verder van muren af rijden. In het figuur hieronder is de instelling "cost factor" in de costmap veranderd. Hoe lager de cost factor hoe meer het kost om dicht bij een muur te rijden en dus hoe verder de robot van muren rijdt. Er zijn twee verschillende soorten costmaps. De Global costmap die gebruikt wordt om met behulp van de kaart een route uit te stippelen in gebieden die de robot niet ziet. De local costmap zorgt voor autonome navigatie in gebieden die sensor van de robot ziet. Hierdoor kan de robot navigeren langs objecten die niet op de kaart staan.

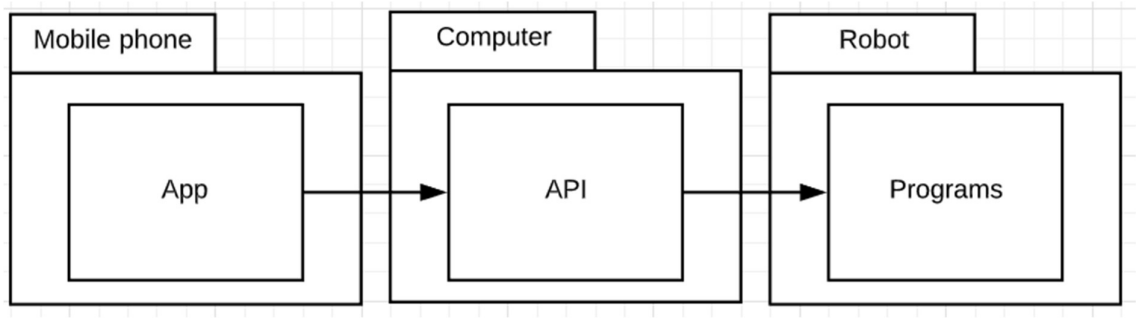


Figuur 11 Een lage cost factor rijdt verder van de muur weg. (Zoals links)

Om daadwerkelijk naar de beacons te rijden moet er een SimpleActionServer aangemaakt worden. De SimpleActionServer pakt de positie van een bluetooth beacon en verstuurt een goal naar de move base van de robot. 'goal' is een move_base_goal waar coördinaten meegestuurd kunnen worden. Wanneer de goal via de client naar de server gestuurd wordt zal de robot indien mogelijk autonoom naar de coördinaten toe rijden. Als dit is gelukt stuurt de action server een bericht terug naar de server.

6.4.2. Hoe kan de turtlebot3 het beste een commando van een mobiel ontvangen?

Rosnodejs[13] brengt alle ROS-functionaliteit naar NodeJs. Je hebt ook de mogelijkheid om een combinatie van Rosbridge en Roslibjs[11] te gebruiken om via javascript met ros te communiceren. Een voordeel van Rosnodejs tegenover Rosbridge en roslibjs is dat het aanzienlijk sneller is omdat het niet via de Rosbridge communiceert maar direct met ROS kan communiceren, ook heeft rosnodejs goede NodeJs ondersteuning wat we precies zoeken om de communicatie tussen robot en telefoon te doen. Er kan met de rosnodejs package Publishers, Subscribers, Nodes, Actionclients en etc. geprogrammeerd worden voor Javascript en NodeJs. Ook kent rosnodejs alle ROS en zelfgemaakte data types. Zo kunnen er makkelijk objecten gestuurd worden naar andere Nodes. Met NodeJs kan er gemakkelijk een api gemaakt worden die acties uitvoert nadat er een signaal is ontvangen. Zo kan er een Express NodeJs server op de Turtlebot3 of computer gezet worden die met de robot communiceert. Telefoons kunnen dan makkelijk communiceren met de turtlebot3 doormiddel van http API-calls. Er is ervoor gekozen om de server/ api op een externe computer te laten draaien zodat er later ook meerdere Turtlebots tegelijkertijd kunnen draaien en dezelfde kaart kunnen ontvangen.



Figuur 12 Deployment diagram

6.4.2.1. Hoe kan de turtlebot3 acties ondernemen op basis van het ontvangen commando?

Er is een Express NodeJs server opgezet dat Rosnodejs gebruikt. De Express server verbindt met de ROS-master op de robot om te communiceren met andere Nodes. In de Express server is er een POST route gemaakt die een X en Y verwacht. In die route wordt de X en Y omgezet in een geldige waypoint (MoveBaseGoal) en wordt er een bericht gepubliceerd op de '/waypoint' topic.

```
const express = require('express')
const app = express()
const port = 3000
app.use(express.json())


// Require rosnodejs itself
const rosnodejs = require('rosnodejs');
// Requires the std_msgs message package
const move_base_msgs = rosnodejs.require('move_base_msgs').msg;
//
let pub
rosnodejs.initNode('my_node')
.then((ros) => {
    pub = ros.advertise('/waypoints', move_base_msgs.MoveBaseGoal);
});

app.post('/waypoint', (req, res) => {
    if(req.body.x && req.body.y) {
        console.log(`Sending data! x:${req.body.x} && y:${req.body.y}`)
        const goal = new move_base_msgs.MoveBaseGoal();
        goal.target_pose.pose.position.x = req.body.x
        goal.target_pose.pose.position.y = req.body.y
        pub.publish(goal)
        res.send('Succes!')
    } else {
        res.send("Missing correct body")
    }
})

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

Figuur 13 Data wordt naar ros publiceert als er een post request op /waypoint binnenkomt

Als de robot, waypoint node en Express server aanstaan kan er een POST request van buitenaf gestuurd worden. Als er een JSON-body met een X en Y wordt meegestuurd zal de server een bericht publiceren en zal de python node een bericht binnen krijgen en vervolgens weer een bericht sturen naar de Move Base om naar de goede coördinaten te rijden.

6.4.3. Hoe kunnen gemaakte kaarten gedeeld worden zodat de Turtlebot vervangen kan worden?

Het POC moet zo opgezet worden dat een Turtlebot makkelijk vervangen kan worden. Het kan zijn dat een Turtlebot kapotgaat en dit moet een bedrijf makkelijk kunnen vervangen. Alle software kan gewoon op de nieuwe Turtlebot gezet worden. De Turtlebot die kapot is gegaan heeft wel de kaart gemaakt en is dus de enige die de kaart bezit. Het zou onnodig zijn om de nieuwe Turtlebot opnieuw rond te laten rijden om een kaart te maken. Daarom is het handiger om de backend die ook signalen ontvangt van bijvoorbeeld een mobiel niet op de Turtlebot te zetten maar echt op een server. Die server kan ook de kaart opslaan en naar de nieuwe Turtlebot sturen. Er is ook een requirement voor de POC om twee Turtlebots rond tegelijkertijd te laten rijden. Voor dit POC is ervoor gekozen deze requirement een won't te maken. Het staat dus niet in de planning om dit te implementeren. De opdrachtgever zou dit later wel willen dus moet het project zo opgezet worden zodat dit in de toekomst wel kan. Het zou bijvoorbeeld voor een groot warehouse handig zijn om meerdere Turtlebots rond te laten rijden om zo sneller bij een ongeval te kunnen zijn. De backend zal hiervoor ook niet op de robot moeten staan maar echt op een server/computer. Beide Turtlebots moeten natuurlijk wel over dezelfde kaart beschikken. Ook moet de server kijken wat de dichtstbijzijnde Turtlebot is om er naartoe te rijden. De begin positie van de Turtlebots zullen opgeslagen moeten worden. Dit heeft de Turtlebot al nodig om zijn positie in de kaart te bepalen. Voor dit POC zal de

kaart wel gestuurd worden vanaf een server maar zal er voor de rest geen aandacht aan meerdere Turtlebots worden besteed omdat dit buiten de complexiteit van deze opdracht valt.

6.4.4. Wat is de beste manier om de software die geschreven wordt te testen?

Omdat de Turtlebot met slam werkt, wat gebruikt maakt van eerder gemaakte statische kaart om de positie te bepalen en te navigeren is er als eerste instantie in de requirements gezegd dat de situatie in een distributiecentrum niet kan veranderen. In de experimenten die hiervoor zijn gedaan is te zien dat de robot redelijk goed nieuwe obstakels kan ontwijken als de obstakels maar tijdelijk of duidelijk waarneembaar zijn. Een dynamische situatie is wel lager in de prioriteit gezet omdat het meer een POC is en over het vinden en navigeren naar een bluetooth beacon gaat. Ook is het belangrijk dat muren zichtbaar zijn tijdens het maken van de kaart. De robot raakt heel makkelijk zijn positie kwijt als hij geen muur in zicht heeft. De laser kan maximaal 3,5 meter vooruitzien. De robot kan wel zijn positie goed bepalen als het geen muur ziet maar als dit langer dan 6 meter is het niet meer nauwkeurig. Ook is het belangrijk dat het een niet te grote lege ruimte is. Als de robot geen oriëntatie punt heeft weet de robot al snel niet welke positie de robot heeft. Met testen moet hier ook rekening mee gehouden worden.

Wat zijn de beste oplossingen voor virtueel testen?

Met Gazebo kan er een virtuele wereld ingeladen worden met een virtuele Turtlebot3.
Hiermee kunnen er test opstellingen gemaakt worden. De test cases kunnen hiermee met verschillende situaties getest worden. Ook kunnen de simulaties versneld afgespeeld worden wat testen sneller maakt. Met virtuele testen is er alleen geen goede manier om de hele flow te testen van mobiel naar robot of om beacons in de kaart te plaatsen.

Wat zijn de beste oplossingen voor fysiek testen?

Voor het fysiek testen kan er in het kantoor getest worden. Door de situatie met Corona kan er waarschijnlijk niet in een echte warehouse getest worden. Gelukkig is het kantoor groot en kunnen veel situaties hier al afgevangen mee worden. Het voordeel hiervan is dat we hiermee kunnen kijken wat de robot doet als er een persoon loopt of als er een stoel van plaats veranderd. Hiermee kan je de beperkingen van de robot vaststellen en hier eventueel wat aan doen.
Ook heeft ICT Group in Deventer meerdere afdelingen en kamers. Hierdoor kunnen er verschillende plattegronden en situaties getest worden.

6.4.5. Wat is de beste manier om de robot te laten navigeren zonder wifi?

Tijdens de realisatie van dit project kwam er nog een extra requirement bij vanuit de opdrachtgever. Tijdens de realisatie ben ik erachter gekomen dat de robot slecht functioneert als er geen wifiverbinding is. Dit komt omdat het eerst de bedoeling was alle programma's op de computer te draaien die ook de server was. Die computer stuurde door wat de robot moest doen en naartoe moest navigeren. Als de wifi verbinding wegviel kon de computer hierdoor niet tegen de robot zeggen dat hij om moest keren of moest stoppen. Ook kwam de laserdata van de robot niet binnen op de computer waardoor de kaart niet meer klopte. Een warehouse is groot en heeft op veel plekken geen wifi daarom heeft de opdrachtgever besloten om hier een extra eis van te maken. Hierdoor is hier ook onderzoek naar gedaan. Omdat de robot maar bestaat uit een Raspberry pi als computer heeft het niet veel rekenkracht. Daarom staan standaard belangrijke programma's zoals het maken van een kaart en navigeren op een externe 'Master' computer. De Raspberry pi zendt alleen alle data uit van de laser en motoren en kan wat data ontvangen voor het bewegen van de motoren. Alle Subscribers en Publishers werken via wifi. Als er geen wifi meer is kan de robot niets ontvangen of versturen met als gevolg dat alle data achterloopt en niet meer klopt. Ook als de

computer wifi verliest kan de computer niet meer praten maar Subscribers en Publishers die aanstaan op de computer. Dat waren de twee uitdagingen die opgelost moesten worden.

Als eerste moet alle gemaakt software op de robot komen te staan zodat de robot kan beslissen waar die naartoe rijdt zonder hulp van de computer. Er is een repo gemaakt van alle gemaakte software en is hierdoor gemakkelijk bij te houden en bij te werken op de computer en de robot. Hierdoor kan er ook makkelijker getest worden. Voor het maken van de kaart en navigatie hierin is gebruik gemaakt van al bestaande software van ROS. Dit stond nog niet op de robot. ROS is heeft groot softwarepakket wat de robot niet makkelijk kan downloaden en vaak niet kan gebruiken omdat de robot gebruik maakt van Rasbian Stretch en niet Ubuntu 16.04. Daarom is elk benodigd pakket apart gedownload en gebouwd.

De benodigde ROS-software:

- AMCL (Lokalisatie systeem)
- Map-server (Voor het openen en opslaan van kaarten)
- Move-base (Het aansturen van de robot)
- Navigation (Voor het rondrijden met hulp van AMCL en de move-base)
- SLAM (Voor het gebruiken van het Gmapping algoritme)

Om dit project daarna te kunnen bouwen moet dit ook via een bepaald command gedaan worden. Dit komt omdat de pi maar 1 gigabyte aan ram heeft en niet zo krachtig is. De turtlebot3 beschikt over een pi model 3b+ wat niet zomaar te upgraden valt omdat de pi model 4 niet werkt met Rasbian stretch en dus ook niet de ROS programma's of de Turtlebot programma. Een andere onboard computer zou ook niet lekker werken omdat we dan van Ubuntu Mate gebruik zouden moeten maken die de Turtlebot programma's niet ondersteund en ROS Kinetic ook niet geheel ondersteund. Gelukkig is het gelukt om de programma's op de pi model 3b+ werkend te krijgen omdat we externe programma's gebruiken hoeven we niet de debuggers erbij te bouwen die veel rekenkracht nodig hebben. Het gebruikte commando: 'catkin_make DCMAKE_BUILD_TYPE=Release -j1'. De '-j1' flag dit zorgt dat het project maar wordt gebuild met maar 1 core van de processor en hierdoor slaagde de build ook. Door de '`DCMAKE_BUILD_TYPE=Release`' naar Release te zetten krijgen we het 'out-of-the-box' programma dat ook het snelste werkt. Het maakt ook geen verschil omdat het al goed geteste software van ROS zelf is.

Alles draaide nu goed op de robot maar de robot werkte alsnog niet zonder wifi omdat alles wat draaide op de robot alsnog alleen via wifi communiceerde. Dit bleek een fout te zijn in de configuratie van de IP-adressen. De robot communiceerde met zijn externe IP-adres. Na dit in 127.0.0.1 te hebben veranderd kon de robot nog wel goed communiceren zonder wifi en met wifi. Dit is gevonden door op het tutorial van de netwerk setup te kijken van ROS[22]. Het nadeel van alles op de robot zetten is dat we hierdoor niet het Slam algoritme Cartographer kunnen gebruiken. Het algoritme is te complex en ondersteund geen Rasbian Stretch. Hierdoor is er toch van het Gmapping algoritme gebruik gemaakt omdat het niet veel kracht gebruikt van de computer. Na testen bleven de kaarten goed bruikbaar en zaten er geen fouten in.

6.4.6. Conclusie

In de deelvragen zijn we erachter gekomen dat de robot goed werkt als er aan bepaalde voorwaarden voldaan wordt. Denk bijvoorbeeld aan de plaatsing van de beacons of obstakels en aan dat de robot alleen LIDAR bezit. Om de opdracht zo haalbaar mogelijk te maken heb ik ervoor gekozen om mij eerst heel streng aan die voorwaarden te houden en het project hier om heen te maken. Wanneer het POC werkt volgens de voorwaarden kunnen we het project uitbreiden zodat de

robot in meer situaties werkt. De voorwaarden waaraan de situatie zich aan eerste instantie aan moet voldoen staan in “afbakening”(kopje 5).

Om de turtlebot3 te laten navigeren naar een bluetooth beacon moet er eerst een kaart gemaakt worden met behulp van LIDAR en een SLAM-algoritme. De kaart kan handmatig of autonoom gemaakt worden. Bij het handmatig maken van de kaart krijg je een completere kaart en kunnen fouten makkelijk voorkomen worden.

Er is ervoor gekozen om het Gmapping algoritme te gebruiken omdat het algoritme het beste werkt als het aanstaat op een robot. Cartographer werkt goed maar kan niet op de robot draaien waardoor we afhankelijk worden van wifi. Hector is te onbetrouwbaar als er iets verkeerd gaat.

Normaal draaien programma’s meestal op een externe computer omdat de robot niet veel rekenkracht heeft. Omdat de robot moet blijven werken zonder wifi zullen programma’s voor het navigeren en het maken van de kaart op de robot gezet worden.

In eerste instantie is het handig om de kaarten wel handmatig te maken om eerst de focus te leggen op het vinden van bluetooth beacons en het rijden hierna toe. Nadat alles werkend is zal het autonoom navigeren om een kaart te genereren de hoogste prioriteit krijgen.

Tijdens het maken van die kaart moeten er drie Nodes aanstaan die bluetooth beacons met de locatie opslaat. Alle Nodes staan op de robot omdat we dit willen bijhouden en opslaan zonder wifi. Node 1 stuurt alle bluetooth low energy signalen door. Node 2 stuurt de positie van de robot. Node 3 moet de gemiddelde signaalsterkte van bluetooth beacons bijhouden met de positie hiervan. De positie van de robot met het beste signaal wordt uiteindelijk opgeslagen. Zo kan je ervan uitgaan dat de robot dicht naar de beacon kan rijden. Omdat de robot er al een keer naartoe is gereden om de meting met het beste signaal te doen.

Er zal een externe computer met een server moeten zijn die doormiddel van ROS in verbinding staat met de robot. Op de server staat een NodeJs Express server die doormiddel van Rosnodejs berichten naar ROS en de robot kan sturen. Nadat de NodeJs server die op de robot staat een bericht krijgt van een mobiel kan de server door middel van Rosnodejs een bericht sturen naar een Node (deze staat op de robot) die de positie van de meegegeven bluetooth beacon met eerder gemaakte map kan sturen naar de Move Base Node(deze staat op de robot). De Turtlebot zal dan naar die coördinaten rijden door middel van de gemaakte kaart. Het is wel nodig om de start positie vast te leggen zodat de robot weet waar het staat. Als de start positie hetzelfde is als voor het maken van de kaart staat het automatisch goed ingesteld en hoeft er niets extra’s gedaan te worden. Anders moet er een signaal naar een Node gestuurd worden met de verwachte positie.

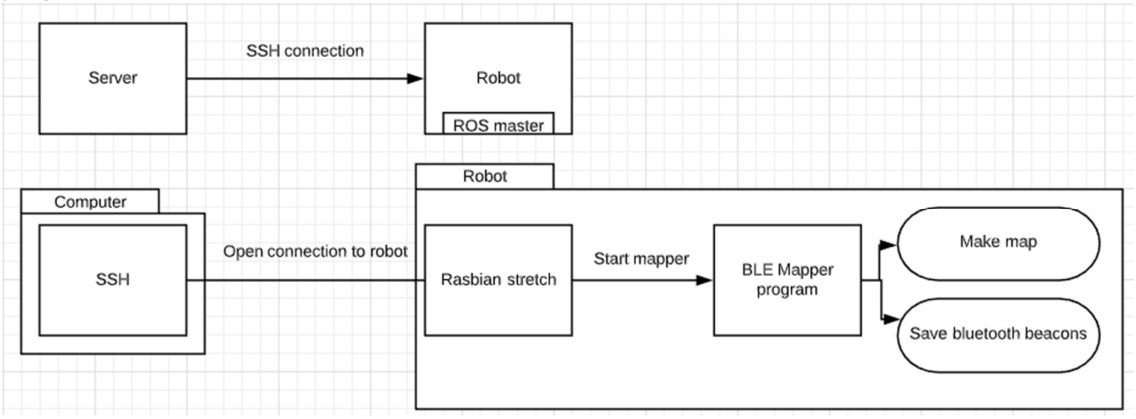
7. Realisatie

In het hoofdstuk realisatie bespreken we wat er uiteindelijk gerealiseerd is en hoe de realisatie werkt.

7.1. Algemene architectuur

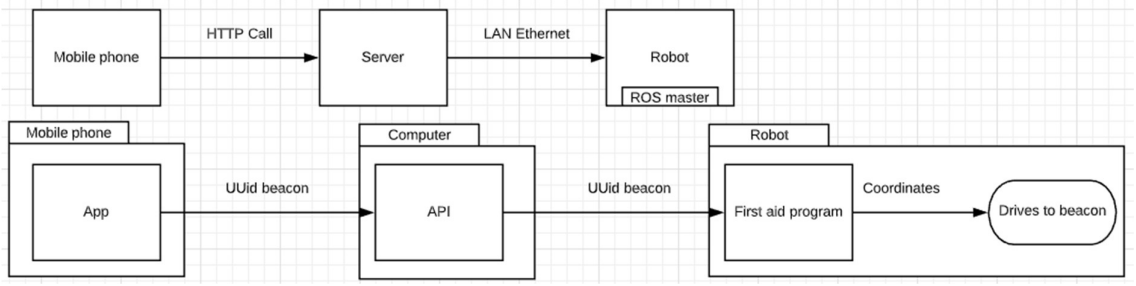
Om het uiteindelijke proces van het project te begrijpen wordt er nu de algemene architectuur uitgelegd. De hele architectuur bestaat uit een api, de robot en de applicatie. Het proces bestaat uit twee fasen waarvoor elk aparte programma’s zijn gemaakt. Fase 1 maakt de kaart en slaat de positie van bluetooth beacons op. Fase 2 kan rijden naar bluetooth beacons. In deze fase reageert de robot op een hulp verzoek van de app en rijdt vervolgens naar de dichtstbijzijnde bluetooth beacon.

Fase 1:
De computer maakt doormiddel van SSH een connectie met de robot. Via SSH moet de “Mapper” programma’s aangezet worden. Hierna wordt er een kaart gemaakt en bluetooth beacons met positie opgeslagen. Als de kaart compleet is moet die ook nog opgeslagen worden door een programma.



Figuur 14 Overzicht communicatie voor het maken van een kaart met bluetooth beacons

Fase 2:
Nu er een kaart is gemaakt kan hierin genavigeerd worden. Voor deel 2 worden ook andere programma’s aangezet dan deel 1. De computer die eerst is gebruikt om alle programma’s op de robot aan te zetten wordt nu ook als API gebruikt. De api kan een bericht van een app ontvangen doormiddel van http en doorsturen naar de robot doormiddel van ROS. Als de robot een beacon krijgt wordt er in een lijst gekeken met bluetooth beacons met bijbehorende posities. Hierna rijdt de robot naar de positie van de bluetooth beacon.



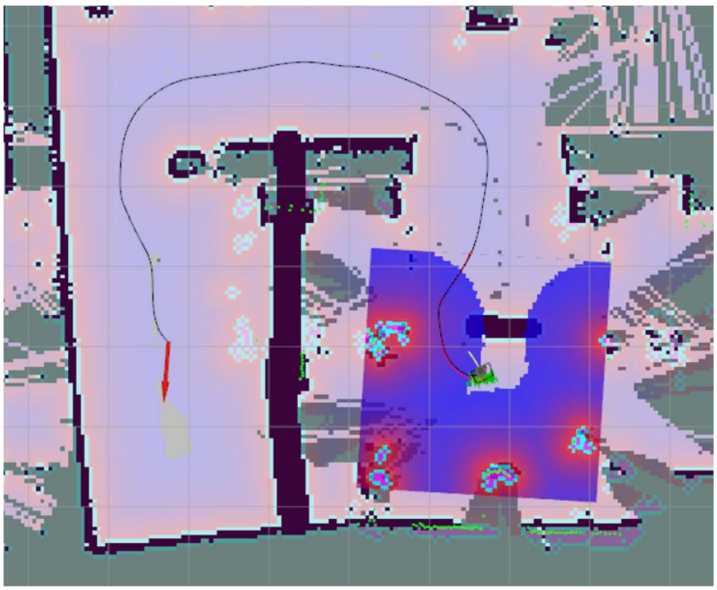
Figuur 15 Overzicht communicatie rijden naar een beacon

7.2. Het verbeteren van kaarten

Fouten in de kaart kunnen altijd nog verbeterd worden. Een kaart kan doormiddel van foto bewerk programma’s zoals GIMP verbeterd worden. Kaarten zijn .pgm (“portable grey map”) bestanden en bestaan maar uit drie kleuren (zwart, wit en grijs). De zwarte kleur staat voor een muur of ‘no go zone’. De witte kleur staat voor de vloer en navigeerbaard gebied. De grijze kleur is onbekend en nog niet in kaart gebracht. Het kan voorkomen dat een muur niet in de kaart is gezet omdat het van glas is. Dit kan je makkelijk met verbeteren door in de kaart een zwarte lijn als muur neer te zetten. Als dit te veel wordt gedaan kan het voorkomen dat de lokalisatie niet meer klopt omdat er een muur staat die er niet is. Bij kleine verbeteringen komt dit echter nooit voor. In een groot warehouse kan het voorkomen dat er plekken zijn waar de robot nooit naartoe of door mag navigeren. Hierdoor kan het zijn dat een groot deel van de kaart nog bewerkt kan worden wat lokalisatie problemen veroorzaakt. Daarom kunnen de lokalisatie problemen voorkomen worden door in te stellen dat er voor lokalisatie de originele kaart gebruikt wordt en voor het plannen van een route de bewerkte kaart gebruikt wordt. Dit kan ook handig zijn in een warehouse waar tijdens het rijden een stelling

helemaal leegstaat. De robot zet in de kaart dat de stelling leeg is en dat de robot er dus doorheen kan rijden omdat het zo in kaart gezet is. Nu is het handig om in de kaart een muur te plaatsen omdat de robot er rekening mee moet houden dat er wel producten in kunnen staan en. De robot zal hierna niet proberen door de stelling heen te rijden.

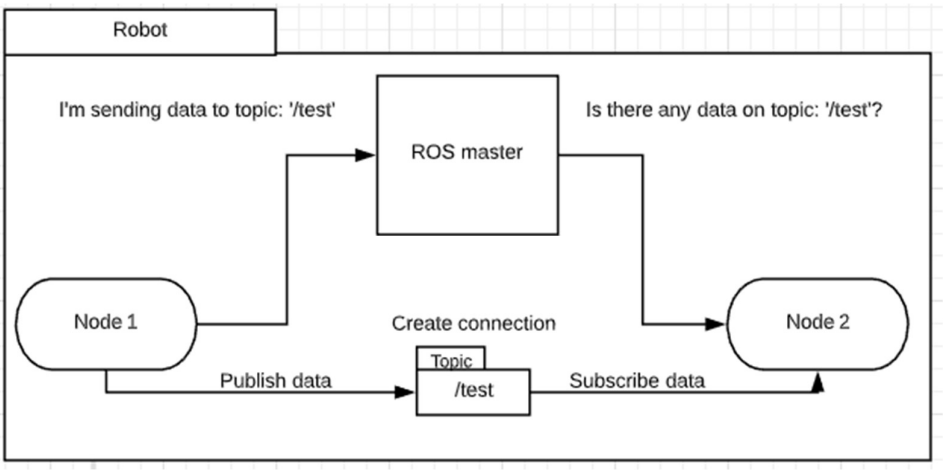
In Figuur 16 is een zwarte lijn in een lege kamer gezet. De route van de robot houdt rekening met de muur en de lokalisatie blijft kloppen omdat die de originele kaart gebruikt zonder muur. Ook is er een zwarte lijn recht voor de robot gezet. De robot ziet de muur niet maar beweegt er wel netjes omheen.



Figuur 16 Voorbeeld van een kaart waar extra muren zijn in bewerkt.

7.3. Communicatie

Er is van drie verschillende soorten communicatie gebruik gemaakt. Het is belangrijk om te weten dat de robot de 'ROS Master' is en verantwoordelijk is voor de connecties tussen ROS-programma's opzetten. Dit is gedaan omdat de robot ook moet werken zonder internet, dit kan alleen als de robot connecties kan blijven maken tussen interne ROS-programma's. Wanneer er wel wifi is kunnen externe apparaten ook communiceren met ROS en data ontvangen en data publiceren.



Figuur 17 Hoe maakt ROS-connecties tussen publishers en subscribers

Hieronder worden de drie soorten communicatie per soort uitgelegd.

App -> API

Bij een ongeval kan je doormiddel van een app een bericht sturen met de dichtstbijzijnde bluetooth beacon. Dit doet de app door middel van een http-call. De app stuurt een JSON bericht met het UUID van een bluetooth beacon naar het adres: 'IP_ADDRESS_HERE'/move.

API -> Robot

De computer waar de API op staat draait ook op ROS en staat in verbinding met de 'ROS-master' op de robot. Op de robot staat een programma aan wat een 'beacon_action' verwacht en hierna naar de beacon toe rijdt. De express server kan doormiddel van het Rosnodejs package die actie met een UUid van een bluetooth beacon vanaf de expres server sturen naar de robot met ROS. Dit wordt gedaan doormiddel van een ROS connectie via internet. Bij het instellen van de robot stel je de IP-adres van de 'ROS-master' in op alle ROS-machines. Hierdoor weet de server dat hij een action naar de robot moet sturen.

Het programma wat naar een beacon rijdt staat in verbinding met andere ROS-programma's zoals de move base en map server. Mocht het internet uitvallen tijdens het navigeren maakt het niet uit voor de robot omdat de communicatie van programma's die op de robot intern gaan en dus geen internet nodig hebben.

SSH -> Robot

Voor het maken van de kaart en het opslaan van de bluetooth beacons moeten programma's op de robot aangezet worden. Dit wordt gedaan doormiddel van een SSH-verbinding van bijvoorbeeld computer naar robot. Als de SSH-verbinding is gelukt kunnen alle programma's bestuurd worden voor het maken van de kaart. Als de robot autonoom de kaart aan het maken is maakt het niet uit als het internet uitvalt omdat alle programma's hiervoor op de robot draaien en niet op de server!

7.4. Robot

In dit hoofdstuk worden alle technische aspecten van de robot uitgelegd. Voor uitleg over de code kan gekeken worden naar de bijlage: 'Technisch document'.

7.4.1. Keuzeverantwoording

Bluepy [20]

Met Bluepy kunnen er bluetooth low energie apparaten gevonden worden. Om hiermee te coderen moeten de Bluetooth rechten wel goed staan. Als je programma's niet als root wil uitvoeren moet je bluetooth permissies aan de pi gebruiker geven. Hiermee kunnen de apparaten die de Turtlebot heeft gevonden doormiddel van ROS naar de server gestuurd worden. Er is voor Bluepy gekozen omdat het goed werkte met ROS. Er is ook getest met het bekendste python bluetooth package 'pybluez' maar die werkte niet goed omdat het een omgevingsvariabele aanpaste die ROS nodig had.

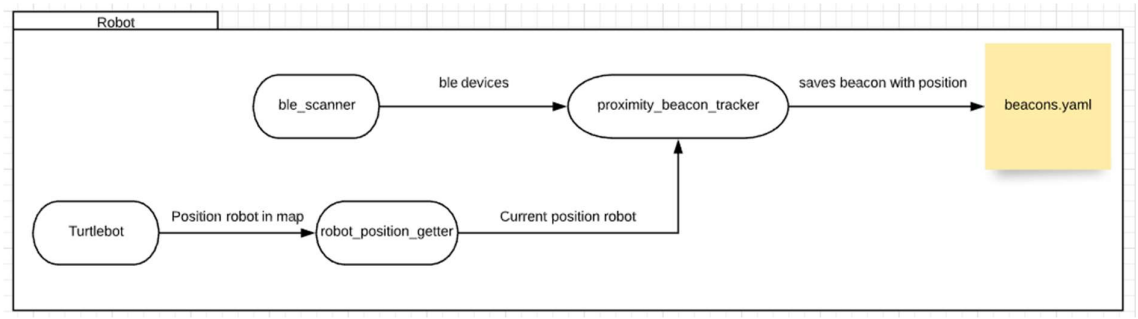
7.4.2. Architectuur

Er zijn verschillende programma's voor de robot geprogrammeerd. Het project heeft twee delen: Het maken van een kaart met bluetooth beacons en het rijden naar een bluetooth beacons. Beide delen hebben ook een andere architectuur met verschillende programma's daarom worden hieronder beide architecturen uitgelegd.

Het maken van een kaart met bluetooth beacons

Het maken van een kaart met bluetooth beacons bestaat uit twee groepen programma's wat tegelijkertijd draaien. Eén groep programma's maakt de kaart doormiddel van Gmapping. De ander slaat bluetooth beacons op met de positie hiervan.

Het opslaan van bluetooth beacons



Figuur 18 Bluetooth beacons opslaan met posities

Voor het opslaan van bluetooth beacons zijn drie verschillende ROS Nodes geschreven: “ble_scanner”, “robot_position_getter” en “proximity_beacon_tracker”.

“ble_scanner”: Er is een bluetooth low energie Node gemaakt voor de robot. De robot stuurt alle bluetooth low-energy via de ‘ble_devices’ topic door. Andere Nodes kunnen hier weer dingen mee doen. Er zijn custom messages aangemaakt om de beacons door te sturen. De topic stuurt ‘devices’ als message. ‘devices’ bestaat als een lijst van het ‘device’ message. De ‘device’ message bestaat uit een id, de RSSI en een lijst met data. Data is een ook custom message in de vorm van een tuple. De tuple bestaat uit een naam, een id en een value. Hiermee kan alle data van een low energy device doorgestuurd worden naar een ander apparaat.

De bluetooth apparaten worden opgehaald doormiddel van Bluepy. De gevonden apparaten worden omgezet in een Device message. Hierna kan het gepubliceerd en door andere Nodes gelezen worden. Om de twee seconden wordt er opnieuw gescanned en data gepubliceerd.

```
string rssi
string addr
ble_device_scanner/DeviceTuple[] data
```

Figuur 19 Device.msg Opbouw van een bluetooth energie device

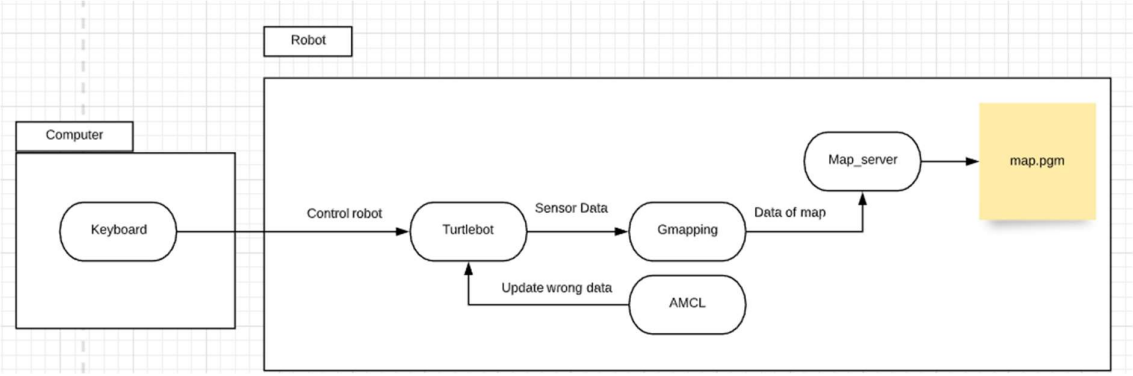
Een bluetooth low energy apparaat bestaat uit een adres een RSSI en data wat kan variëren denk bijvoorbeeld aan naam of merk.

“robot_position_getter” stuurt de positie van de robot door. Ondertussen wordt er AMCL uitgevoerd. AMCL zorgt ervoor dat de positie van de Turtlebot blijft kloppen met de positie die in de kaart staat.

“proximity_beacon_tracker” is de belangrijkste ROS Node. Deze Node Luistert naar ble_scanner en zoekt beacons. Elke keer als er een nieuwe lijst van bluetooth low energy apparaten wordt gepubliceerd wordt er een functie uitgevoerd. Deze functie houdt een lijst bij van alle gevonden bluetooth beacons met hun beste positie (door te luisteren naar robot_position_getter). **Wat zeer belangrijk is om te weten is dat we hierdoor zeker weten dat de positie die gevonden is met de meting ook later bereikbaar is omdat de robot momenteel ook op die positie staat.** Eerst wordt er gekeken of een apparaat wel een bluetooth beacon is. Elk bluetooth beacon heeft dezelfde begin van UUid als dit zo is het dus een bluetooth beacon van ons. Als dit niet zo is doe dan niets met het gevonden bluetooth apparaat. Als er een bluetooth beacon is gevonden en in de lijst zit voeg dan de meting toe aan een queue anders voeg hem toe aan de lijst als nieuwe beacon. In de queue kunnen maximaal 5 metingen. We gebruiken 5 metingen omdat de signaal sterkte van bluetooth soms

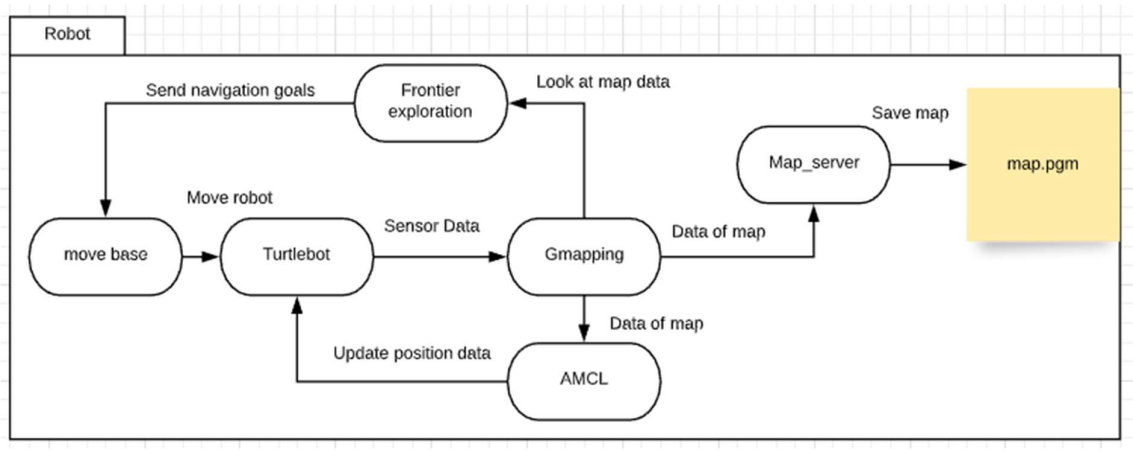
onstabiel is. De queue is first in first out dus als het maximum wordt overschreden wordt de oudste meting verwijderd om plaats te maken. Met de 5 metingen wordt er gekeken of er misschien een nieuwe beste meting is. Als dat zo is wordt de positie en het gemiddelde signaal hiervan opgeslagen. De beste meting is dus de best gevonden meting waar we zeker van weten dat de robot ertoe kan navigeren.

Het maken van een kaart



Figuur 20 Genereren van een kaart.

Terwijl er naar bluetooth beacons wordt gezocht wordt er ook een kaart gegenereerd (Kijk bovenstaand figuur). Gmapping zorgt hiervoor! De Turtlebot publiceert al zijn data en Gmapping maakt gebruik om hier een kaart van te maken. Als de robot even vast zit kan iets genaamd AMCL dit terug vertellen aan de Turtlebot. Bijvoorbeeld: De robot rijdt een opstapje op en dit duurt een tijdje. AMCL lokaliseert de robot door middel van de 'Monte Carlo localization approach' om dit makkelijk uit te leggen: AMCL kijkt naar de laser data die de robot terug geeft en kijkt waar die data het meest overeenkomt in de kaart. Als de data die Turtlebot doorgeeft niet meer klopt omdat de robot vast zien kan daardoor de robot denken dat er vooruit is bewogen omdat de wielen zijn bewogen. AMCL kan zien dat de robot niet ver vooruit is gegaan en kan data van de Turtlebot updaten zodat het weer klopt. Gmapping publiceert alle data van over de kaart op de topic '/map'. De map server kan hierna luisteren en dit in een herbruikbare kaart (map.pgm) opslaan. Om een kaart te genereren wordt er in bovenstaand figuur handmatig rondgereden door een extern programma.

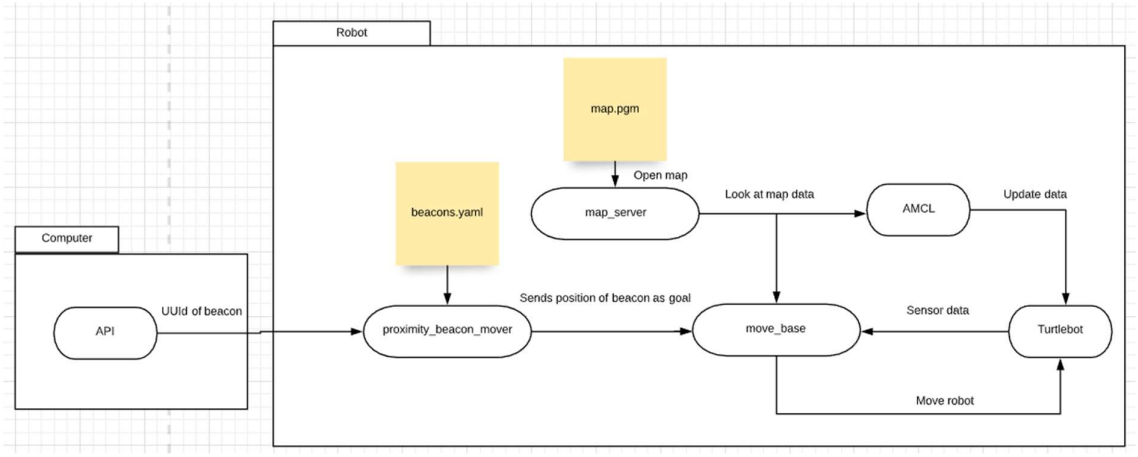


Figuur 21 Autonoom maken van een kaart.

Voor het autonoom maken van de kaart wordt Frontier Exploration gebruikt (zie bovenstaand figuur). Frontier Exploration kijkt naar de data van de kaart en ziet welke stukken nog niet ontdekt

zijn. Frontier exploration stuurt hierna een actie naar de move base van de robot om naar het onbekend gebied toe te rijden. Er zijn twee opties voor het maken van een kaart: Je kan een lege actie sturen naar Frontier exploration waarna frontier exploration alles in kaart probeert te brengen. Er is een programma gemaakt om dit signaal te sturen. De tweede optie is dat je een oppervlakte bestaande uit vier punten meestuurt. Frontier exploration zal nu alleen alles in dat oppervlakte ligt in kaart brengen. Je ziet in dit figuur ook dat we niet afhankelijk zijn van de computer en dus zonder wifi kunnen navigeren.

Het rijden naar een bluetooth beacon



Figuur 22 Het rijden naar een bluetooth beacon

Er is een launch bestand gemaakt om alles met betrekking tot het rijden naar een bluetooth beacon makkelijk op te starten. Om te navigeren in een kaart wordt de Move Base Node van Ros aangezet. Op de Turtlebot wordt ondertussen alle data gepubliceerd. Net zoals bij het maken van de kaart staat AMCL aan die data zoals positie van de robot kan verbeteren en updaten. De map server laadt de kaart in. Op basis van die kaart kan de move base een route maken. Move base is een Action Server die coördinaten kan ontvangen. Nadat coördinaten ontvangen zijn wordt er gekeken of er hier wel in de kaart genavigeerd kan worden. Als dat zo is begint de robot met navigeren, zo niet kan dit terug worden gegeven.

Er is een eigen SimpleActionServer aangemaakt genaamd proximity_beacon_mover. Proximity_beacon_mover is een action server die een Beacon actie kan ontvangen. Een beacon actie bestaat uit een adres, een feedback en een resultaat. Het adres kan meegestuurd worden door een client (Bij ons is dit de API-server). Met feedback en resultaat kan de client de status van de SimpleActionServer bijhouden. Wanneer er een actie wordt ontvangen wordt er gekeken naar het eerder gemaakt bestand met bluetooth beacons (“beacons.yaml”) of het adres aanwezig is. Als dat zo is wordt de beste positie gepakt en verstuurd als MoveBaseGoalAction naar de move base en indien mogelijk naartoe gereden. Als de beacon niet bestaat of niet navigeerbaar is wordt dit al resultaat teruggestuurd naar de client.

7.4.3. Implementatie

Welke userstories zijn getest en geïmplementeerd van de robot.

NR.	MOSCOW	USERSTORY	HOW TO TEST
4	Must	De robot moet bluetooth beacons kunnen lokaliseren doormiddel van bluetooth binnen de acceptabele bluetooth radius van de hardware. De bluetooth radius bij de bluetooth beacon is 15 meter indien het signaal onverstoord is.	
5	Must	Bij het instellen van de robot moet de robot rond kunnen rijden en een kaart kunnen maken van de omgeving. (Handmatig)	
5	Must	Bij het instellen van de robot moet de robot rond kunnen rijden en een kaart kunnen maken van de omgeving. (Autonoom)	
10	Should	Bij het instellen van de robot moet de robot rond kunnen rijden en bluetooth beacons ontdekken en hiervan de positie opslaan.	
11	Should	De robot na een signaal van de server te hebben ontvangen, navigeren en een kaart kunnen maken zonder wifi.	
12	Must	De robot moet de eerste verdieping van het kantoor van ICT Group in Deventer in één nacht ik kaart kunnen brengen.	Moet nog getest worden in het kantoor.
14	Should	De robot moet obstakels kunnen ontwijken terwijl hij een kaart aan het maken is.	
15	Should	De robot moet obstakels kunnen ontwijken terwijl hij navigeert naar een bluetooth beacon.	

Tabel 6 Implementatie van userstories

7.5. Server

In dit hoofdstuk worden alle technische aspecten van de server uitgelegd. Voor uitleg over de code kan gekeken worden naar de bijlage: 'Technisch document'.

7.5.1. Keuzeverantwoording

Express [21]

Express is een simpele NodeJs http-client. In een Express server kunnen http-calls ontvangen worden. Je hebt toegang tot de standaard GET, POST, PUT, DELETE-calls. Er is voor express gekozen omdat het zo simpel is. Binnen 5 minuten heb je een server die data kan ontvangen. Ook werkt het met NodeJs wat we nodig hebben om van Rosnodejs gebruik te maken. Omdat de API maar één route gaat hebben is er voor express gekozen. Ook is er voor express gekozen omdat het werkt met Rosnodejs.

Rosnodejs [13]

Rosnodejs brengt alle ROS-functionaliteit naar NodeJs. Er kan met de Rosnodejs package Publishers, Subscribers, Nodes etc. geprogrammeerd worden voor Javascript en NodeJs. Ook kent rosnodejs alle data types die ROS heeft zo kunnen er makkelijk objecten gestuurd naar andere Nodes. Met NodeJs kan er gemakkelijk een api gemaakt worden die acties uitvoert nadat er een signaal is ontvangen. Telefoons kunnen makkelijk communiceren met de turtlebot3 doormiddel van http calls.

7.5.2. Architectuur

De server bezit een API voor het sturen van een actie naar de Robot vanaf een mobiel. De server is een expres NodeJs server dat gebruikt maakt van "Rosnodejs" voor communicatie met ROS. Rosnodejs stuurt een actie met een bluetooth beacon naar de robot. Er is een custom actie hiervoor aangemaakt genaamd beaconAction. Deze action is vrij standaard alleen kan er een adres van een beacon meegestuurd worden. De SimpleActionServer die deze actie ontvangt kan doormiddel van 'result' en 'state' feedback terug aan de api geven bijvoorbeeld als de beacon niet bestaat of als de robot onderweg is.

```
# Goal
string address

---
# Result
string result

---
# Feedback
string state
```

Figuur 23 Inhoud van een beaconAction

De API bezit maar twee routes:

- De POST route '/move', deze route verwacht een JSON-body met een bluetooth beacon erin. '/move' zet de bluetooth beacon om in een beaconAction en stuurt die naar de robot
- De POST route '/back', deze route verwacht niets. Deze route stuurt een x en y van 0 naar de move_base, de robot zal dan terug rijden naar zijn begin positie.

7.5.3. Implementatie

Welke userstories zijn getest en geïmplementeerd van de server.

NR.	MOSCOW	USERSTORY	HOW TO TEST
6	Must	De server moet signalen van een mobiel kunnen ontvangen via wifi.	
7	Must	De server moet na een wifisignaal gekregen te hebben, de robot automatisch kunnen opstarten en navigeren naar de correcte bluetooth beacon doormiddel van een eerder gemaakt kaart met opgeslagen bluetooth beacons.	
12	Must	De server moet doormiddel van ROS, signalen naar de robot kunnen sturen.	

Tabel 7 Implementatie van userstories

7.6. App

In dit hoofdstuk worden alle technische aspecten van de app uitgelegd. Voor uitleg over de code en ontwerp kan gekeken worden naar de bijlage: 'Technisch document' en 'Functioneel ontwerp'.

7.6.1. Keuzeverantwoording

Vue [19]

Er wordt gebruik gemaakt van het web framework VueJS. Het maken van herbruikbare componenten is erg makkelijk. Vue heeft als voordeel dat alle logica vaak in 1 component staat: styling, script en templating. Er is voor Vue gekozen omdat ik hier de meeste ervaring mee heb. Door Vue, Cordova en Quasar te gebruiken kan alle functionaliteit van de app zo snel mogelijk gemaakt worden. Wat meer tijd overlaat voor nieuwe technieken in de robot. Hierdoor krijgen we een zo compleet mogelijk product.

Quasar [18]

Quasar is een framework voor Vue wat het makkelijker maakt om snel mooie applicaties te bouwen. Quasar biedt naast de Vue componenten een groot scala aan componenten die je makkelijk naar wens kunt aanpassen. Quasar komt met de mogelijkheid om vanuit één codebase een applicatie te bouwen voor meerdere platformen. Het heeft Cordova support waardoor je native functionaliteit van een device aan kunt spreken door middel van plug-ins.

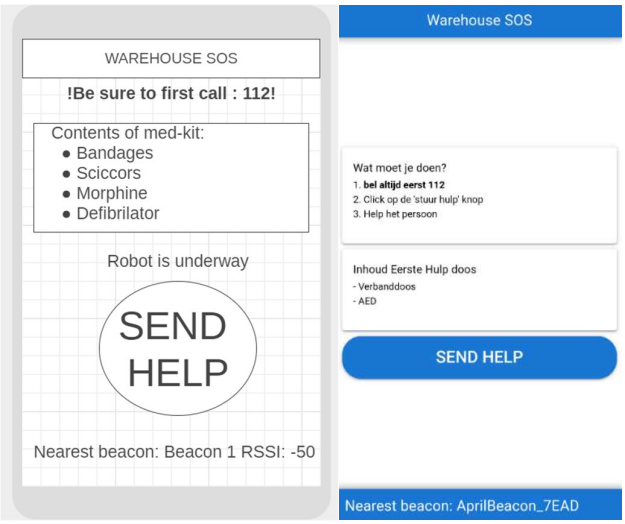
Cordova [15]

Er is gekozen om voor de mobiele applicatie gebruik te maken van Cordova. Cordova maakt het mogelijk om native functionaliteit aan te spreken. Cordova werkt goed samen met quasar. Cordova heeft een bluetooth low energy plugin wat het mogelijk maakt om bluetooth beacons te vinden. Dit is wat we nodig hebben om de dichtstbijzijnde beacon naar de robot te kunnen sturen.

Cordova BLE plugin [14]

Met deze plugin kunnen er bluetooth low energy apparaten via code gevonden worden. Met de gevonden apparaten kan de dichtstbijzijnde bluetooth beacon gevonden worden en naar de server gestuurd worden.

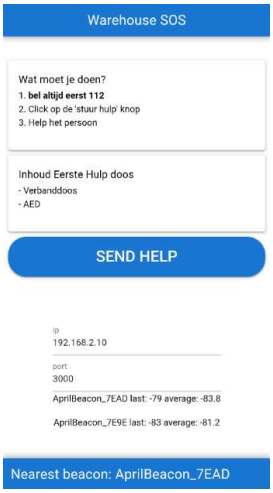
7.6.2. Ontwerp



Figuur 24 Mockup en uiteindelijk design van app.

De app bestaat maar uit één scherm om de app simpel en overzichtelijk te houden. Alle nodige informatie is beschikbaar op dit scherm. Het belangrijkste gedeelte van de app is het sturen van de robot daarom is dit met een grote knop met 'SEND HELP' of 'stuur hulp' aangeduid om op de knop te kunnen drukken moet de app wel een beacon hebben gevonden. Voor het testen van de applicatie is het handig om de dichtstbijzijnde beacon te tonen onderaan de app. Wanneer er op die knop is gedrukt kan er als uitbreiding nog altijd extra informatie over de robot getoond worden zoals hoe ver weg de robot is en waar de robot zich bevindt. Verder staat er nog wat informatie over de inhoud van de verband doos en welke acties er uit gevoerd moeten worden boven aan de app. Dit is gedaan om de gebruiker de juiste acties te laten ondernemen.

Wanneer er 5 keer op de onderste balk wordt gedrukt opent ook nog een debug menu waar je wat instellingen kan aanpassen en de lijst met gevonden beacons kan zien.



Figuur 25 App met debug menu open.

Er wordt gebruik gemaakt van quasar. Quasar maakt gebruik van de material design patterns van Google. Componenten van Quasar geven hierdoor een vertrouwd gevoel waardoor gebruikers snel weten wat bijvoorbeeld een knop is.

7.6.3. Architectuur

De primaire functie van de app is de dichtstbijzijnde bluetooth beacon doorsturen naar de server. De architectuur van de app is dus ook niet ingewikkeld. De beacons worden gevonden door een Cordova plugin: ”Cordova-plugin-ble-central”. Elke seconde sturen de beacons opnieuw hun signaalsterkte wat opgevangen kan worden door een mobiel. De lijst met laatste metingen worden bijgehouden. Wanneer de gebruiker op de ‘stuur hulp’ knop klikt, wordt er in de lijst met metingen gekeken naar de beste meting. Die meting wordt dan doormiddel van een http-call (Hiervoor is Axios gebruikt) naar de API gestuurd.

De App is een webapplicatie wat doormiddel van Quasar en Cordova is omgezet tot volwaardige app. Door Cordova kan er ook van native functionaliteit van het mobiel gebruik gemaakt worden. De app is getest op Android maar kan ook makkelijk werkend gemaakt worden op IOS door Cordova.

7.6.4. Implementatie

Welke userstories zijn getest en geïmplementeerd van de app.

NR.	MOSCOW	USERSTORY	HOW TO TEST
1	Must	De app moet bluetooth beacons via bluetooth kunnen vinden die binnen de acceptabele bluetooth radius vallen van de hardware. Bij de bluetooth beacon is dat 15 meter indien het signaal onverstoord is.	
2	Must	De app moet de dichtstbijzijnde bluetooth beacon kunnen tonen indien die er is.	
3	Must	De applicatie moet via wifi een bericht met de dichtstbijzijnde bluetooth beacon kunnen sturen naar de server.	
8	Should	Via de app moet het zichtbaar zijn of de robot onderweg is.	Je krijgt een signaal maar het wordt nog niet goed bijgehouden.
9	Should	De gevonden bluetooth beacons moeten zichtbaar zijn in de app.	

Tabel 8 Implementatie van userstories

8. Oplevering

Uiteindelijk is er het volgende opgeleverd:

Software:

- Robot met werkende software
- ROS-workspace (Code robot en server)
- App source code
- Android App APK

Documentatie:

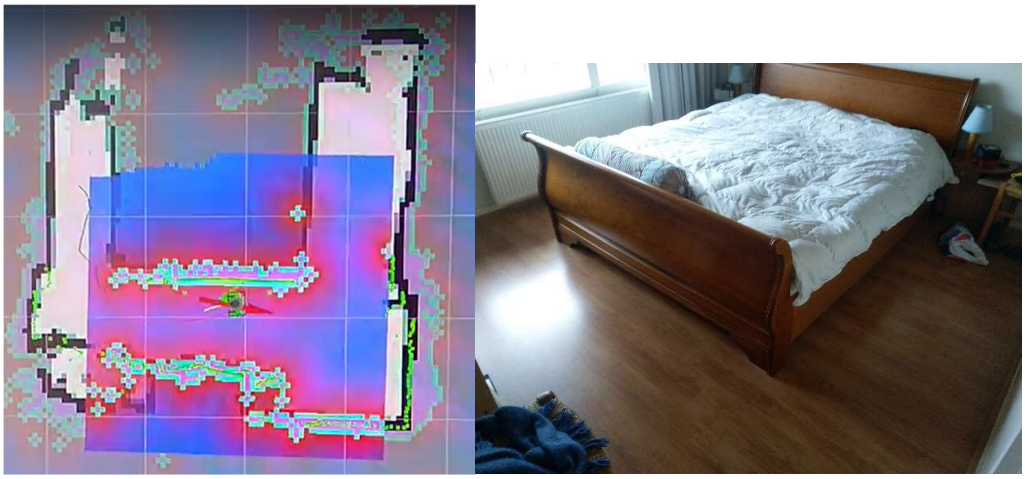
- Afstudeerverslag
- Technisch document
- Functioneel document
- Plan van aanpak
- Handleiding
- Development omgeving
- Onderzoek Robot
- Onderzoek App

9. Testen

Alle userstories bevatten een ‘how to test’. De userstories mochten pas naar ‘Done’ als de ‘how to test’ zonder problemen is afgerond. Als er toch iets niet bleek te werken, werden de taken in de userstory bijgewerkt met bijvoorbeeld de bug of vergeten functionaliteit. Bijna alle delen van het project zijn afhankelijk van elkaar. Hierdoor blijf je extra goed testen omdat je tijdens het testen van een nieuwe userstory ook opnieuw de functionaliteit van de oude userstories blijft testen. Hierdoor kwam het soms voor dat je nieuwe fouten of bugs in functionaliteit van oudere userstories tegenkomt die eigenlijk al klaar waren. Ik heb het gevoel dat deze manier van werken erg effectief is geweest omdat je niet aan het einde van het project erachter komt dat een vergeten stuk code niet meer werkt zoals verwacht.

Tijdens het project is er vooral fysiek getest. Dit is gedaan omdat alles gebruik maakt van fysieke hardware en dat niet mogelijk is te testen in een gesimuleerde omgeving. Ook zijn we er tijdens de fysieke testen erachter gekomen dat de uitkomsten anders zijn dan in een gesimuleerde test. Door Corona hebben we niet in een warehouse kunnen testen maar heeft de opdrachtgever wel als eis gezegd dat het project moet werken in het kantoor in Deventer. De meeste testen zijn bij mij thuis gedaan met verschillende kamers als verschillende test situaties. Alle testen in mijn huis zijn goed uitgevoerd maar een nadeel van thuis testen is dat de kamers niet groot waren dus weet je niet of

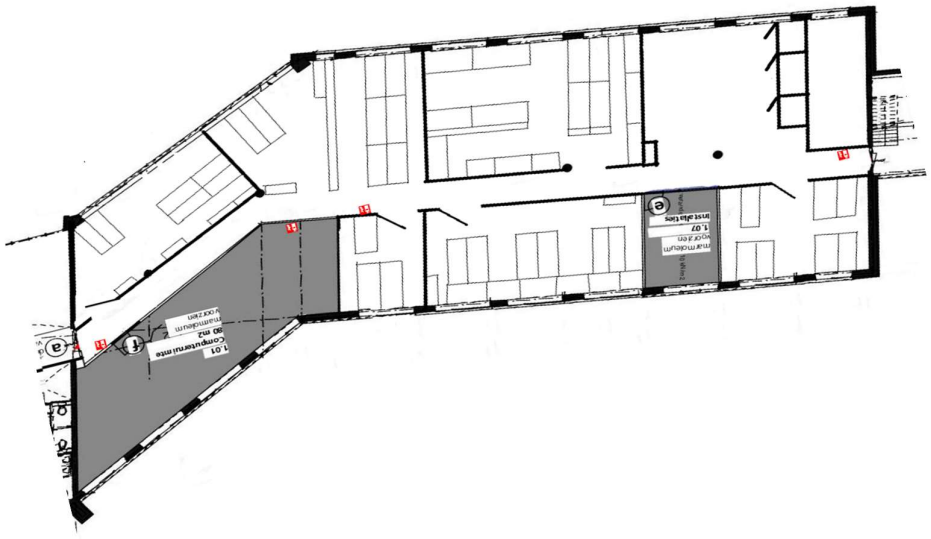
het blijft werken bij een grotere schaal. Daarom is er een grote test uitgevoerd in het kantoor in Deventer.



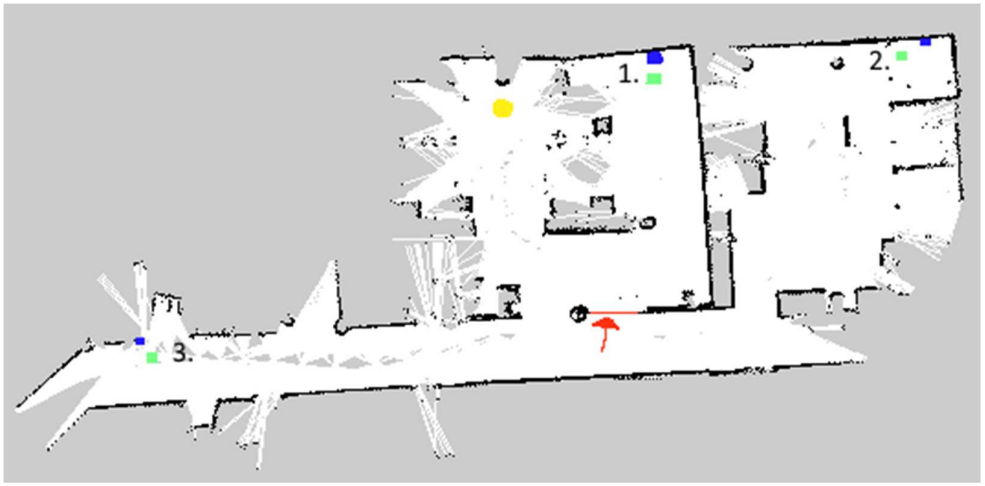
Figuur 26 Foto van slaapkamer en uiteindelijke kaart

Deze kaart is gemaakt in een slaapkamer. Deze kaart is autonoom gemaakt en duurde ongeveer 40 seconden om te maken. Aan het einde van de twee gangpaden staan bluetooth beacons. Met deze situaties kon goed de navigatie in kleine ruimtes en gangpaden van warehouses getest worden. De test verliep goed en de robot wist altijd snel te navigeren naar een bluetooth beacon. Na vaak thuis te testen is er ook in het kantoor getest om op een grotere schaal en realistischere situatie te testen In Figuur 26 duurde het ongeveer 40 seconden om van het ene gangpad naar het andere ganpad te rijden. Wat een route is van ongeveer 8 meter is. De robot moet door smalle gangpadden rijden en drie bochten maken waardoor de robot niet altijd op maximale snelheid kan rijden. Bij het autonoom rijden heeft de robot de beacons ongeveer 2 meter van hun echte positie af opgeslagen.

Voor het testen op het kantoor is als eerste stap handmatig rondgereden op de afdeling. De bluetooth beacons zijn in hoeken van de afdeling geplaatst zodat de robot een grote en complexe route moest produceren. De robot rijdt ongeveer met een snelheid van 0,25m per seconden vooruit. Deze snelheid was ook de verwachting van de opdrachtgever. Hierom is er ook geen strenge eis aan snelheid gesteld. Wel heb ik in kopje 10, een aanbeveling gedaan om het project bruikbaarder te maken. Er is wel een eis gesteld dat de robot de eerste verdieping van ICT Group in een avond in kaart kan brengen. De eerste verdieping in kaart brengen zonder werkkamers zoals in figuur 30 kossten ongeveer een 20 minuten. De onderste werkkamers zijn buiten beschouwing gelaten bij het testen omdat daar opstapjes zaten waar de robot niet overeen kon navigeren. Bij handmatig testen kan je de beacons zo dichtbij mogelijk positioneren door dicht naar de beacons te rijden en stil te staan. Bij het autonoom rijden positioneerd de robot de beacons ongeveer binnen 3 meter nauwkeurig. Soms is het dichterbij als de robot er autonoom toevallig dicht bij rijdt. De positionering van de beacons maken ook niet heel erg uit in een warehouse als de robot na een signaal te hebben ontvangen maar naar het juiste ganpad rijdt en wel dicht bij de beacon stopt.



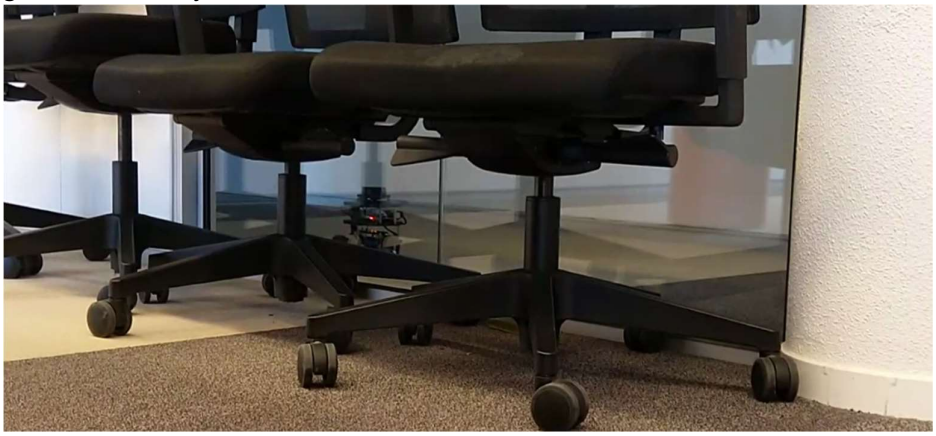
Figuur 27 Plattegrond eerste verdieping



Figuur 28 Handmatig gemaakte kaart van de eerste verdieping in Deventer.

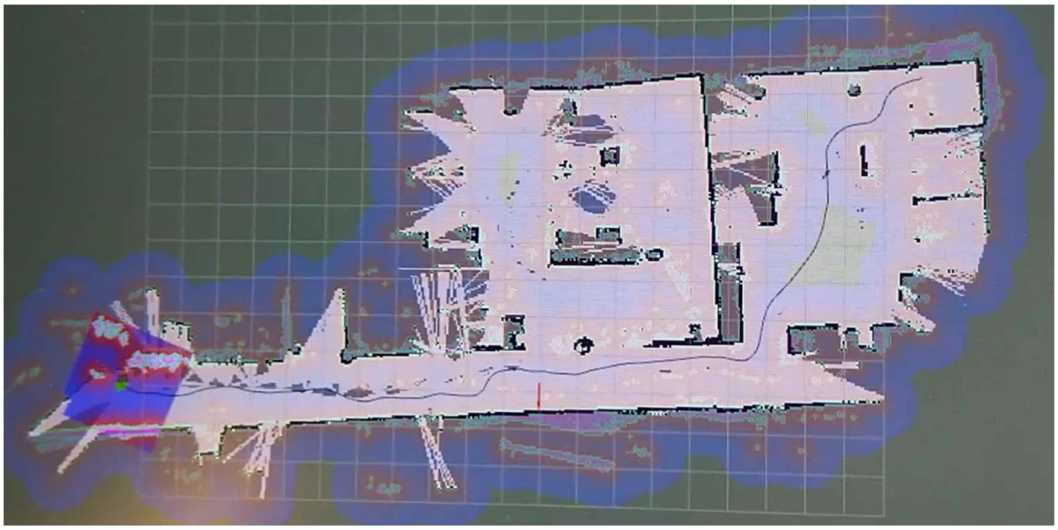
De hele afdeling is expres niet volledig in kaart gebracht omdat de robot goede routes moet blijven produceren als er muren niet in de kaart zijn gezet. De robot moet bijvoorbeeld niet proberen een sluiproute te maken door in onbekend gebied te rijden. De robot zou dan eerst in een doodlopend eind kunnen rijden en dan weer terug moeten rijden. Dit willen we natuurlijk niet voor een first aid robot. De testen in het kantoor liepen naar verwachting. De kaart is mooi geworden en alle muren zijn duidelijk te zien. De positie van de bluetooth beacon kwam ook in de buurt van de daadwerkelijke posities. De blauwe stippen in de kaart is waar de bluetooth beacons geplaatst zijn. De groene stippen zijn waar de robot naartoe reedt na een SOS signaal van de app. De gele stip is de positie van de robot aan het begin van het maken van de kaart en ook de begin positie van de robot voor het rijden naar een bluetooth beacon. De robot maakt alleen routes door het witte gebied heen hierdoor weten we dat de robot de snelst mogelijke route in de kaart pakt en geen onmogelijke routes maakt. De navigatie werkte altijd goed en de robot kwam nooit vast te zitten. De nadelen van de robot waren ook duidelijk tijdens alle testen. De rode streep in het bovenstaande figuur is een

glazen wand. De glazen wand heeft de robot niet in kaart gebracht omdat de laser door het glas gaat. Als de robot bijvoorbeeld van beacon 1 naar beacon 2 wil navigeren gaat hij proberen door de glazen wand te rijden.



Figuur 29 Glazenwand en bureaustoelen die niet in kaart zijn gezet.

Het andere nadeel is dat het een kantoor vol met bureaustoelen is. Er is al duidelijk gemaakt in de randvoorwaarden dat elk obstakel van de robot op laser hoogte moet staan. De wielen van bureaustoelen doen dat niet. Hierdoor rijdt de robot tegen de stoelen aan als het in de weg komt. Dit was de verwachte uitkomst en is allemaal makkelijk te voorkomen. Gelukkig kunnen fouten in de kaart altijd nog verbeterd worden door foto bewerk programma's zoals uitgelegd in kopje 7.2 'het verbeteren van een kaart'. De fout in onderstaande kaart is bijvoorbeeld is dat er geen muur is geplaats op de kaart omdat het een glazen wand is. Deze fout kan verbeterd worden door in GIMP een zwarte lijn te trekken als muur.



Figuur 30 Voorbeeld route van beacon 3 naar beacon 2

In bovenstaand figuur duurde het ongeveer 150 seconden om een afstand van ongeveer 30 meter te navigeren. De robot stopte hierna ook precies naast de beacon omdat hier handmatig naast is gereden. Er is na het handmatig testen ook een demo gegeven aan de opdrachtgever waar deze resultaten zijn besproken. De week na dat deze scriptie is ingeleverd gaat er autonoom in het kantoor getest worden.

10. Aanbevelingen

Het resultaat van het project is een proof of concept en is niet compleet gereed voor 'echt' gebruik. Dit komt onder andere door de robot die is gebruikt en de beperkingen hiervan. De eerste aanbeveling is dus ook om een grotere robot te gebruiken. Een grote robot is sterker en kan beter en sneller navigeren. De turtlebot3 navigeert niet snel en kan geen opstapjes op, ook dit is geen probleem voor een krachtigere robot met grote wielen.

Bijna alles wordt nu op basis van de LIDAR-sensor en SLAM gedaan. LIDAR en SLAM zijn goede indoor technologieën maar een nadeel ervan is dat het alleen maar metingen van punten zijn. De LIDAR-sensor zit ook op een bepaalde hoogte en kan niet naar boven of naar onder kijken. Hierdoor ziet het geen opstapjes of eventuele trappen. Dit is ook bij de testen als nadeel naar boven gekomen.

Een simpele verbetering is om van een 3d LIDAR sensor gebruik te maken. Een 3d LIDAR sensor ziet zowel de vloer als het plafond. Om dit te implementeren moet alleen de sensor vervangen worden. Als de data van de laser goed wordt meegegeven dan neemt Gmapping dit ook goed op in de kaart. Dit zou de simpelste oplossing zijn om de robot bruikbaarder te maken in een echte situatie. Een distributiecentrum heeft namelijk vaak kabels of planken liggen waar de robot niet over moet rijden.

Een mogelijk vervolg van deze opdracht en verbetering van de robot zie ik gebeuren met Machine learning. Als de robot een RGBD Camera bezit kan de robot een echt model van een pand maken. (Voor een idee kijk video: 'Online RGBD SLAM with Kinect'[30]). Dit kan al gedaan worden met een goedkope Kinect camera. Als dit is gedaan kan er gebruik gemaakt worden door RGB-D positionering algoritmes (Zie onderzoek voor betere uitleg: 'RGB-D Mapping: Using Kinect-Style Depth Camera's for Dense 3d Modeling of Indoor Environments'[31])

Tijdens het navigeren kan er gekeken worden naar de 3d camera om te kijken of het overeenkomt met het model. De robot kan op basis hiervan dus beter zijn positie nachecken. Ook kan er met de camera machine learning worden uitgevoerd. Denk dan bijvoorbeeld aan het ontdekken van bewegende machines of personen. Zo kan de robot een liggend persoon spotten en hier naartoe rijden. Ook kan er met de RGBD-camera gekeken worden of er opstapjes bevinden of dat de vloer opeens ophoudt door een trap.

Als laatste kan er gekeken om de gebruikers ervaring beter te maken. Nu moeten veel programma's via SSH aangezet worden zoals het beginnen met het maken van de kaart. Dit kan de mobiele app misschien ook doen. Denk bijvoorbeeld aan het maken en opslaan van de kaart terwijl je de kaart en robot live op je mobiel ziet.

Het handige van ROS is dat alle programma's blijven werken. Alleen de configuratie van de robot moet veranderd worden.

11. Bronnen

[1] Robotis. (Z.d.), TurtleBot3 Specifications. Geraadpleegd op 17/02/2020 van http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/

[2] Intel. (Z.d.), Intel joule 570x Developer kit. Geraadpleegd op 17/02/2020 van https://ark.intel.com/content/www/us/en/ark/products/96414/intel-joule-570x-developer-kit.html

[3] Raspberrypi. (Z.d.), Raspberry Pi 3 Model B+ Geraadpleegd op 17/02/2020 van https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[4] Clearpath (z.d.) Intro to ros. Geraadpleegd op 18/02/2020 van https://www.clearpathrobotics.com/assets/guides/ros/Intro%20to%20the%20Robot%20Operating%20System.html

[5] ROS. (Z.d.). *Rviz - ROS Wiki*. Geraadpleegd op 19 februari 2020, van http://wiki.ros.org/rviz

[6] Robotis. (z.d.). *Turtlebot3 - SLAM*. Geraadpleegd op 19 februari 2020, van http://emanual.robotis.com/docs/en/platform/turtlebot3/slam/

[7] Wikipedia-bijdragers. (2018, 17 juli). *Lidar*. Geraadpleegd op 19 februari 2020, van https://nl.wikipedia.org/wiki/Lidar

[8] Wikipedia-bijdragers. (2019, 7 April). *Simultaneous localization and mapping*. Geraadpleegd op 19 februari 2020, van https://nl.wikipedia.org/wiki/Simultaneous_localization_and_mapping

[9] Filatov Anton, Filatov Artyom, Krinkin KIrill, Chen Baian, Molodan Diana(2017, 8 Augustus). 2D SLAM Quality Evaluation Methods, Geraadpleegd Op 19/02/2020, van https://arxiv.org/pdf/1708.02354.pdf

[10] ROS. (z.d.-a). *Rosbridge_suite - ROS Wiki*. Geraadpleegd op 19 februari 2020, van http://wiki.ros.org/rosbridge_suite

[11] The Construct. (2019, 4 december). *Developing Web Interfaces For ROS Robots - Ep 1: Introduction to ROSBridge Server*. Geraadpleegd op 19 februari 2020, van https://www.youtube.com/watch?v=fJDwgOJPRJE

[12] Cognitive Robotics at ENSTA :: Indoor Navigation. (z.d.). Geraadpleegd op 27 februari 2020, van http://cogrob.ensta-paris.fr/loopclosure.html

[13] ROS. (z.d.-b). *Rosnodejs - ROS Wiki*. Geraadpleegd op 3 maart 2020, van http://wiki.ros.org/rosnodejs

[14] Evothings/cordova-ble. (z.d.). Geraadpleegd op 20 februari 2020, van https://github.com/evothings/cordova-ble

[15] Cordova. (z.d.). *Apache Cordova*. Geraadpleegd op 26 februari 2020, van https://cordova.apache.org/

[16] Bluetooth. (z.d.). *Rssi [Bluetooth® LE Wiki]*. Geraadpleegd op 26 februari 2020, van https://bluetoothle.wiki/rssi

[17] beaconstac. (z.d.). *What is a Bluetooth beacon? How do BLE beacons work?* Geraadpleegd op 26 februari 2020, van https://www.beaconstac.com/what-is-a-bluetooth-beacon

[18] Quasar. (z.d.). *Quasar Framework - Build high-performance VueJS user interfaces in record time*. Geraadpleegd op 23 maart 2020, van https://quasar.dev/

[19] VueJs. (z.d.). *Vue.js*. Geraadpleegd op 22 maart 2020, van https://vuejs.org/

[20] I. (z.d.). *IanHarvey/bluepy*. Geraadpleegd op 2 april 2020, van https://github.com/IanHarvey/bluepy

[21] *Express - Node.js web application framework*. (z.d.). Geraadpleegd op 30 februari 2020, van https://expressjs.com/

[22] ROS. (z.d.-b). *ROS/NetworkSetup - ROS Wiki*. Geraadpleegd op 25 april 2020, van http://wiki.ros.org/ROS/NetworkSetup

[23] Wikipedia-bijdragers. (2020, 1 juni). Kanban. Geraadpleegd op 8 maart 2020, van https://nl.wikipedia.org/wiki/Kanban

[24] Community research - ICT research methods. (z.d.). Geraadpleegd op 20 februari 2020, van http://ictresearchmethods.nl/Community_research

[25] *Literature study - ICT research methods*. (z.d.). Geraadpleegd op 20 februari 2020, van http://ictresearchmethods.nl/Literature_study

[26] *Ionic - Cross-Platform Mobile App Development*. (z.d.). Geraadpleegd op 8 juni 2020, van https://ionicframework.com/

[27] *React Native · A framework for building native apps using React*. (z.d.). Geraadpleegd op 8 juni 2020, van https://reactnative.dev/

[28] *Rosbag/Tutorials/Recording and playing back data - ROS Wiki*. (z.d.). Geraadpleegd op 22 februari 2020, van http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data

[29] Gazebo. (z.d.). Geraadpleegd van http://gazebosim.org/

[30] introlab. (2013, 31 januari). *Online RGBD SLAM with a Kinect*. Geraadpleegd op 20 mei 2020, van https://www.youtube.com/watch?v=AMLwjo80WzI

[31] Henry, P. (2012, 19 april). *RGB-D Mapping: Using Kinect-Style Depth Cameras for Dense 3D Modeling of Indoor Environments*. Geraadpleegd op 20 juni 2020, van https://www.researchgate.net/publication/254098812_RGB-D_Mapping_Using_Kinect-Style_Depth_Cameras_for_Dense_3D_Modeling_of_Indoor_Environments

[32] *Amcl - ROS Wiki*. (z.d.). Geraadpleegd op 3 maart 2020, van http://wiki.ros.org/amcl

[33] Mihai, N. F. (z.d.). *Improvement of Synchronization Algorithms in Home Position for Robots with 5 Degrees of Mobility*. Geraadpleegd van https://www.researchgate.net/figure/This-figure-from-the-ROS-wiki-3-shows-how-an-Action-Client-and-Action-Server_fig4_265084275