NTNU
Kunnskap for en bedre verden

Autonom-robot med LIDAR-sensor

Sondre Martin Kvellestad Jensen

Master i kybernetikk og robotikk
Innlevert: juni 2018
Hovedveileder: Tor Engebret Onshus, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Sammendrag

Målet med denne masteroppgaven var å ferdigstille en autonom robot som kan brukes i NTNU's "LEGO-robot" prosjekt. Denne roboten skal være i stand til å navigere i et område og sende informasjon om hvilke hindringer som finnes i dette området til en server. Denne serveren bruker informasjonen fra roboten til å lage et kart over området. Roboten skal også kunne samarbeide med de andre robotene i "LEGO-prosjektet" for å lage et 2D kart over omgivelsene.

Under prosjektoppgava TTK4551 høsten 2017, ble alle delene av roboten sett sammen. Det ble lagt mye arbeid i å lage et kretskort for å koble komponentene sammen. Dette kretskortet fungerte ikke og et nytt ble derfor laget under denne masteroppgava. Det nye kretskortet tok utgangspunkt i et design fra en tidligere prosjektoppgave, (Nilsen 2017). Det ble gjort endringer til kretskortet slik at både kompasset og LIDAR-sensoren enkelt kan kobles til roboten.

Programvaren til roboten, som det ble tatt utgangspunkt i, var utviklet under en tidligere masteroppgave (Amsen 2017). Denne koden er så blitt endret slik at den fungerer med de nye portene som blir brukt på det nye kretskortet. Det ble også laget nye drivere til de komponentene som var ulike dem fra Andersen og Rødseth sin master. Dette innebærer blant annet motorene og servoen til IR-tårnet.

Gjennom masteren ble det brukt mye tid på feilsøking av roboten. Det ble avdekket mange feil på roboten som alle er blitt rettet. Disse feilene er blant annet; skade på kompasset, spenningsfall over IR-sensorene og feilkobling på det nye kretskortet. Etter at feilene ble funnet og rettet, fungerte roboten som ønsket.

Da roboten var operativ og i stand til å utføre kartlegging ved bruk av IR-sensorene, ble den ene IR-sensoren byttet ut med en LIDAR-sensor. Roboten ble omprogrammert slik at den kunne motta data fra Lidar-sensoren. I prosjektoppgava TTK4551 (2017), ble en «LIDAR lite v3» - sensor testet i forhold til å utføre målinger over lange distanser. For at Lidar sensoren skulle fungere, ble det skrevet en Arduino IDE kode der funksjoner fra et eksisterende bibliotek fra Sparkfun ble brukt til å styre sensoren (TTK44551, 2017). Dette biblioteket var ikke kompatibelt med resten av koden til roboten og det ble derfor skrevet en C-kode som tok utgangspunkt i dette biblioteket.

Det ble laget et feste til LIDAR-sensoren slik at den enkelt kan kobles på og av roboten. Funksjonaliteten til sensoren ble implementert i koden til roboten slik at den tok i bruk LIDAR-sensoren istedenfor den fremste IR-sensoren. Det ble gjort endringer i serveren som styrte robotene, slik at serveren kunne behandle data fra LIDAR-sensoren. Endringene i serveren ble gjort slik at roboter som ikke bruker LIDAR-sensor, fortsatt kan bruke serveren som normalt.

For å få roboten til å fungere på samme server som blir brukt av de fleste robotene i "LEGO-robot" prosjektet, ble robotens navigering skrevet om fra polarkoordinater til kartesiske koordinater. Roboten ble testet sammen med de andre robotene. På bakgrunn av dette kunne en konkludere rundt robotens prestasjon både med og uten LIDAR-sensor.

Summary

The aim of this thesis was to finalize an autonomous robot that can be used in NTNU's ”LEGO robot” project. This robot should be able to navigate in an area and send information about the obstacles that exist in this area to a server. This server uses the information from the robot to create a map of the area. The robot should also be able to collaborate with the other robots in the ”LEGO project” to create a 2D map of the surroundings.

During the project TTK4551 in the autumn of 2017, all parts of the robot were assembled. Much work was done in making a circuit board to connect the components together. This circuit board did not work and a new one was therefore made under this master's thesis. The new circuit board was based on a design from a previous project, (Nilsen 2017). Changes were made to the PCB so that both the compass and the LIDAR sensor can easily be connected to the robot.

Starting point for the software of the robot, was developed under an earlier master thesis (Amsen 2017). This code has then been modified to work with the new ports used on the new PCB. New drivers were also created for the components that were different from Andersen and Rødseth's master. This includes the engines and the servo of the IR tower.

Throughout the master, a lot of time was spent on troubleshooting the robot. There were revealed many errors on the robot, all of which have been corrected. These errors include; damage to the compass, voltage drop over the IR sensors, and faulty connection on the new circuit board. After the errors were found and corrected, the robot worked as desired.

When the robot was operational and able to perform mapping using IR sensors, one IR sensor was replaced with a LIDAR sensor. The robot was then reprogrammed to receive data from the Lidar sensor. In the project TTK4551 (2017), a ”LIDAR lite v3” sensor was tested in relation to measuring long distances. In order for the LIDAR sensor to work, an Arduino IDE code was written where functions from an existing Sparkfun library were used to control the sensor (TTK44551, 2017). This library was incompatible with the rest of the code of the robot, and it was therefore through this master, written a C-code that was based on this library.

It was made a mount for the LIDAR sensor such that it could easily be connected and disconected from the robot. The functionality of the sensor was implemented in the code of the robot such that it used the LIDAR sensor instead of one IR sensor. Changes were made to the server that controlled the robots, allowing the server to process data from the LIDAR sensor. The changes in the server were made so that robots that don't use a LIDAR sensor still can use the server as normal.

In order to make the robot work on the same server used by most robots in the ”LEGO robot” project, the robot's navigation was written from Polar coordinates to Cartesian coordinates. The robot was tested together with the other robots. Based on this, one could conclude around the robot's performance with and without the LIDAR sensor.

Konklusjon

Gjennom dette prosjektet er det blitt utviklet og ferdigstilt en ny robot basert på Arduino-roboten som Andersen og Rødseth laget under sin masteroppgave, våren 2016. Dette er også en videreutvikling av prosjektet som ble startet under prosjektoppgaven høsten 2017, i faget TTK4551. Ved slutten av denne masteren er roboten operativ og fungerer på linje med de andre robotene i LEGO-robotprosjektet.

Roboten ble bygd med tanke på å rette problemer som eksisterte på Andersen og Rødseth sin robot. Roboten har nå ikke problem med at hjulene spinner, slik den gamle Arduino-roboten deres hadde. I tillegg er kommunikasjonsproblemene med Bluetooth dongelen fra den gamle roboten rettet. Roboten får kontakt med serveren og detter ikke ut.

Ved slutten av prosjektet 2017, var det mange problem med roboten. Problemene var hovedsakelig knytt til kretskortet som kobler alle komponentene sammen på roboten. Alle problemene fra prosjektet i 2017 er nå blitt rettet. Rettelsene bestod i å lage et nytt kretskort og programmere de driverne som manglet. Etter alle feilene på roboten ble funnet og rettet, klarte roboten å få kontakt med serveren og kartlegge omgivelsene og hindringer den møtte på.

Ettersom IR-sensorene har relativt kort rekkevidde, ca 40 cm, ble den ene IR-sensoren byttet ut med en LIDAR-sensor. En LIDAR- sensor har en rekkevidde på opptil 40 meter. Dette ble vellykket implementert ved at roboten nå klarte å ta i bruk målingene gjennomført av LIDAR-sensoren og sende de videre til serveren. Serveren ble endret slik at den kan ta imot og behandle avstander som LIDAR-sensoren opererer innenfor. Disse endringene er gjort på en slik måte at andre roboter vil fungere som vanlig på samme server. Kartet som blir laget ut fra dataen til LIDAR-sensoren ligner på området som ble scannet.

Under prosjektet ble roboten testet med LIDAR-sensor i forhold til IR-sensor. Resultatet var at kartet IR-sensorene klarte å lage var mer presist enn det kartet LIDAR-sensoren lagde. Dette skyltes at LIDAR-sensoren, som er en mer presis sensor, er mer følsom for unøyaktigheter hos roboten, enn det IR-sensorene er. Disse unøyaktighetene oppstår mest når roboten roterer, noe som fører til at LIDAR-sensoren detekterer punkter som blir kartlagt feil. Dette er mulig å ordne enten ved bruk av mer nøyaktige måleutstyr på roboten, eller ved å endre hvordan sensorene oppfører seg når roboten roterer.

Kartlegging ved bruk av LIDAR-sensoren skjer mye raskere fordi den kan kartlegge et større område i gangen enn IR-sensorer kan. Dette gjelder på små baner. Dette er fordi navigeringa serveren gir roboten ikke er optimalisert med tanke på rekkevidden til LIDAR-sensoren. Roboten bruker dermed unødvendig mye tid på å kjøre igjennom områder den allerede har besøkt.

For å gjøre det mulig å kjøre roboten på samme server som de fleste robotene i LEGO-robotprosjektet bruker, ble navigeringa til roboten skrevet om fra polarkoordinater til kartesiske koordinater. Det kunne da bli gjennomført en felles test, hvor alle de operative robotene kjørte sammen i en testbane. Under denne testen ble det også avdekket et problem med LIDAR-sensoren. Ettersom rekkevidde på LIDAR-sensoren er mye lengre enn rekkevidde på IR-sensorer, var det mulig for sensoren å detektere andre roboter i banen som et hinder. Dette førte til feilmerking noe som gjør at kartet blir feil.

Kartlegging ved bruk av LIDAR-sensor fungere, men fungerer ikke like bra som først antatt. LIDAR-sensoren er et mye bedre verktøy enn IR-sensorene, men på grunn av det kreves det mere nøyaktighet i systemet for å utnytte det fulle potensiale til LIDAR-sensoren. Det krever også store endringer på serveren som driver robotene for å gjøre det fornuftig å bruke denne sensoren. På lengre sikt vil det være mulig å erstatte hele IR-tårnet med en eller flere LIDAR-sensorer. Det vil bety store endringer på både roboten og serveren for å gjøre dette.

Forord

Denne oppgava er ei masteroppgave for 2-årig master, teknisk kybernetikk på Norges Teknisk-Naturvitenskaplige Universitet (NTNU). Masteroppgava ble skrevet våren 2018 og utgjør grunnlaget for evaluering i faget *TTK4900 Teknisk kybernetikk*. Oppgava teller 30 studiepoeng og avslutter masterstudiet.

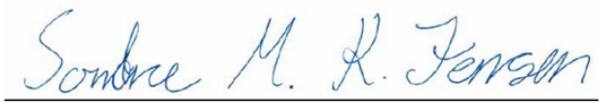
Fra fordypningsprosjektet, TTK4551 Teknisk kybernetikk (høsten 2017), ble det konstruert en Arduino-robot etter den gamle modellen fra Andersen og Rødseth sin masteroppgave 2016. Denne roboten hadde all maskinvare montert og koblet opp, men hadde mangler både i programvare og på koblingsbrettet, som kobler alle delene av roboten sammen. Det ble derfor lagt stor vekt på å rette disse feilene slik at roboten ble operativ i dette masterprosjektet. Mer om utgangspunktet fra forrige prosjekt finnes under seksjon 1.2.

En stor takk går til Elektrisk- og Mekanisk-verksted i kybernetikkblokka på Gløshaugen. Elektrisk-verksted har vært til stor hjelp ved å stille til disposisjon utstyr for å lodde komponenter og til å dra ledninger på roboten. Mekanisk-verksted har vært til stor hjelp med å konstruere feste til LIDAR-sensoren og nytt feste til servoen, da denne måtte byttes ut.

En takk går til Simen Nilssen som designet et nytt kretskort til Arduino-roboten sin. Dette kort-designet gav utgangspunkt for det som sener ble bestilt og brukt i dette prosjektet. Mer om dette finnes under seksjon 5.3.

Det går også en takk til NTNU og Omega Verksted. NTNU stilte *"motion capture"* studio til disposisjon. Dette gjorde det mulig å måle og loggføre hvordan roboten kjører. Omega Verksted stilte til disposisjon Oscilloskop under feilsøking. Begge disse verktøyene var til stor hjelp under feilsøking og testing av roboten.

Til slutt går en takk til Tor Onshus og medstudentene på kontor G242. Onshus har vært til stor hjelp som veileder ved å tilrettelegge for at prosjektet ble vellykket. Medstudentene på kontor G242 har vert til stor hjelp når det gjelder å diskutere tekniske spørsmål og ved å lage et godt og trivelig arbeidsmiljø.



Sondre Martin Kvellestad Jensen

Trondheim, juni 2018

Innhold

Akronymer

IR	=	InfraRed
PWM	=	Pulse-Width Modulation
LIDAR	=	LIght Detection And Ranging
RTOS	=	Real-Time Operating System
LED	=	Light-Emitting Diode
I2C	=	Inter-Integrated Circuit
SPI	=	Serial Peripheral Interface-bus
SCL	=	Serial Clock Line
SDA	=	Serial Data Line
IMU	=	Inertial Measurement Unit
PCB	=	Printed Circuit Board
SLAM	=	Simultaneous Localization And Mapping

Figurer

Kapittel 1

Introduksjon

1.1 Bakgrunnsinformasjon

Dette prosjektet er en videreutvikling av et robotprosjekt ved NTNU, i Trondheim, kalla ”*LEGO-robot prosjektet*”. Prosjektet ble startet i 2004 [1] og har vært under videreutvikling siden da. Målet med prosjektet er å kartlegge et område ved bruk av autonome roboter. Robotene bruker avstandsmåling til å lage en punktsky som representerer omgivelsene og sender disse til en Java-basert server. Serveren lager så en visuell representasjon av alle punkta som robotene har sendt til serveren. All kommunikasjon mellom serveren og robotene skjer via en Bluetooth-enhet montert på robotene og en montert på serveren.

Ved starten av vårensemesteret 2018 fantes det to slike roboter. En robot som bruker en Mindstorm NXT til å kjøre roboten, og en som bruker et Arduino-kort. Begge disse robotene kartlegger omgivelsene ved hjelp av IR-sensorer som er festet på et roterende tårn på roboten. Disse sensorene har et maksimalt rekkevidde på 80 cm [2], men er mest presise på en avstand mellom 10 - 40 cm. Denne begrensingen gjør at roboten selv er nødt til å traversere gjennom hele rommet for å få kartlagt det. Dette kan ta lang tid og det kan oppstå unøyaktigheter mellom posisjonen roboten tror den befinner seg i og der den faktisk befinner seg.

Serveren som kommuniserer med roboten skal ha mulighet til å kommunisere med flere roboter på en gang. Dette skal gjøre det mulig for robotene å samarbeide med å kartlegge området. Ved starten av dette prosjektet er Arduino-roboten ute av drift, og NXT-roboten er eneste opperative robot. Det er derfor lagt vekt på å lage flere roboter som kan brukes i dette systemet slik at ”*LEGO-robot*” prosjektet kan utvikles videre.

1.2 Prosjektoppgave 2017

I prosjektoppgava 2017 ble det som nevnt gjort et forsøk på å lage en nyere versjon av Arduino-roboten. Det ble også gjort arbeid for å erstatte en av IR-sensorene på roboten med en LIDAR-sensor, som har et mye større rekkevidde. Dette prosjektet er dokumentert i rapporten fra prosjektet [3]. Følgende er en oppsummering av hva som ble gjort gjennom prosjektoppgava.

Det ble lagt stor vekt på hardware til roboten i prosjektet 2017. Alle delene ble bestilt med tanke på hvordan de kunne forbedre roboten uten at en måtte gjøre store endringer i koden. Disse ble så koblet samen ved hjelp av et kretskort fra tegningene til Andersen og Rødseth [4]. Dette kretskortet viste seg å ikke stemme med hvordan Arduino-roboten var koplet på det tidspunktet. For å rette disse feilene måtte en gjøres mange lokale endringer, som å trekke kabler og skape brudd på kretskortet. Det ble derfor avgjort at Nilsen skulle designe et nytt kretskort til bruk i hans masterprosjekt og i dette masterprosjektet. Tegninger og koplingskjemaer som viste hvordan kretskortet virkelig var koplet ble da laget for å gjøre denne prosessen lettere. Mer om det kretskortet finnes i Nilsen sin rapport [5]. Dette kretskortet ble ikke ferdigstilt ved slutten av 2017 prosjektet og implementering av dette kortet ble derfor en oppgave i dette masterprosjektet.

Gjennom konstruksjon av roboten ble det lagt stor vekt på å løse problemer som Andersen og Rødseth hadde støtt på med sin robot. Dette var blant annet problemet med støy på ulike sensorer, støy ved kommunikasjon over Bluetooth, hjulene spant under kjøring og dårlig kontakt på Bluetooth-dongelen. De fleste av disse problemene ble løst ved å endre plasseringen av komponenter på roboten og på den måten skjerme både IMU, kompass og Bluetooth-dongelen fra støykilder. Det ble brukt andre hjul og andre motorer for å hindre roboten fra å spinne, og det ble designet et nytt feste til Bluetooth-dongelen som gjord det lettere å kople den av og på uten å slite ut kontaktpunktet.

Ettersom noen av komponentene på LIDAR-roboten er ulike komponentene på den gamle Arduino-roboten, måtte koden til roboten endres. Servoen og IR-tårnet brukt på LIDAR-roboten ble tatt fra en gammel kassert robot. Dette ble gjort for å spare tid, men viste seg å ikke være lurt ettersom den nye servoen trengte andre pulser for å rotere mellom 0 og 90 grader, noe som tok tid å kalibrere.

Begge Bluetooth-donglene som står for kommunikasjonen mellom serveren og roboten ble kodet og ferdigstilte i løpet av prosjektet 2017. Ved enden av prosjektet 2017 var dette den eneste delen av roboten som fungerte som den skulle. Når server-dongelen kjørte og klient-dongelen var montert på kretskortet til roboten, vil klient dongelen blinke rødt helt til den får kontakt med server-dongelen. Når dette skjer vil begge donglene blinke i takt for så å lyse rødt. Dette indikerte at begge dongelene hadde gjennomført en ”*handshake*” og er klare til å overføre informasjon.

Ettersom mulighetene for å bruke en LIDAR-sensor istedenfor IR-sensor ble definert som en del av oppgava, ble det gjort forsøk på å måle avstander ved hjelp av denne. Dette ble gjort ved å lage en enkel kode i "*Arduino IDE*" som skrev ut distansen til et objekt i cm. For å oppnå dette ble det tatt i bruk et bibliotek fra Sparkfun [6], som håndterte I^2C kommunikasjonen mellom Arduino-kortet og LIDAR-sensoren. Ettersom roboten ikke fungerte ved enden av prosjektet 2017 ble det aldrig gjort noe forsøk på å implementere LIDAR-sensoren med resten av koden til roboten. Arduino koden som ble skrevet er likevel et bra utgangspunkt å starte fra når sensoren skal kodes om til C.

1.3 Oppgavebeskrivelse

Denne master-oppgava går ut på å ferdigstille roboten som ble påbegynt i prosjektoppgava i faget "*TTK4551 Teknisk kybernetikk, fordypningsprosjekt*" [3]. Gjennom oppgava skal roboten bli programmert slik at den fungerer på linje med resten av de eksisterende robotene som er blitt laget gjennom tidligere master oppgaver. Det skal også gjøres endringer på hvordan roboten identifiserer objekter ved at den nå skal nytte en LIDAR sensor til å måle avstander. Hele denne oppgava kan dermed brytest ned til tre deler: Hardware, Software og LIDAR.

1.3.1 Hardware

Denne delen tar utgangspunkt i den koden brukt på den gamle Arduino-roboten fra Andersen og Rødseth sitt prosjekt [4]. Gjennom prosjektet skal denne koden omprogrammeres og alle drivere må endres slik at roboten blir operativ med IR-sensorer.

- Gjøre endringer på Nilsens kretskort
- Bestille kretskortet
- Lodde på komponenter
- Feilsøke og rette feil underveis

1.3.2 Software

Denne delen tar utgangspunkt i koden brukt på den gamle Arduino-roboten fra Andersen og Rødseth sitt prosjekt [4]. Gjennom prosjektet skal denne koden omprogrammeres, og alle drivere må endres slik at roboten blir operativ med IR-sensorer.

- Endre koden til alle protene som blir byttet på kretskortet
- Endre driveren til motorene
- Endre driveren til servoen.
- Endre driveren til enkoderene
- Feilsøke og rette feil som oppstår underveis

1.3.3 LIDAR

Etter at roboten er operativ, skal den ene IR-sensoren byttes ut med en LIDAR-sensor. Formålet med dette er å teste om kartlegging er mulig med denne type sensor og sørge for at implementering av flere LIDAR-sensorer blir lettere i framtiden.

- Montere LIDAR-sensoren til roboten
- Lage driver til LIDAR-sensoren
- Implementer LIDAR-sensoren i robotens kode
- Endringer på serveren så den kan behandle informasjon fra LIDAR-sensoren

1.4 Innhold i rapporten

Denne rapporten omhandler arbeidet som ble gjort for å ferdigstille den nye Arduino-baserte roboten for det autonome LEGO-robotprosjektet ved NTNU. Rapporten tar også med trinnene mot å erstatte den ene IR-sensoren på roboten med en LIDAR-sensor og implementere dette på en god og fornuftig måte. Det er her tatt utgangspunkt i at leseren har kunnskaper innenfor programmering og elektroniske koblinger.

Rapporten er hovedsakelig delt inn i fire deler:

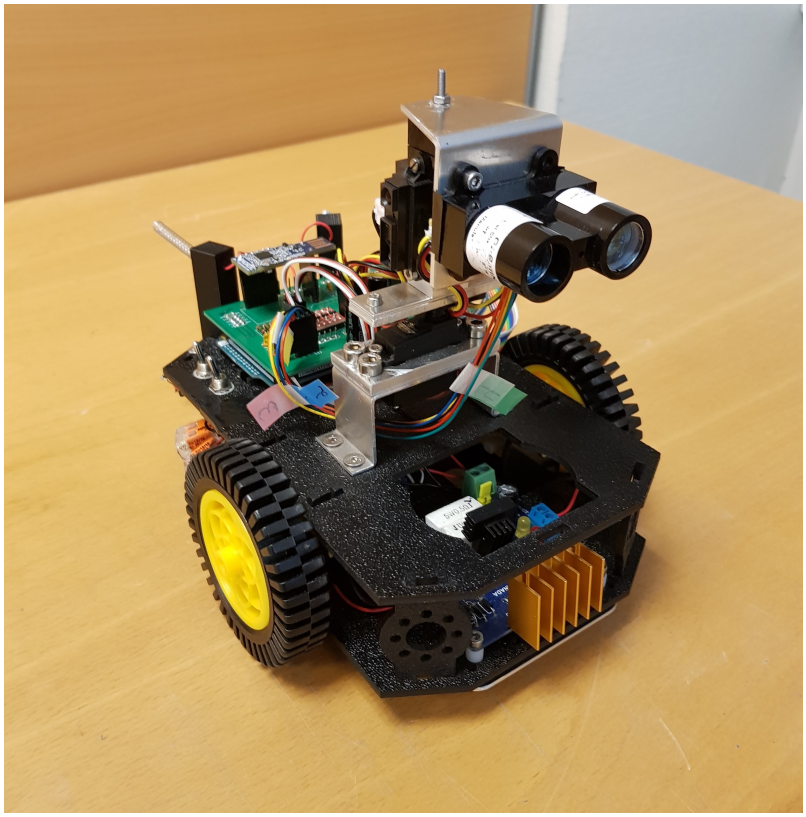
1. IR-baserte roboten
2. LIDAR-basert robot
3. Felles test med flere roboter
4. Diskusjon

Den IR-basserte delen omhandler hvilke endringer som måtte til for å få robotene operativ og til å fungere slik den skulle. Her er det satt fokus på feilene som ble funnet underveis, og hvilke problem som oppstod. Denne delen inkludere også tester som ble gjort mens roboten brukte IR-sensorer til å navigere.

Den LIDAR-baserte delen inneholder forklaringer om hvordan LIDAR-sensoren er tenkt å fungere. Her er det også skrivet om hvordan driveren for sensoren er laget og hvordan sensoren er implementert sammen med både robotens-kode og serverens-kode. Denne delen av rapporten omhandler også tester som er gjort med LIDAR-sensor og sammenlikninger mellom IR-basert robot og LIDAR-basert robot.

Mot slutten av prosjektet ble det gjennomført en felles test av alle de operative robotene i NTNU sitt LEGO-robot prosjekt. Dette ble gjort for å finne ut i hvilken grad robotene fungerte sammen. Denne delen tar også utgangspunkt i hvilken endringer som ble implementerte for å få denne roboten til å virke sammen med resten av robotene, og resultatet av testen.

Til slutt kommer en diskusjonsdel hvor resultatet fra denne masteren blir drøftet. Denne delen inkluderer også de endringene og problemene som fortsatt eksistere ved roboten og hva som bør rettes av den som fortsetter prosjektet.



Figur 1.1: Den ferdigstilte roboten

Kapittel 2

Hvordan roboten fungerer

2.1 Hele systemet

Formålet med dette systemet er å lage en robot som kan kartlegge omgivelsene sine og sende dataen tilbake til en server hvor den blir loggført og illustrert som et kart. Dette oppnås ved å bruke IR-sensorer montert til et roterende tårn på roboten. Dette tårnet gir roboten en 360° synsvinkel der den på en avstand opptil 40 cm blir detektert. Sensorenes rekkevidde er begrenset til 40 cm ettersom den blir svært unøyaktige ved større avstander.

På både roboten og serveren er det festet en nRF51 Bluetooth-dongel. Disse dongelene er kodet for å overføre informasjon mellom dem, med server-dongelen fungerende som en sentral og robot-dongelen som en klient. Dette gjør at flere roboter kan kobles på samme serverdongel, men ikke omvendt. På den måten kan flere roboter kjøre sammen på samme server uten å forstyrre hverandre.

Over disse Bluetooth-dongelene blir informasjon fra hver robot overført til serveren. Det blir også overført hvilken estimert posisjon roboten har. Denne estimerte posisjonen blir funnet som en kombinasjon av kompasset, IMU-en og motorenkoderene på roboten. På serveren blir disse sammenlikna med ønsket posisjon til roboten og kartet oppdateres da avhengig av hvor roboten befinner seg og hva hver sensor har detektert. Roboten sender også over et navn den er blitt tildelt. Dette er nytting for å vite hvilken robot som befinner seg hvor, og blir brukt senere for å skille roboten med LIDAR-sensor fra andre roboter.

Serveren som brukes i dette prosjektet er en Java basert server. Den bruker data fra robotene til å regne ut et optimalt punkt roboten burde reise til, for å oppdage mest mulig uutforska terreng. Serveren vil bare regne ut et nytt punkt om roboten ankommer den ønskede destinasjonen, eller om den møter på en vegg i rett linje til destinasjonen. Da blir de nye koordinatene sendt til roboten, som så roterer og kjører i en rett linje mot sitt nye mål. Koordinatene som sendes fra serveren er utrykt i polarkoordinater.

2.2 Roboten

2.2.1 Deler

Under prosjektet 2017 ble det satt fokus på hvilke deler roboten skulle bygges opp av og hvorfor hver enkelt del ble valgt [3]. Disse delene ble valgt med utgangspunkt i å lage en forbedret versjon av den gamle Arduino-roboten som ble konstruert av Andersen og Rødseth i 2016 [4]. Den viktigeste komponenten i roboten er Arduino-kortet. Dette er et Arduino Mega 2560 [14] med en ATmega2560 mikrokontroller. Det er på denne mikrokontrolleren all koden til roboten blir programmert. Arduino Mega kortet kan bruke en spenningskilde på mellom 7 til 12VDC og kobles derfor direkte til batteriet. Kortet har godt tilgjengelige koblingspunkt til alle pinnene på mikrokontrolleren, noe som gjør det lettere å teste underveis. Arduino-kortet har også en 5V og en 3,3V spenningsutgang som er nyttig til å forsyne ulike komponenter med spenning.

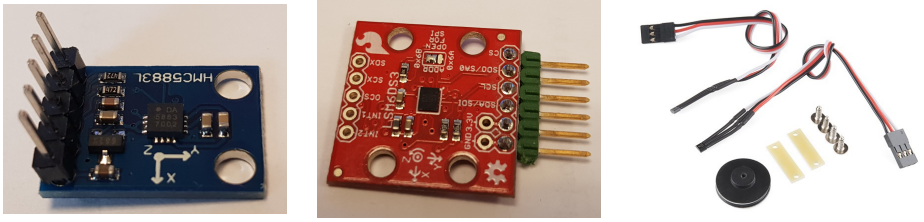
Figur 2.1: Arduino Mega 2560

For å styre motorene ble det brukt en ”*2A Dual L298 H-Bridge*” motorkontroller [15]. Denne kan kobles på en spenningskilde mellom 6 og 35 VDC og er derfor koblet direkte til batteriet. Motorkontrolleren kan forsyne hver motor med opptil 2A strøm og motorkontrollen har også en 5V spenningskilde som kan brukes til å forsyne andre komponenter med spenning. Fra Andersen og Rødseth sin rapport [4] ble det konkludert at denne spenningskilden var mer stabil enn den fra Arduino-kortet, og derfor er denne nytta over 5V fra Arduino-kortet.

Figur 2.2: Motorkontroller

Batteriet som ble brukt var et 11.1V "*Li-ion*" batteri. Dette batteriet har en kapasitet på 4600 mAh og er likt batteriet brukt på den gamle Arduino roboten til Andersen og Rødseth [16]. Ettersom batteriet er helt likt kan roboten lades på samme måte som den gamle Arduino roboten, ved å nytte et dokking system som ble laget under Strande sin masteroppgave [17]. Mer om dette under seksjon 2.3.3.

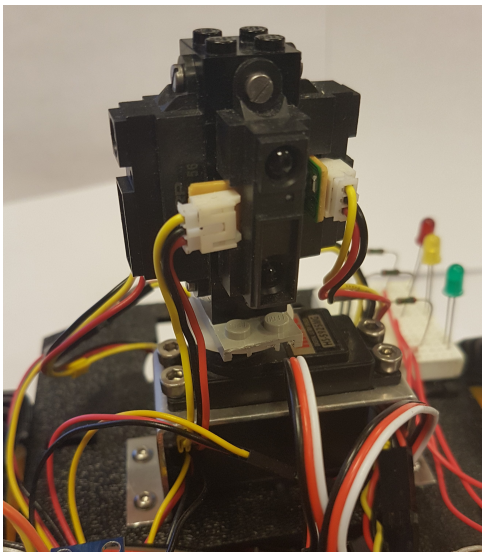
For å hjelpe roboten å navigere ble det montert på et kompass, IMU og motorenkoderer. Kompasset er et "*HMC5883L*" kompass og den bruker magnetisme til å finne x- og y-koordinater som igjen brukes til å finne nord [18]. IMU-en er en "*LSM6DS3*" IMU fra Sparkfun [13]. Den har 6 grader av frihet som er bevegelse i x- y- og z-retning og rotasjon rundt x, y og z. Roboten bruker en kombinasjon av denne og kompasset for å fastslå hvilken vei den peker. Motorenkoderene er enkle enkodere fra DAGU som måler overganger mellom negative til positive pulser med en magnet [19]. Disse magnetene kobles direkte på motoren og roterer med samme hastighet som selve motoren. De blir brukt for å måle distansen hjulene på roboten har rotert og på den måten måle distansen roboten har bevegd seg.



(a) Kompass **(b)** IMU **(c)** Enkoder

Figur 2.3: Navigering komponenter

Midt på roboten er det plassert en servo som roterer et tårn bygget av legoklosser. Fra prosjektet 2017 [3] ble det brukt "*HS-5925MG*" [12], men grunnet hvor mye strøm den trakk når den roterte ble den byttet ut med "*S05NF*" [11]. På hver flatside av tårnet er det festet en IR-sensor. Disse IR-sensorene er av typen "*GP2Y0A21YK*" som er billig og bruker lite strøm. De er mest presise på en avstand på rundt 24 cm og blir mer unøyaktige dess større avvik vi har fra denne avstanden. Sensorene har en rekkevidde mellom 10 – 80 cm [2], men er begrenset til 40 cm på serveren ettersom de blir unøyaktige når avstanden er større.



Figur 2.4: IR tårn

I Andersen og Rødseth sin rapport ble det beskrevet at de opplevde problemer med hastigheten på motorene, i forhold til hvor fort roboten bevegde seg [4]. For å forhindre dette ble det brukt motorer med større gir på den nye roboten. Dermed endte det opp med "*DG01D*" motorer fra DAGU, med et 1:120 girsystem [20]. Dette førte til at motorene måtte gå fortere for å få ønsket fart, men differansen fra roboten begynner å bevege seg til den går for fort, ble også større ved å bruke disse motorene.

2.2.2 Kode

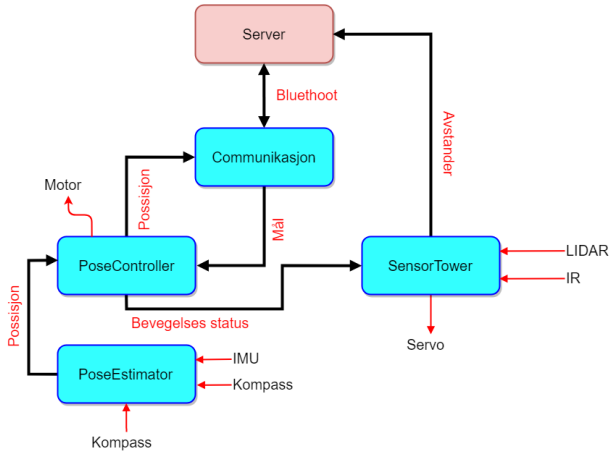
Ettersom mikrokontrolleren ”*ATmega2560*” bare har en kjerne er den bare i stand til å jobbe med en oppgave om gangen [10]. Det blir derfor tatt i bruk et sanntids operativsystem som heter ”*FreeRTOS*”. FreeRTOS gjør det mulig for mikrokontrolleren å bytte mellom flere oppgaver (task's) under kjøring. Dette får det til å virke som om de forskjellige oppgavene kjører samtidig, mens de i virkeligheten bare blir byttet mellom med jamne mellomrom. På denne måten kan for eksempel kommunikasjonen mellom robot og server opprettholdes, mens roboten samtidig kontrollerer at den kjører i rett retning og tar måledata [21].

Koden til roboten er i grove trekk bygd opp av fire FreeRTOS oppgaver og en initialiseringsdel. Initialiseringdelen av koden sørger for å gjøre alle de ulike komponentene i roboten klare for drift. Om en komponent på roboten ikke fungerer kommer det ofte fram her. Initialiseringen skjer før FreeRTOS begynner å bytte mellom oppgaver og blir bare kjørt under oppstart av roboten.

De fire oppgavene FreeRTOS bytter mellom er:

- vMainCommunicationTask
- vMainSensorTowerTask
- vMainPoseEstimatorTask
- vMainPoseControllerTask

Alle disse oppgavene er forklart i mer detaljert lengre nede i denne seksjonen av rapporten. For å illustrere hvoran de ulike oppgavene påvirker hverandre er det laget et flytskjema i figur 2.5.



Figur 2.5: Flytskjema over roboten

vMainCommunicationTask

Denne oppgaven er ansvarlig for å etablere og opprettholde kontakten mellom server og roboten. Denne mottar beskjeder fra serveren og sender informasjon om hvor roboten befinner seg tilbake til serveren. Alle beskjedene den mottar fra serveren er kommandoer om hvilken oppgave roboten skal utføre. Om den skal stå i ro, kjøre etter målet eller finne et nytt mål. Selve operasjonen til de ulike komponentene til roboten blir håndtert under andre oppgaver. For å forhindre at FreeRTOS ødelegger for kommunikasjonen mellom serveren og roboten er det lagt inn begrensninger som gjør at FreeRTOS ikke kan bytte oppgaver mens roboten leser fra serveren.

vMainSensorTowerTask

SensorTower-oppgava har ansvar for alt som skal skje med IR-tårnet på roboten. Den vil rotere tårnet så lenge roboten står stille og vil rotere tårnet tilbake til startposisjon når roboten får beskjed om å bevege seg. Alle målinger fra IR-sensorene blir også gjennomført her. Det hentes ut individuelt for hver sensor og sendes direkte videre til serveren fra denne tasken. Dette er gjort for å få serveren oppdatert så fort som mulig slik at posisjonen til roboten ikke forandrer seg før de nye avstandsmålingene er oppdaterte. Målingene til LIDAR-sensoren blir også håndterte her. De blir tatt og behandlet like etter IR-målingene. Grunnet ulike problemer med målingene fra LIDAR-sensoren ble FreeRTOS forhindret fra å bytte oppgave mens roboten og LIDAR-sensoren kommuniserer. Mer om dette finnest under seksjon 5.4.1.

vMainPoseEstimatorTask

Estimator-oppgava har som formål å estimere hvor langt og i hvilken retning roboten har bevegd seg. Roboten bruker hovedsakelig enkoderene til dette. Som forklart under seksjon 3.3 kan en regne ut avstanden roboten har bevegd seg ved å telle hvor mange pulser enkoderene har telt mellom hvert intervall. I tillegg blir både gyroskopet, kompasset og forskjellen på pulser fra det høyre og venstre hjulet brukt til å estimere hvilken retning roboten kjører. Det er her nytta et Kalman-filter for å filtrere vekk støy fra sensorene og finne de mest presise målingene [22].

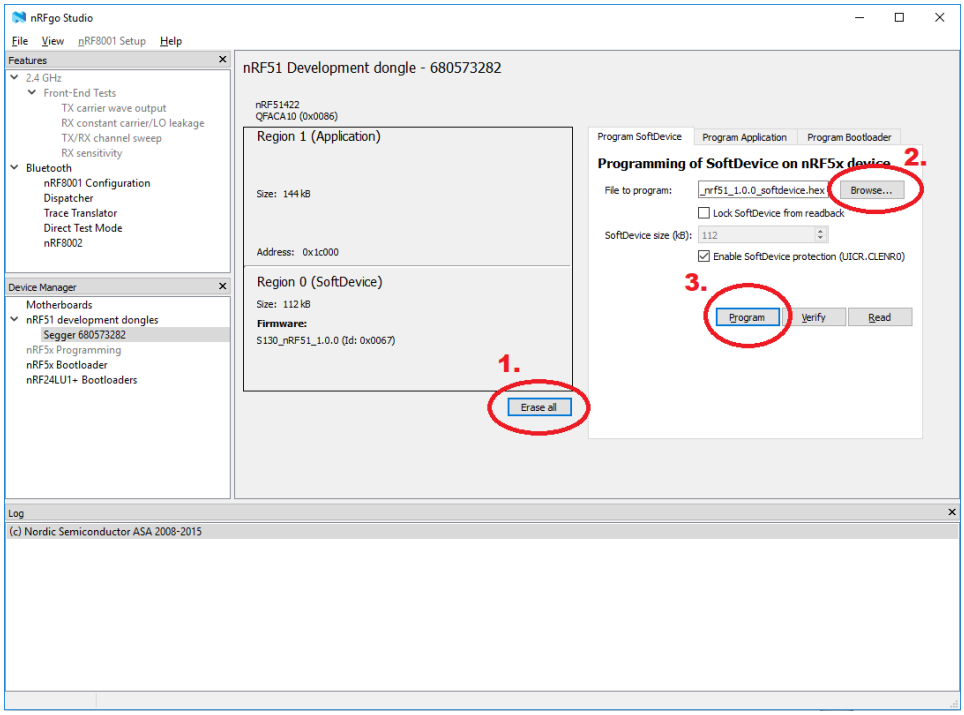
vMainPoseControllerTask

Denne oppgava har ansvaret for at roboten beveger seg i rett retning. Ettersom motorene roterer med ulikt tempo med samme PWM verdi fører dette til at roboten fort kan kjøre skjevt. Det er derfor implementert en enkel PI-regulator i denne tasken. Den mottar ønsket posisjon fra kommunikasjons-oppgava og estimert posisjon fra estimator-oppgava. Deretter regner PI-regulatoren ut hva pådrag hver motor må ha for å kjøre bent, eller for å rette opp i eventuell skjevhet mens roboten kjører. Det er også lagt inn her at i det roboten nærmer seg posisjonen sin, så vil maksimalt pådrag reduseres, noe som gjør at roboten senker hastighet og ikke kjører forbi målet sitt. Denne oppgava sender informasjon til sensor-oppgava om roboten kjører eller står stille og informasjon til kommunikasjons-oppgava om hvilken posisjon roboten har.

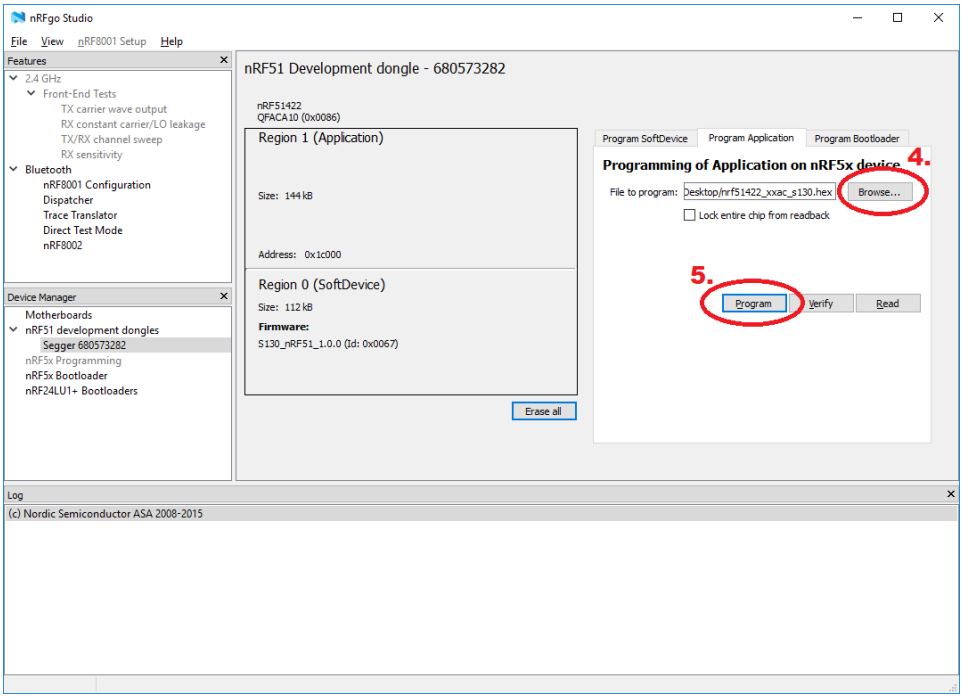
2.3 Praktisk

2.3.1 Klargjøring

For å bruke roboten sammen med serveren trenges det en nRF51 Bluetooth-dongel med sentral-programvare installert. Dette kan bli gjort via "*nRFgo Studio*". Det er da viktig at en først installerer SoftDevice før en installerer sentral-programvaren. Sentral-programvaren nytter funksjoner fra SoftDevice for å få Bluetooth-dongelen til å lete etter klienter. Etter SoftDevice er programmert til dongelen, kan sentral-programvaren programmeres. For å gjøre det lettere å gjenskape denne enheten er all kode som ligger på toppen eksportert fra Keil μVision til en hex-fil som kan programmeres til dongelen via SoftDevice. Både SoftDevice og hex-fila med sentral-koden på, finnest under Vedlegg 1. Bildene under illustrere hvordan både SoftDevice og applikasjons-koden kan bli installert på dongelen. Merk at om noe skal forandres på Bluetooth-dongelen så må dette gjøres i Keil μVision og en ny hex-fil må så eksporteres.



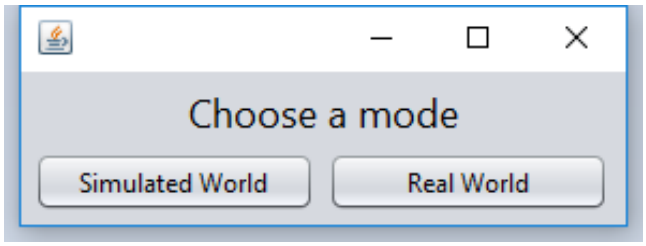
Figur 2.6: SoftDevice i nRFgo



Figur 2.7: Application i nRFgo

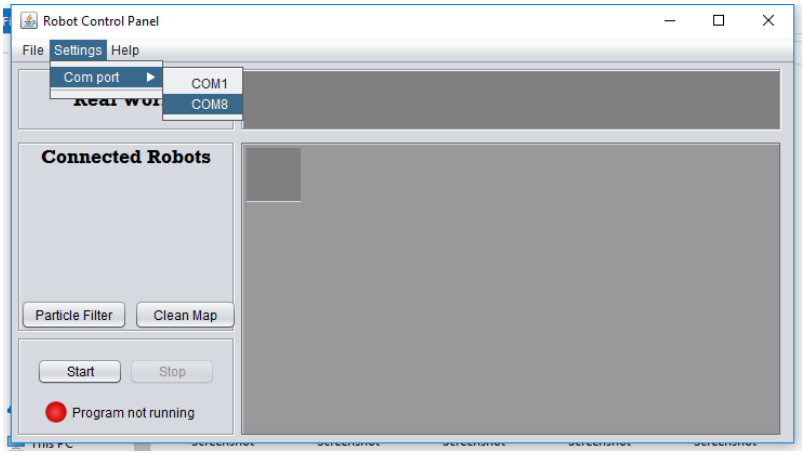
2.3.2 Bruk av roboten

Når roboten skal brukes må en ha en datamaskin med JDK9 tilgjengelig. Da kan serveren startes og en vil få et valg om hvilken modus en ønsker å bruke, som illustrert på figur 2.8. ”*Simulated World*” gir et simulert kart med simulerte roboter. Dette er nyttig om en ønsker å teste funksjoner i serveren. Om en velger Real World får en muligheten til å koble til roboten og kjøre kartlegging i et virkelig område. Det er denne modusen som er brukt i dette prosjektet.



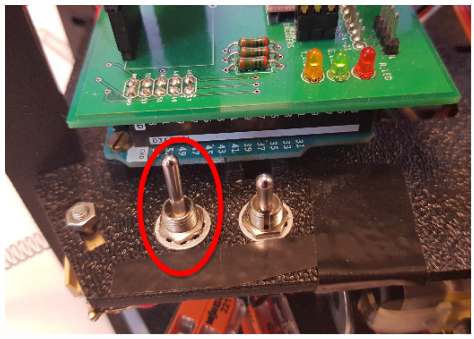
Figur 2.8: Velg en modus

Etter Real World er valgt får en opp et vindu likt det vist i figur 2.9. Her må en velge com-porten Bluetooth-dongelen er koblet til. Dette gjøres ved å velge fanen ”*Settings*” for så å velge ”Com port” og så den aktuelle porten. Om PC-en serveren kjører på har flere com-enheter tilkoblet kan flere com-porter vises i menyen. Da kan en finne hvilken com-port som brukes til Bluetooth-dongelen ved å gå inn på ”Device Mangager” på en Windows-PC.



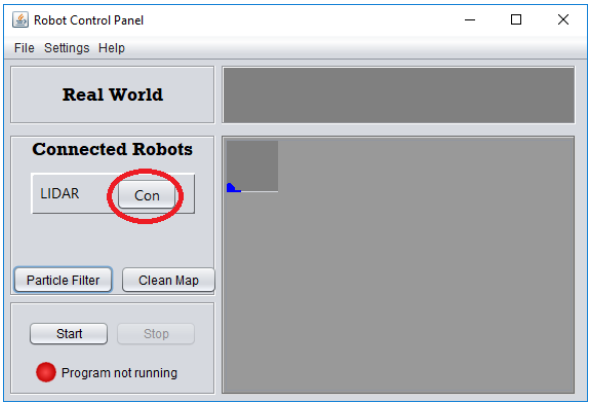
Figur 2.9: Velg COM-port

Når serveren er satt opp kan roboten skrues på. Dette gjøres ved bryteren montert på siden av roboten som vist på figur 2.10. Den grønne lysdioden på roboten vil da begynne å lyse. Dette indikerer at initialiseringen ble vellykket gjennomført og at roboten er klar til bruk. Om det grønne lyset ikke begynner å lyse og det rød lyser istedet, indikerer det at noe har gått galt under initialisering. Dette kan oppstå dersom noe er galt med kompasset, IMU-en eller LIDAR-sensoren. Om ingen lys begynner å lyse etter roboten er skrudd på betyr det at roboten har for lite minne.



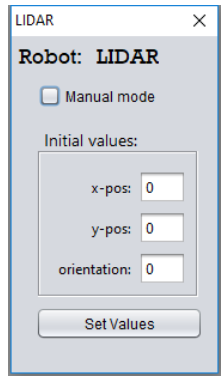
Figur 2.10: Av/På bryter

Når roboten er skrudd på, vil Bluetooth-dongelen starte å blinke rødt. Dette indikerer at den leter etter en sentral å koble seg opp mot. Om den finner en sentral den kan koble til så lyser lysdioden på dongelen konstant rødt. Sentral dongelen som er koblet til serveren vil også lyse rødt for å indikere at det er kontakt mellom den og en klient. Om alt er gjort rett så langt vil roboten dukke opp i lista over tilkobla roboter som vist på figur 2.11.



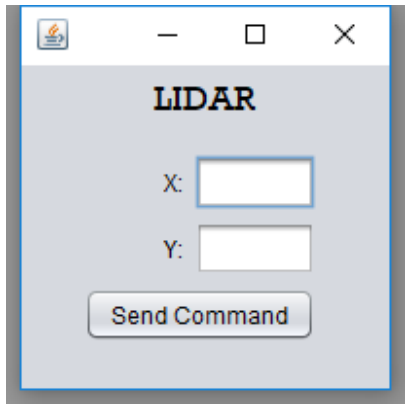
Figur 2.11: Koble til robot

Ved å presse ”connect” ved siden av roboten som skal brukes dukker det opp et nytt vindu som vises på figur 2.12. Her settes start koordinatene til roboten. Disse koordinatene blir brukt om flere roboter skal kjøre sammen. Da må avstanden mellom dem stemme for at målingene skal bli korrekte. Koordinatsystemet bruker høyrehåndsregelen med z pekende opp fra roboten. Dette vil si at x er positiv i retning framover fra roboten og y er positiv til venstre for roboten. Alle avstander er målt i centimeter. ”Orientation” beskriver i hvilken retning roboten er plassert. Om det er flere roboter som brukes i systemet må det spesifiseres hvordan fronten på roboten er plassert rundt z i forhold til hverandre.



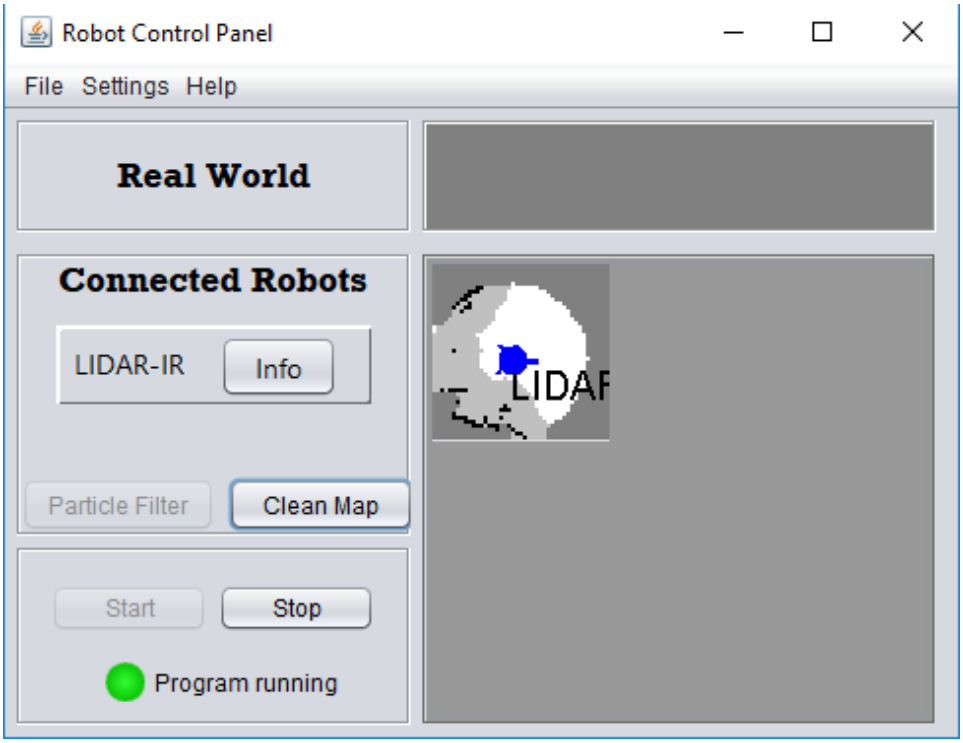
Figur 2.12: Start Robot

Øverst i figur 2.12 er det en merket boks hvor brukeren har mulighet til å kjøre roboten i ”manuell mode”. Dette gjør at roboten ignorerer punktene serveren vil den skal kjøre til. Da kan roboten isteden kjøres manuelt ved å skrive inn ønskede retning eller koordinater. I en servere som bruker polarkoordinater blir dette gjort ved å spesifisere hvor mange grader roboten skal kjøre og hvor langt den skal kjøre i den retningen. Om serveren bruker kartesiske koordinater blir dette gjort litt annerledes. Der spesifiseres hvilken posisjon roboten er ønsket å ende oppi. Da tas det utgangspunkt i hvilken posisjon den ble tildelt i figur 2.12 og roboten vil så bevege seg i en rett linje for å ankomme de spesifiserte koordinatene. Roboten kjøres manuelt ved å trykke på ”info” som står ved siden av den roboten som skal brukes, vist på figur 2.14. Deretter trykker en på ”manuel drive” og får fram et vindu som illustrert på figur 2.13.



Figur 2.13: Manuell styring

Når roboten er koblet til serveren kan kartlegging startes ved å presse start, som vist på figur 2.14. Kartlegging kan bli gjennomført i både manuell modus og i standard modus. Roboten blir representert av en farget sirkel med en linje for å indikere retningen roboten peker. Punktet serveren har regnet ut at roboten skal kjøre til, er vist som et kryss av samme farge som roboten. Etter hvert som roboten kjører vil den avdekke mer av terrenget den navigerer rundt. Hvert gang en av sensorene finner et objekt innenfor rekkevidde vil dette bli illustrert som et svart punkt på kartet. Ethvert slikt svart punkt er omringet av en grå sone. Denne grå sonene indikerer et område roboten ikke kan navigere innenfor, uten fare for å kollidere.

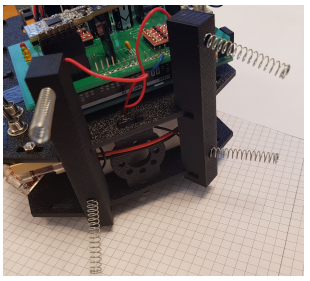


Figur 2.14: Kartlegging

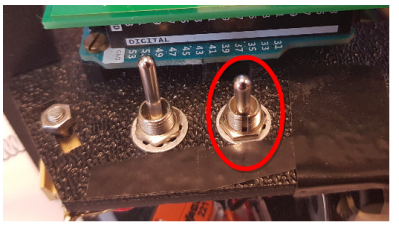
Resten av kartet er farget hvitt. Dette indikerer at det er et åpent terreng her hvor roboten kan navigere gjennom. Det er med andre ord ikke funnet noe objekt her. Når roboten skal navigere til sitt nye mål, i standard modus, vil den unngå alle grå soner og planlegge en rute igjennom de hvite områdene for å nå målet sitt. Etter roboten er ferdig med kartleggingen kan den stoppes ved å klikke på ”Stop”. Da kan en også nytte alternativet ”Clean Map” som renser opp i støy på kartet og drar linjer mellom de ulike punktene som ble oppdaget under kartlegging.

2.3.3 Lading

Under Strande sitt prosjekt i 2017 ble det utviklet et dokking-system for robotene i NTNU sitt LEGO-robot prosjekt. Formålet med denne dokkingen var å gjøre det mulig for roboten å detektere at batterinivået begynner å bli lavt, og få roboten til å navigere til dokkingen på egenhånd [17]. Den roboten dette prosjektet omhandler, har ingen funksjon for å detektere batterinivå, men kretskortet er lagt opp til at en slik funksjon kan legges til ved en senere videreutvikling av roboten. Lading av roboten blir derfor gjort via spiralene bakerst på roboten.



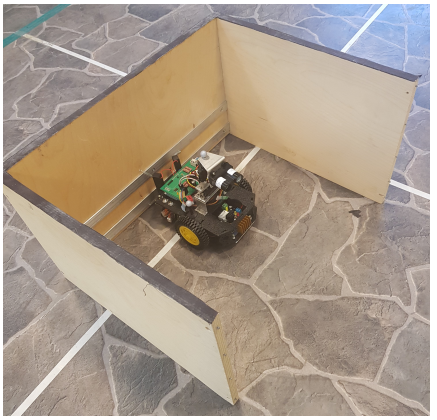
(a) Lade tårn



(b) LadeBryter

Figur 2.15: Lade tårn og bryteren til å aktivere det

For å lade roboten må begge spiralene både oppe og nede ha kontakt med metall platene på dokkingstasjonen. Ettersom det ikke er noen form for strømmåling på roboten gir selve roboten ingen indikasjon på at den blir ladet, men lyset på laderen som gir strøm til dokkingstasjonen vil skifte farge til rød når den lader en enhet. For å forhindre at spiralene har spenning mens roboten kjører er det lagt inn en bryter på roboten for å koble disse ut. Om roboten skal lades må denne settes i på-posisjon.



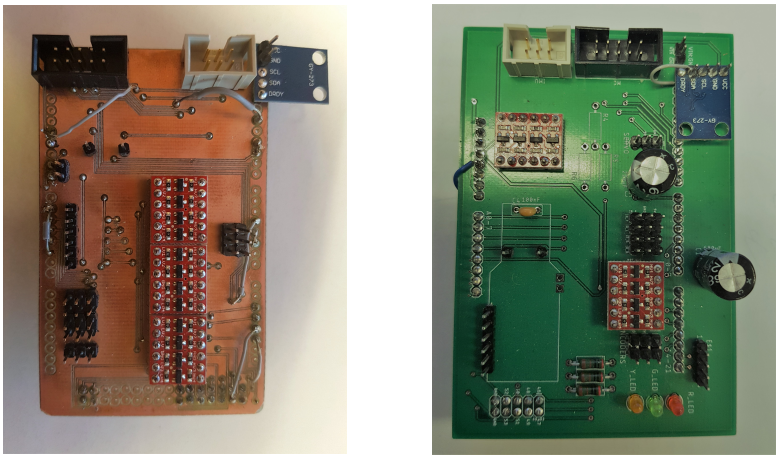
Figur 2.16: Ladestasjon

Kapittel 3

Endringer av robot

3.1 Oppkobling

Alle komponenter på roboten er koblet sammen ved bruk av et kretskort plassert på toppen av Arduino-kortet. Det gamle kortet, som ble designet av Andersen og Rødseth [4], ble i dette prosjektet byttet ut med et nytt designet av Nilssen [5]. Ettersom mange av problemene fra prosjektet 2017 [3] kom av at kortet ble laget på skolen, ble det bestemt at det nye kretskoret skulle lagest profesjonellt. Dette endte opp med at et nytt kretskort ble bestillt fra Seeed Studio [7].

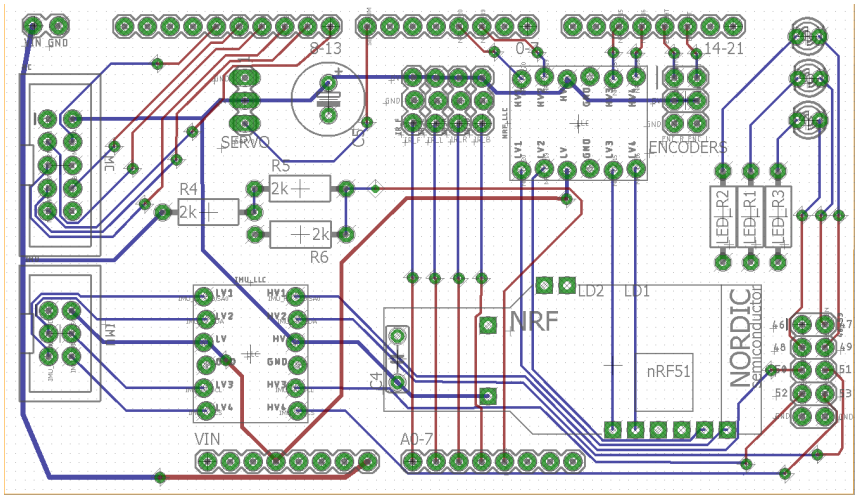


(a) Det gamle kretskortet

(b) Det nye kretskortet

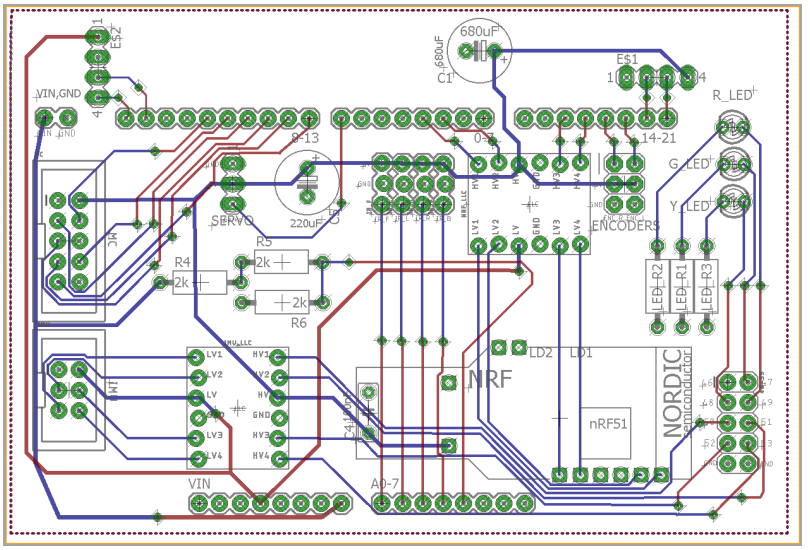
Figur 3.1: Begge versjonene av kretskortet

For å gjøre det mulig å koble til en LIDAR sensor på dette kretskortet, ble det gjort noen endringer på toppen av Nilssen sitt design. Det ble da laget et koblingspunkt for LIDAR-sensoren som henter strøm fra motorkontrolleren, på samme måte som IR-sensorene og Servoen. LIDAR-sensoren kobles direkte til Arduino-kortet sin SDA- og SCL-port for kommunikasjon over I^2C og jord blir koblet til felles jord på kretskortet. Ettersom forsyningsspenningen bør være stabil for at LIDAR-sensoren skal fungere best mulig, var det anbefalt av Sparkfun å koble en 680μF parallelt med LIDAR-sensoren [8].



Figur 3.2: Nilsen sitt design av kretskortet

På Andersen og Rødseth sitt kretskort ble kompasset ikke koblet til kretskortet, men heller koblet gjennom det og rett i Arduino-kortet. Dette ble upraktisk i lengden ettersom denne kontakten var dårlig og førte til feil i kommunikasjon mellom kompasset og Arduino-kortet. For å forenkle kompasstilkoblingen ble kompasset også lagt til Nilsen sitt design [5]. Kompasset blir nå loddet direkte på kretskortet, hvor den henter spenning fra 3,3V spenningskilden fra Arduino-kortet. Jord blir koblet på felles jord på kretskortet og I^2C koblingene går driekte til SDA- og SCL-porten på Arduino-kortet.

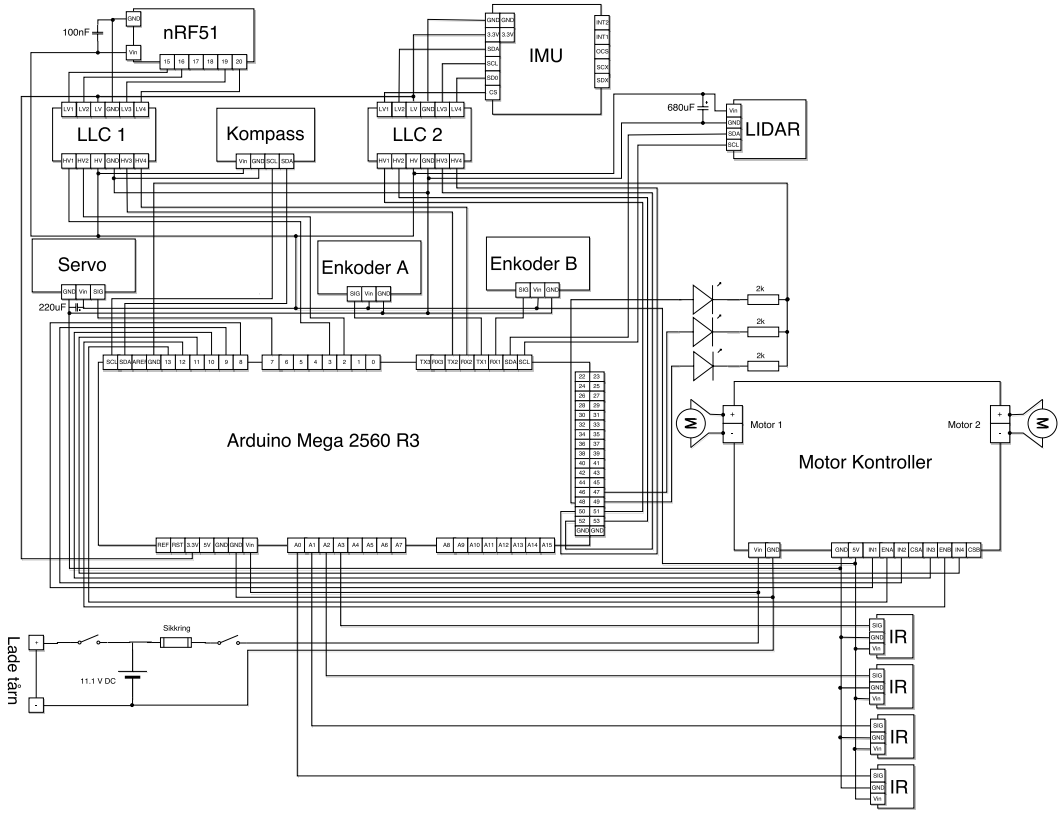


Figur 3.3: Tegning av kretskort med endringer

Dette nye kretskortet endret mange av tilkoblingsportene for de ulike enhetene på roboten. Dette medførte at koblingsskjemaet for roboten også ble endret. Ettersom alle de nye endringene må kodes om i roboten, ble dette lettest gjort ved å lage et nytt koblingskjema over hele roboten. Det kom da fram at disse portene var blitt endret mellom de ulike versjonene av roboten:

Signal	Gammel port	Ny port
MC_IN1(L)	PA2(AD2)	PB5(0C1A)
MC_ENA(L)	PB7(OC0A/OC1C/PCINT7)	PB7(OC0A/OC1C/PCINT7)
MC_ENA(L)	PC3(A11)	Fjernet
MC_IN2(L)	PA4(AD4)	PB4(0C2A)
MC_IN3(R)	PC5(A13)	PH6(0C2B)
MC_ENB(R)	PA6(AD6)	Fjernet
MC_ENB(R)	PG5(OC0B)	PB7(0C0A)
MC_IN4(R)	PC7(A15)	PH5(0C4C)
nRF 17	PA7(AD7)	Fjernet
nRF 18	PC6(A14)	Fjernet
nRF 19	PC4(A12)	Fjernet
nRF 19	PC2(A10)	PE5(OC3C/INT5)
nRF 20	PC2(A10)	PE5(OC3C/INT5)
Servo PWM	PB5(OC1A/PCINT5)	PH4(OC4B)

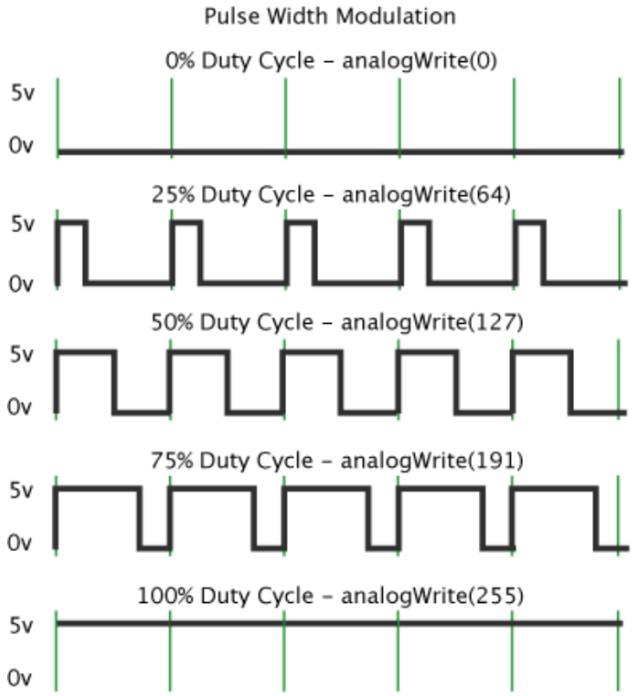
Dette resulterte i følgende koblingskjema:



Figur 3.4: Koblingskjema over hele roboten

3.2 Motorkontroll

Under programmering av motorkontrolleren opppsto et problem der den ene motoren ikke ville kjøre. Det ble da skrevet et kort program i Arduino IDE for å teste om motorene virket. Under testing i Arduino IDE fungerte begge motorene, men den venstre motoren kjørte saktere enn den høyre med samme pådrag.



Figur 3.5: "Duty Cycle" illustrert [9]

Måten motorkontrollen styrer hastigheten på motoren, er at den mottar et PWM signal fra Arduino-kortet for hver motor. PWM står for "*Pulse Width Modulation*" og er en metode Arduino bruker for å simulere analoge signal. Ettersom Arduino bare sender ut 5V eller ingenting, kan den simulere analoge signal ved å bytte mellom høyt (5V) og lavt (0V) signal. Den tiden av en syklus hvor pulsen er høy kalles "*Duty Cycle*" og hvis den er på 50% så simulerer dette et signal på 2,5V [9]. Når motorkontrolleren mottar dette signalet gir den ulik kraft til motoren avhengig av hvor stor Duty Cycle PWM-porten sender ut.

Etter mye feilsøking viste det seg at den venstre motoren var koblet på en 16-bit PWM, mens den høyre var koblet på en 8-bit PWM utgang. I Arudino IDE medførte det at den pulsmengden som ble påførte motoren ikke var lik på begge motorene, og dermed kjørte motorene med ulik hastighet. Når motorene ble implementert i AVR medførte dette at det trengtes et annet register for å velge modus på "*16-bit PWM*"-inngangen. Ettersom den først var antatt å være 8-bit hadde denne porten vært innstilt til "*PWM, Phase Correct, 10-bit*". Ved å endre registeret til "*Fast PWM, 8-bit*" kunne denne PWM-porten også brukes som en 8-bit PWM port og motorene kunne kjøres med samme hastighet [10][s. 145].

Table 17-2. Waveform Generation Mode Bit Description(1)

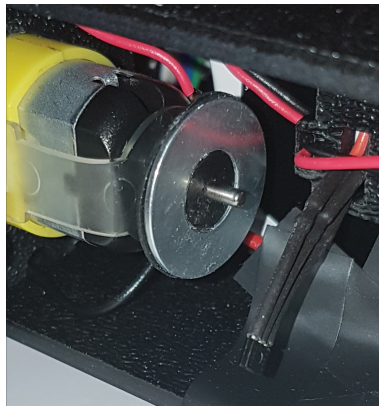
Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Figur 3.6: 16-bit PWM register [10][s. 145]

3.3 Enkoder

Ettersom girene og hjulene på roboten er forskjellige fra de som er brukt på den gamle Arduino-roboten, måtte enkoderene kalibreres. Dette er viktig ettersom roboten regner ut hvor langt den har kjørt og hvor rask motorene kjørere sammenliknet med hverandre ut fra disse enkoderene.

Enkoderene som ble brukt på roboten er 8-pols magnet enkodere. De sender et signal (tick) hver gang de registrerer et bytte fra en positive pol til en negativ og visa versa. Magnet skiva er festet på motoren og roterer i takt med motoren. Da kan vi enkelt regne ut hvor langt hjulet har rotert for hver tick.



Figur 3.7: Magnetskive og enkoder

Vi finner omkretsen av hjulet ved å gange π med diameter, men ettersom girene girer ned hver rotasjon av hjulene med 1:120 må vi dele på 120. Siden det er 8 poler på magnetskiva har vi mulighet til å få 8 ticks for hver gang motoren rotterer. Roboten teller et tick hver gang en får en "*opp puls*", som tilsvarer hver gang den finner en positiv pol på magnet hjulet. Det førte til at vi også må dele på 4. Da sitter vi igjen med den lengden hjulet har bevegd seg for hver tick roboten registrerer.

$$\frac{\pi \cdot Diameter}{Gir \cdot Pulser} = \frac{\pi \cdot 80}{120 \cdot 4} = 0.524mm \tag{3.1}$$

3.4 Servotårn

Gjennom testing av roboten fann IR-sensorene vegger og punkt som ikke eksisterte. Etter roboten har kjørt et stykke, vil det bli registrert så mange ikke-eksisterende vegger at roboten ikke klarer å kjøre videre. Dette var opprinnelig antatt å komme av støy, men etter testing ble det klart at dette kom av hakking i strømforsyninga til IR-sensorene.

"*HS-5925MG*"-Servoen som brukes til å rotere tårnet, bruker samme strømforsyning som IR-sensorene. Dette medførte at når servoen roterte ville den dra mer strøm, som førte til at IR-sensorene ikke fikk 5 volt. Det ble lagt inn en kondensator mellom spenningskilden og jord for å motvirke denne effekten, men ettersom servoen brukt på denne roboten er ulik de brukt tidligere, var ikke kondensatoren stor nok til å motvirke spenningsfallene. Etter å ha eksperimentert med ulike kondensatorer, ble det prøvd å erstatte servoen med den samme som ble brukt på den gamle Arduino-roboten (S05NF) [11]. S05NF servoen bruker mye mindre strøm enn "*HS-5925MG*" [12], og dette løste problemet.

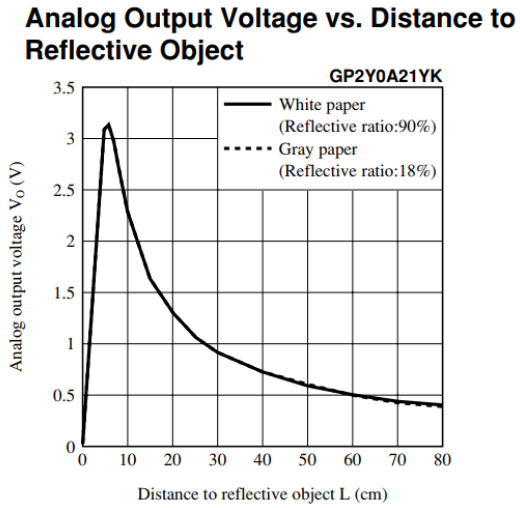
3.5 Kalibrering av IR-sensorer

Måten "*GP2Y0A21YK*"-sensorene fungerer på, er ved å sende ut et infrarødt lys som reflekteres tilbake fra et objekt. Det blir så sendt en analog verdi via datalederen ut fra IR-sensoren som representerer styrken på signalet som ble reflektert tilbake til sensoren [2]. Denne analoge verdien blir så målt på arduino kortet hvor styrken ligger mellom 0 og 5V. Dette blir representert ut fra en verdi fra 0-255 i koden. Denne analog verdien blir så slått opp i en tabell med 255 element hvor hvert element inneholder den tilsvarende avstanden.

```
80  {   // Left sensor connected to PINF0 = 0
81      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,80,80,80,
82      80,80,80,80,80,80,80,80,79,78,77,75,73,71,70,68,66,65,64,62,61,
83      60,58,57,56,55,54,53,52,51,51,50,49,48,47,46,45,44,43,43,42,41,
84      41,40,40,39,39,38,38,37,36,36,35,35,34,34,33,33,32,32,32,31,31,
85      31,30,30,29,29,29,28,28,28,27,27,27,26,26,26,25,25,25,25,25,24,
86      24,24,24,24,23,23,23,23,22,22,22,22,22,21,21,21,21,21,20,20,20,
87      20,20,19,19,19,19,19,19,19,18,18,18,18,18,18,17,17,17,17,17,17,
88      17,17,17,16,16,16,16,16,16,16,16,15,15,15,15,15,15,15,15,15,15,
89      14,14,14,14,14,14,14,14,14,14,14,14,13,13,13,13,13,13,13,13,13,
90      13,13,13,13,13,13,12,12,12,12,12,12,12,12,12,12,12,11,11,11,11,
91      11,11,11,11,11,11,11,11,11,11,11,11,10,10,10,10,10,10,10,10,10,
92      10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,1},
93
```

Figur 3.8: kalibrering tabell til en IR-sensor

Uheldigvis er det ulikt fra sensor til sensor hvilke avstander analogverdiene representerer. Derfor må hver sensor kalibreres ved å øke avstanden og notere analogverdien etter hvert som avstanden forandrer seg. Det ble en stor forbedring etter at IR-sensorene ble kalibrerte. Hvilke verdier hver sensor gir for de ulike distansene de måler er vist i figur 3.9. Her ser en tydlig at forskjellen i spenning hver sensor leverer, blir mindre dess lengre unna objektet er. Dette gjør at sensoren blir mer unøyaktig dess lengre unna objektet er.



Figur 3.9: Spenning til distanse forhold [2]

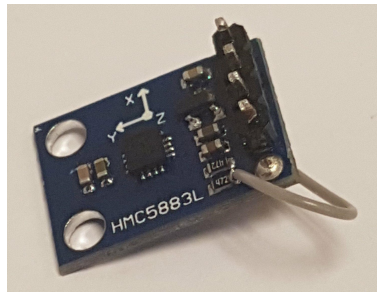
3.6 Skadet kompass

I tidligere rapporter står det at kompasset på roboten ikke blir brukt i koden, men har vært forsøkt implementert og kan kommenteres inn i koden [22]. Det viste seg at kompassdelen av koden var viktigere for robotens evne til å navigere vellykket. Kompasset på roboten fungerer som en kontroll for hvilken retning roboten står i, i forhold til kartete som blir laget på serveren.

Dette førte til problemer under programmering av roboten. Siden kompasset ble sett på som unødvendig for å få roboten operativ ble den nedprioritert i starten. Når roboten så skulle testes for første gang viste det seg at roboten havnet i en uendelig loop under initialisering der roboten venter på svar fra I^2C-porten fra kompasset. Det ble da oppdaget at det var gjort en feil på det nye kretskortet. Koblingen for SDA og SCL gjekk begge til feil port, noe som gjorde det umulig for roboten å kommunisere med kompasset over I^2C. Dette ble løst lokalt på kretskortet ved å dra ledninger og kutte koblingen som vist på figur 3.10.

Figur 3.10: Retting av feilen på kretskortet

Ettersom kompasset var loddet fast i det gamle kretskortet måtte den loddes av og på det nye før det kunne brukes til roboten. Dette førte uheldigvis til at kompasset tok skade under lodding. Denne skaden gjorde at kompassets x- og y-akse ble feil, noe som lagde støy for navigeringen til roboten. Denne støyen gjorde det umulig for roboten å kjøre i en bein linje. Dette ble først ordnet ved å kommentere vekk kompasset fra roboten, men dette var heller ingen bra løsning da avvik i rotasjonen på roboten ikke lenger ble rettet. Når roboten hadde kjørt en stund ville avviket være så stort at roboten ikke markerte hindringer rett i forhold til virkeligheten.



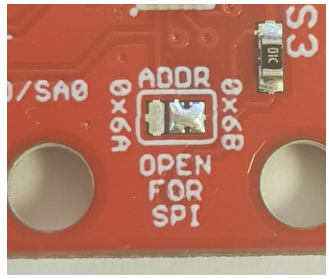
Figur 3.11: Reparert kompass

For å finne ut om kompasset var skadet etter lodding, ble det brukt et multimeter for å lokalisere brudd der kompasset så svidd ut. Det ble da funnet et brudd mellom den ene motstanden på kompasset og SDA inngangen på kompasset. For å ordne dette ble det loddet på en ledning mellom disse to punktene. Dette er vist i figur 3.11.

Deretter ble kompasset testet i Arduino IDE, der en kunne dra nytte av ”*serial monitor*” til å lese ut verdier for å se om kompasset fortsatt virket. Det viste seg at kompasset leverte helt forventa verdier nå. Kompasset ble så kommentert tilbake i roboten og robotens navigering forbedret seg betraktelig.

3.7 IMU

Under koding av roboten oppstod det også et problem når IMU-en skulle tas med i koden. Når roboten kom til den delen av koden der IMU-en ble initialere, hengte roboten seg opp under kommunikasjon over SPI. Etter mye feilsøking kom det fram at for å nytte SPI-funksjonaliteten til "*LSM6DS3*" måtte det loddes vekk en kortslutning mellom to pinner på chipen. Dette kom ikke tydelig fram i databladet til komponenten, men står skrevet fysisk på enheten [13].



Figur 3.12: IMU hedder

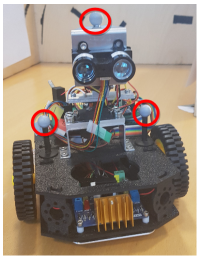
Etter denne kortslutningen ble loddet vekk kunne IMU-en kommunisere over SPI som ment. Likevel oppstod det et problem når målingene fra IMU-en skulle brukes til å hjelpe roboten navigere. Når roboten skulle rotere ville den begynne å rotere i en retning for så å ombestemme seg og rotere tilbake. Dette viste seg å komme fra hvordan IMU-en var montert under roboten. IMU-en var blitt skrudd fast i feil retning og derfor registrerte den en rotasjon til høyre som venstre og omvendt. Etter dette ble rettet opp klarte roboten å rotere i riktig retning slik at fronten peker mot den ønskede destinasjonen.

Kapittel 4

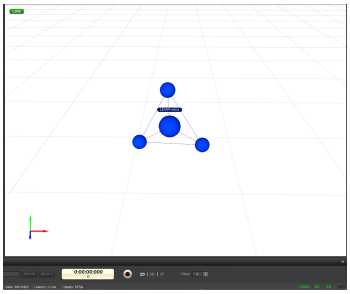
Funksjonstest med IR-sensor

4.1 Kjørepresisjon

For å kontrollere hvor presist roboten kjørte, ble det gjennomført tester ved *Motion Tracking*"-utstyr. Dette utstyret ble gjort tilgjengelig av NTNU via B333 hvor et helt "OptiTrack" system var tilgjengelig. Ved å ta i bruk programmet "*Motion*" og reflekterende markører kunne bevegelsene til roboten måles og dokumenteres mer presist enn ellers mulig. Dette blir gjort ved å bruke kameraer som er festet i taket rundt en bane. Disse er kalibrerte slik at de vet hvor i et x-, y-, z-rom de befinner seg, og kan dermed regne ut hvor hver markør befinner seg.



(a) Robot med markører

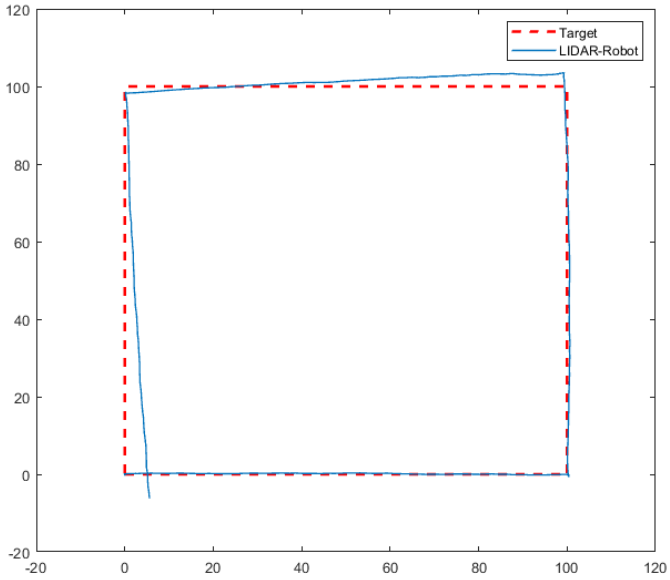


(b) Markørene i Motion

Figur 4.1: Motion Tracking utstyr

For å detektere rotasjon må minimum tre markører festes til roboten, og hver markør må plasseres slik at de til enhver tid blir filmet av minimum 3 kamera. Det ble derfor festet en på hver sida av IR-tårnet og en på toppen av roboten, som illustrert på figur 4.1. Etter disse markørene er festet på roboten kan vi så detekteres i Motion, og ved å merke formen de tre markørene lager kan roboten spores.

Formålet med denne testen var å måle hvor bra roboten klarte å kjøre til gitte koordinater, og hvor stort avvik som ble skapt mens roboten kjørte. Roboten ble dermed satt i manuell modus hvor alle koordinatene roboten skal kjøre til styres manuelt. Roboten ble derfor satt til å starte i posisjon 0.0 for så å kjøre til 100.0 etterfulgt av 100.100 så 0.100 og til slutt tilbake til 0.0. Dette skulle ideelt gitt et perfekt kvadrat med 100 cm langs hver side, men i praksis forventer vi avvik fra dette.



Figur 4.2: Presisjonstest

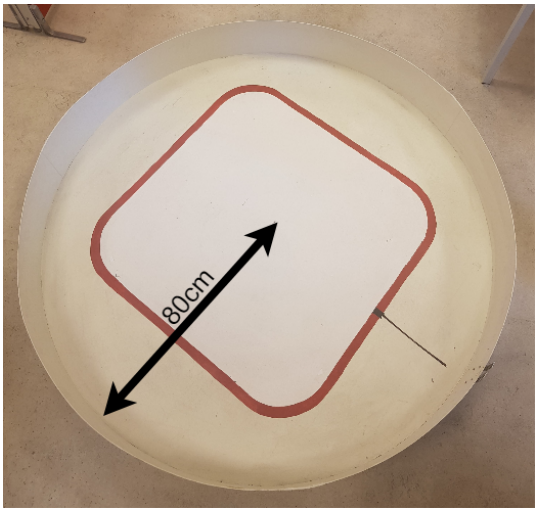
Figur 4.2 viser resultatet av denne testen. Dette er resultatet etter at alle feilene beskrevet i kapitel 3 er blitt rettet. Som en kan se ut fra figuren klarer roboten å kjøre helt bent langs banen, men har et lite avvik når den svinger. Dette avviket kommer trolig av flere grunner. Den første er at komponentene som er brukt til å måle rotasjon er billige og derfor mindre presise. Kompasset på roboten er ikke bra skjermet for støy og reprasjonen til skaden, forklart under seksjon 3.6, gjør denne komponenten mer utsatt for støy. Når roboten kjører i manuell modus gjør den ingen handling for å hindre at den kjører forbi målet sitt.

Tross avviket under rotasjon er roboten presis nok til å gjennomføre målinger og få kartlagt objekt den finner til en logisk plassering. Under seksjonen 8.2 er det diskutert hva som kan gjøres under videre arbeid for å gjøre roboten mer presis.

4.2 Kartlegging

4.2.1 Sirkelbane

For å teste hvor bra roboten klarer å kartlegge i kontrollerte omgivelser, ble det tatt i bruk en testbane. Denne testbanen ble i starten gjort så enkel som mulig for å ha få variabler som kan forstyrre sensorene på roboten. Testbanen består av en sirkel hvor roboten skal finne alle veggene i sirkelen og plassere de rett i henhold til hvor roboten befinner seg. Ettersom IR-sensorene kan oppføre seg ulikt avhengig av hvilken overflate de møter, er hele sirkelbanen hvit.

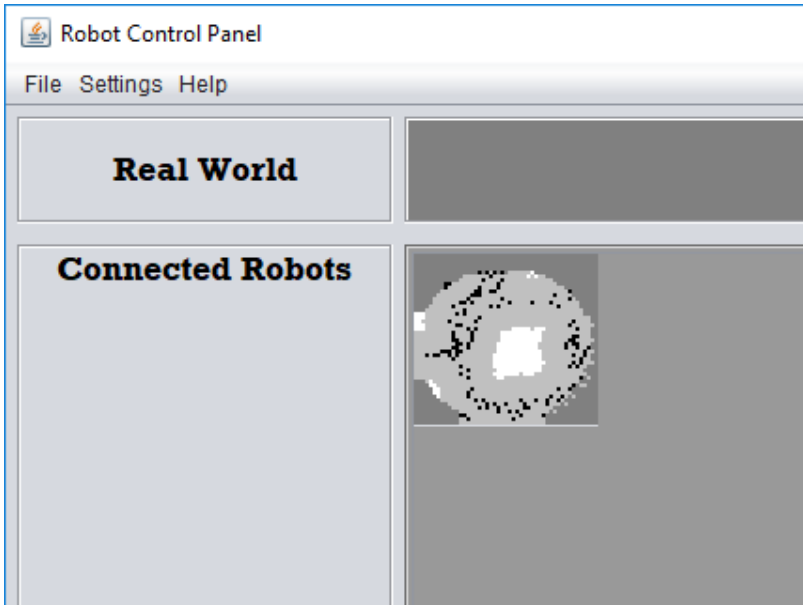


Figur 4.3: Sirkelbane

Denne testbanen ble nyttet under testing og feilsøking gjennom alle punktene i kapittel 3. Dette gav en visuell tilbakemelding på hvor bra roboten virker i praksis og gjorde det lettere å identifisere problemer med kartleggingen som ikke kom fram ellers. Denne banen var også ideell å bruke ettersom den var lett å transportere samtidig som den var stor nok til at roboten måtte kjøre et stykke for å kartlegge hele banen.

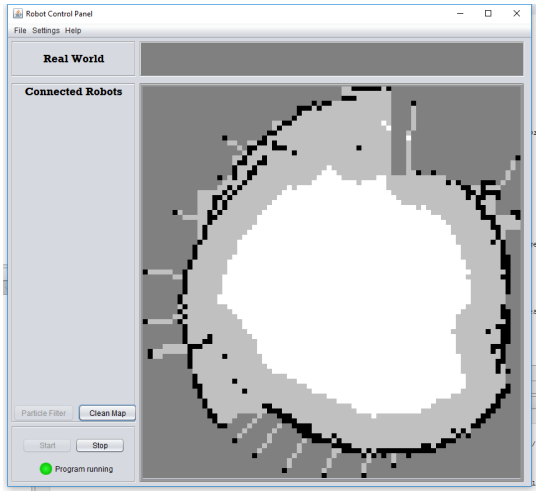
4.2.2 Testresultat

Sirkelbanen ble brukt som testbane gjennom feilsøking av roboten, ble mange mislykkede forsøk på å kartlegge sirkelen før roboten var istand til å gjøre denne jobben. Den første feilen som ble identifisert etter et forsøk på å kartlegge testbanen var servo-feilen forklart under seksjon 3.4. På figur 4.4 kan en tydelig se at roboten er omringet av små svarte prikker. Disse prikkene kommer av at drift spenningen over IR-sensorene har falt slik at signalet de gir tilbake til roboten er lavere enn det den tilsvarende distansen til objektet burde gitt. Dermed finner sensorene objekter som ikke eksisterer og etter litt tid er det så mange slike at roboten ikke klarer å kjøre.



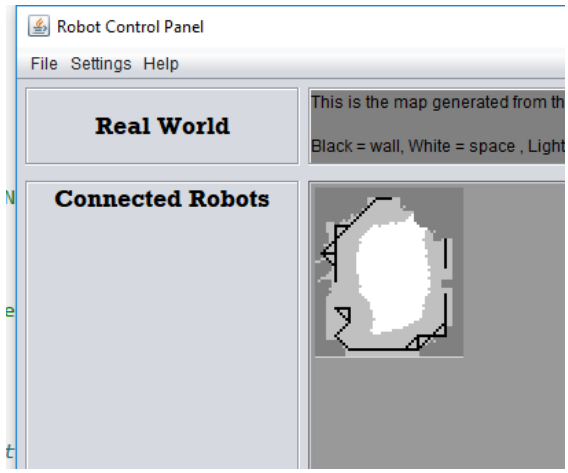
Figur 4.4: Målinger med strøm problemer

Etter servoen ble byttet ble resultatet på banen betraktelig bedre. Roboten klarte nå å kjøre rundt og detektere vegger rundt sirkelen. Problemet nå var at hver sensor ikke var konsekvent på hvilken avstand de rapporterte til serveren. Dette gjorde at vegger som var blitt kartlagt av en sensor ble endret når en annen sensor detekterte dem. Etter roboten hadde kjørt gjennom banen noen ganger utartet dette seg som at veggene forskjellige sensorer markerte ikke var på rett plass i forhold til hverandre. Dette er vist på figur 4.5.



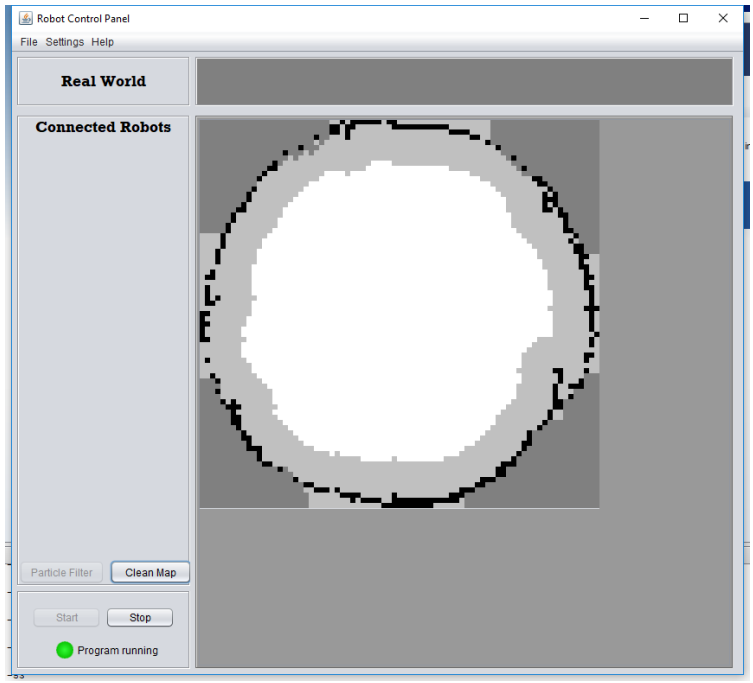
Figur 4.5: Ukalibrerte IR-sensorer

Denne feilen oppstod av at hver IR-sensor leverte ulike verdier for samme avstander. Ved å kalibrere hver sensor manuelt klarte IR-sensorene å finne samme avstander til samme objekter. Etter det var gjort, ble resultatet som vist på figur 4.6. På denne figuren er veggene sammenhengende, men sirkelens form er ulik den i virkeligheten. Dette kom av at roboten her kjørte kun ved hjelp av gyroskopet på IMU-en. Når kompasset ble fikset og kodet tilbake i roboten ble resultatet betraktelig bedre.



Figur 4.6: Test uten kompass

På figur 4.7 ser vi at sirkelen er blitt vellykket kartlagt og representerer den virkelige testbanen. Kartlegging ble her gjennomført ved bruk av autonom kartlegging og resultatet var bra nok til at implementering av LIDAR-sensoren kunne prøves. Roboten ble senere testet i en mer kompleks bane under seksjon 6.3 og seksjon 7.2.



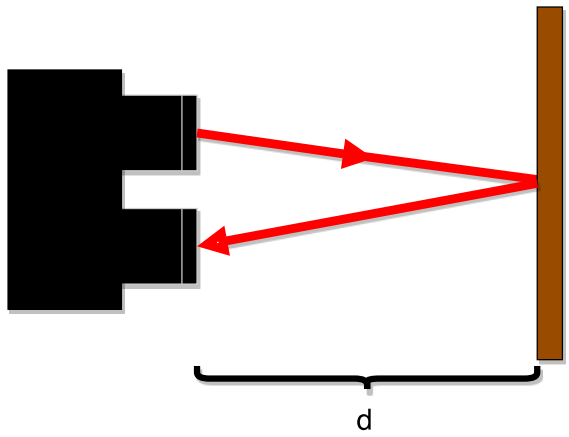
Figur 4.7: Vellykka test

Kapittel 5

LIDAR

5.1 Teori

LIDAR-sensoren er en fellesbetegnelse for lysbasert sensorer som bruker en laser for å måle avstander. LIDAR står for "*Light Detectuin and Ranging*" og blir brukt i alt fra romforskning til fartskontroll. Sensoren måler avstander ved å sende ut en lyspuls fra sensoren som så blir reflektert tilbake når den treffer et objekt. Når lyset blir reflektert tilbake til sensoren blir det observert av et kamera på sensoren og tida (t) lyset brukte på å reise fram og tilbake blir lagret. Siden vi bare skal måle avstanden til objektet, og lyset har reist fram og tilbake, må denne tida deles på to. Ved så å multiplisere med lysets hastighet (c) finner sensoren så avstanden til punktet. Hele avstanden kan dermed regnes ut ved hjelp av denne formelen: $d = \frac{c \cdot d}{2}$ [23].



Figur 5.1: Hvordan LIDAR måler lengde

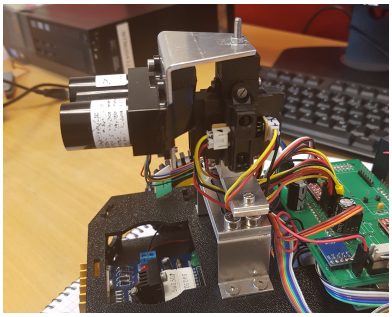
LIDAR-sensorer kan bruke de aller fleste typer lys for å detektere avstander, men siden infrarødt lys ikke er synlig for mennesker blir det brukt i de fleste LIDAR-sensorer. Ettersom roboten skal brukes i normale omstendigheter hvor mennesker kan befinne seg er det viktig at vi bruker en laser som ikke vil være skadelig for mennesker. For å definere hvilken laser som er lov å bruke har Statens strålevern tatt i bruk et klassifiseringssystem hvor ulike laserer klassifiseres etter hvilken risiko de representerer. Klasse 1 er den mest vanlige laseren som blir brukt, og den er ufarlig for mennesker [24].

Figur 5.2: LIDAR lite v3

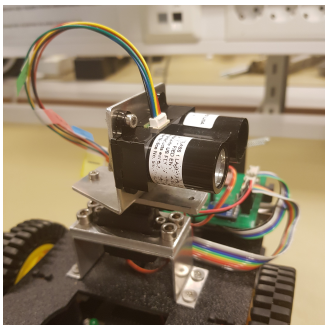
LIDAR-sensoren som blir brukt i denne masteroppgava er en LIDAR lite v3, som er utviklet av Garmin. Den har en rekkevidde på 40 meter og bruker mindre enn 130 mA per måling. Sensoren trenger 5 V driftsspenning for å være operativ og trekker rundt 100 mA ved kontinuerlig drift. Den har mulighet til å bli konfigurert av brukeren og kan kommunisere med andre enheter over I^2C eller PWM [26].

5.2 Montering

LIDAR-sensoren ble først montert på roboten som en erstatning for den ene IR-sensoren. Dette ble enkelt gjort ved å lage et feste til LIDAR-sensoren som så ble festet på toppen av det allerede eksisterende sensor-tårnet. Det gav den fordelen at det ble enkelt å bytte mellom IR-sensor og LIDAR-sensor, noe som var til stor hjelp igjennom feilsøking og koding av roboten. På denne måten kunne roboten enkelt testes i samme omgivelsen som tidligere, kun med den endringen at den nå bruker en LIDAR-sensor i stedet for en IR-sensor. Dette visest på figur 5.3a.



(a) Feste med IR-sensorer



(b) Feste uten IR-sensorer

Figur 5.3: Begge versjonene av feste for LIDAR-sensorene

Festet for LIDAR-sensoren ble også designet slik at sensoren kan erstatte hele IR-tårnet. Formålet med det var å finne ut om det var mulig å kartlegge ved bruk av bare en LIDAR-sensor. Dette er illustrert på figur 5.3b. Ettersom servoen som roterer sensoren bare roterer 180○ var planen å bruke en IR-sensor på baksiden av LIDAR-sensoren. Det viste seg at om servoen fikk beskjed å bevege seg til akkurat 180○, eller 0○, så begynte den å spinne ukontrollert i sirkel. Dette hendte også for alle distanser over 180○, eller under 0○. For å forhindre at denne feilen skulle oppstå måtte servoen begrenses til 5○ fra disse grensene. Ettersom rekkeviddet til servoen da ble under 180○ var det ikke mulig å utføre en fulstendig 360○ søk ved bruk av kun to sensorer. Tiltak for å bruke dette festet er videre diskutert under videre arbeid (8.2).

5.3 LIDAR driver

I høst prosjektet 2017, ble det laget et Arduino IDE program som måler avstand til et punkt i centimeter. Dette programmet benyttet et bibliotek Gerber hadde laget og et I^2C bibliotek integrert i Arduino for å kommunisere mellom sensoren og mikrokontrolleren. Disse bibliotekene forenklet prosessen med å sette opp sensoren og forespørre etter data. Begge disse bibliotekene måtte lages fra grunnen av i C-kode, for at de skulle virke i sammen med resten av roboten.

5.3.1 I^2C

I^2C er en av de vanligste måtene å kommunisere mellom ulike komponenter og en mikrokontroller. Det fungere ved hjelp av en SDA og SCL port, der hver komponent kobles i serie med desse to. SCL er en klokke-port, som har i oppgave å sørge for at alle komponentene snakker med samme hastighet. SDA er en dataport og der skubbes dataen over til alle enhetene som er koblet i serie. Mikrokontrolleren som styrer kommunikasjonen i denne koblingen kalles "*master*" og alle komponentene som er koblet i serie kalles "*slave*".

Ettersom I^2C er en av de vanligste kommunikasjonsprotokollene var ikke denne så vanskelig å implementere. I^2C koden består hovedsakelig av 4 funksjoner: Initialisering, start kommunikasjon, skriving, lesing og stopp kommunikasjon [26].

$$SCL\_F = \frac{CPU\_F}{16 + 2 \cdot TWBR \cdot Prescaler} \tag{5.1}$$

Initialisering av I^2C skjer kun en gang i programmet. Der settes hastigheten kommunikasjonen skal skje med. Den regnes ut fra denne formelen 5.1. Her er F_CPU hastigheten til microkontrolleren, F_SCL hastigheten kommunikasjonen over I^2C skal gå med. Ettersom vi er interesserte i å finne ut hvile verdi vi må skrive til registeret (TWBR) for å opnå en bestemt kommunikasjons hastighet, må vi snu formelen. Vi ender da opp med formel 5.4.

$$TWBR = \frac{\frac{F\_CPU}{F\_SCL} - 16}{2 \cdot Prescaler} = \frac{\frac{16000000UL}{100000UL} - 16}{2 \cdot 1} = 72 \tag{5.2}$$

Startdelen av koden brukes hver gang kommunikasjonen mellom master og slave starter. Den setter verdier i registeret for å starte kommunikasjon. Etter det er overført, skrives en adresse til enhetene på kanalen. Denne adressa er meint å være 8-bit adressa til komponenten som skal kommuniseres med. Etter denne adressa er sendt til programmet venter den på en bekreftelse på om at adressa er mottatt fra slaven. Om slaven sender en bekreftelse vil programmet fortsette, ellers vil en feilmelding skrives.

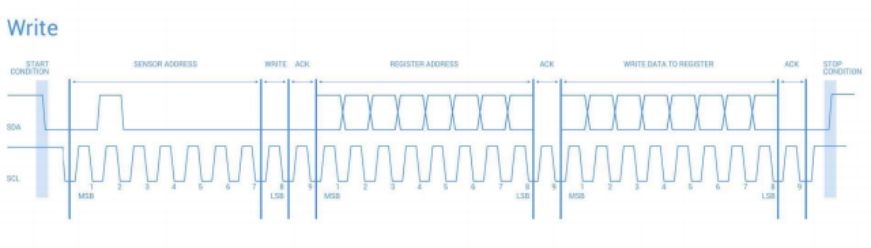
Både lesing og skriving vil som navnet sier, brukes til å enten lese informasjon fra slaven eller skrive informasjon til den. All informasjon som skrives eller leses er 8-bit, så om mer informasjon skal overføres må flere skrive- eller lese-funksjoner kjøres. Det ble også laget en bekreftelse-versjon av både lese- og skrive-funksjonen. I likhet med start-funksjonen vil disse sjekke etter en bekreftelse fra slaven på kommunikasjon og skrive en feilmelding om den ikke mottar dette.

Stop-funksjonen kommer etter skriving eller lesting er ferdig. Den skriver et stop-bit i registeret til masteren, som stopper I^2C kommunikasjonen. Etter dette er enheten klar til å sende en ny start kommando neste gang den vil lese eller skrive noe.

5.3.2 LIDAR lite v3

Koden som styrer LIDAR-sensoren tar i bruk I^2C koden som er beskrevet i seksjon 5.3.1. Denne koden setter opp ulike moduser for LIDAR sensoren, på samme måte som i Arduino biblioteket fra Sparkfun [6], og skriver til registrer som beskrevet i LIDAR lite v3 sitt datablad [23]. De tre hovedkomponentene av denne koden er: Initialisering, Reset og Distanse.

LIDAR-driveren har også en lese-funksjon og en skrive-funksjon. Disse har som oppgave å sørge for at bits som skal skrives til eller leses fra sensoren blir overført korrekt. Skrive-funksjonen brukes under andre funksjoner hver gang noe skal endres på sensoren og lese-funksjonen brukes når noe skal leses ut fra sensoren. Begge disse metodene har bekreftelse innebygd i koden. Dette skal sørge for at en feilmelding blir vist, dersom det blir problemer med kommunikasjonen til enheten. Skrive-funksjonen blir brukt i initialisering-, Reset- og Distanse-funksjonene, mens lese-funksjonen blir bare brukt i Distanse-funksjonen som har til oppgave å lese ut måledata fra sensoren. Hvordan skrive-funksjonen fungerer blir illustrert på figur 5.4 og hvordan lesing blir utført er illustrert på figur 5.5.



Figur 5.4: Hvordan skrive over I^2C [23]

Initialisering tar seg av oppsettet av LIDAR-sensoren slik at den er klar for bruk. Her kjøres initialisering av I^2C og konfigurering etter ønsket funksjon på LIDAR-sensoren. Ved å sende inn en heltalls verdi mellom 0 – 5 kan vi velge mellom 6 ulike funksjoner for hvordan LIDAR-sensoren vil måle avstander. Disse valgene gir oss muligheten å endre hvor fort sensoren tar nye målinger. Dette kommer på bekostning av rekkevidde og hvor nøyaktig sensorene måler, noe som gjør sensoren mer sensitiv for støy.

Disse ulike innstillingene for hvordan LIDAR-sensoren måler er som følger:

0. **Standard modus**
   Denne modusen er den måten sensoren er ment å ta målinger på. Den balanserer behovet for raske målinger med presise målinger.

1. **Kort rekkevidde**
   Denne modusen kan settes om en bare ønsker korte målinger med sensoren. Den skrur også opp hastigheten målingene blir tatt med, som gjør den veldig unøyaktig ved større avstander.

2. **Standard rekkevidde**
   Denne modusen gjør at sensoren er nesten like presis som standard modus for målinger over lengre rekkevidder, men skrur opp hastigheten på målinger over korte rekkevidder.

3. **Maksimalt rekkevidde**
   Denne prioriterer målinger over lange rekkevidder og er dermed mer nøyaktig over større områder, men mindre nøyaktig på nære målinger.

4. **Høysensitivitet**
   Denne modusen overstyrer LIDAR-sensorens algoritme til å identifisere gyldige målinger og bruker isteden en terskelverdi til å identifisere hvilke målinger som er relevante å beholde. Ved å sette denne terskelen høy får sensoren en høyere sensitivitet, men er også mer utsatt for støy.

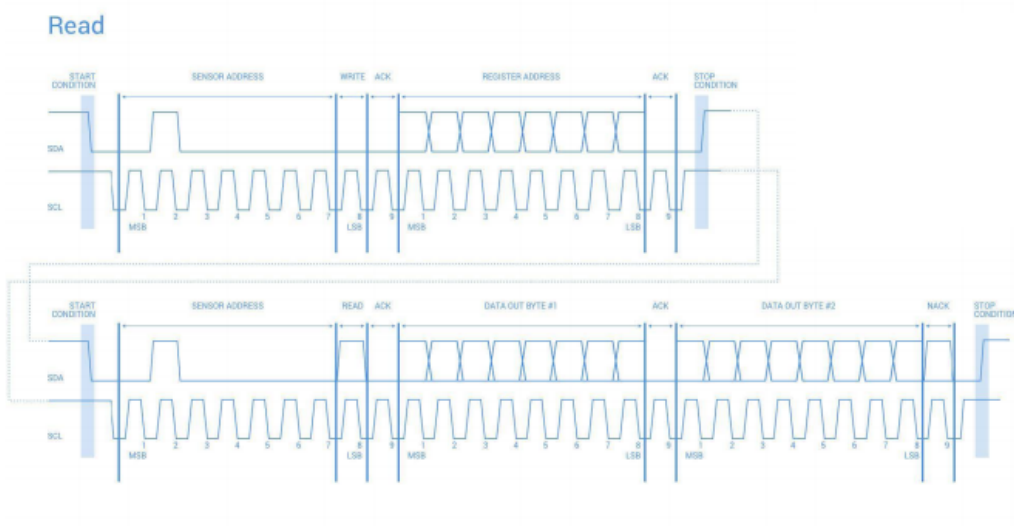
5. **Lav sensitivitet**
   Denne modusen, i likhet med den forrige, overstyrer LIDAR-sensorens algoritme til å identifisere gyldige målinger med en terskelverdi. Her er denne terskelen gjort liten slik at mindre feilmargin blir godtatt, men dermed også mindre målinger. Denne har en lav sensitivitet og er sikrere mot støy.

Disse innstillingene er de samme som fins i databladet til LIDAR lite v3 [23] og blir oppnådd ved å skrive heksadesimal-verdiene som er beskrevet i dette databladet til LIDAR-sensoren.

For å gjøre det mulig å resette alle innstillinger lagret på LIDAR-sensoren, ble det lagt inn en reset-funksjon. Denne funksjonen kan kjøres ved å bruke "*LIDARreset("I^2C-adresse")*" funksjonen. En viktig ting å være klarover om en bruker denne funksjonen, er at I^2C-adressa til LIDAR-sensoren også blir resatt til standardverdi. Dette betyr at om en har bytta adressa tidligere så vil standard-adressa, som er 0x62, måtte brukes for å kommunisere med sensoren videre. For denne roboten er ikke adressa endra, og dette blir derfor ikke et problem. Hvis en skulle implementert flere LIDAR-sensorer på samme robot ved et senere prosjekt, kan det bli en god løsning siden sensorene vil ha samme adresse.

Distanse-funksjonen er hovedfunksjonen i LIDAR koden. Det er her vi henter ut avstanden til objektet sensoren har regnet ut avstanden til. For å starte lesing av data må vi først skrive verdien 0x04 eller 0x03 til register 0x00. Om verdien 0x04 skrives vil LIDAR-sensoren utføre en "*bias-korreksjon*" hvor den bruker måledata til å regne ut en bias på målingen og så retter de neste målingene etter denne biasen. Om 0x003 sendes vil bare en vanlig måling bli tatt. Det er anbefalt å kjøre en bias-korreksjon for hver hundrede måling. I koden velger en mellom disse to funksjonene ved å sende inn en 1 eller en 0 for "*biasCorrection*" i funksjonen "*LIDARdistande("biasCorrection", "I²C-addresse")*".

Etter en måling er tatt av sensoren begynner mikrokontrolleren å lese fra register 0x01 på sensoren. Om første bit-en som blir returnert er 1 så betyr det at sensoren er opptatt og målingene er ikke klare til å bli lest. Hvis den returnerer 0, så er målingene klare. Koden er programmert slik at den vil sjekke 999 ganger om sensoren er klar. Hvis den ikke er klar etter 999 forsøk vil den returnere en feilmelding, dersom ingen ting er galt pleier sensoren å være klar etter et til to forsøk.



Figur 5.5: Hvordan lese over I²C [23]

Når sensoren er klar til å bli lest fra, kan vi kalle på register 0x8f for så å sende et stopp-bit. Dette vil gjøre at sensoren starter å sende informasjon til mikrokontrollene. Dette er vist på figur 5.5. Ettersom sensoren har et rekkevidde på opptil 40 meter blir dette framstilt i 16-bit. Disse blir delt opp i to segment på 8-bit for å overføres til mikrokontrolleren. De første 8 bitene som blir lest fra sensoren må så utføres en "*bit-shift*" operasjon på. Dette flytter de 8 steg til venstre og lager plass til de 8 neste bitene som skal motas fra sensoren. Når dette er gjort kan de sendes videre og leses som en avstand i cm utrykt i binærtall.

5.4 Problemer med LIDAR

Det oppstod noen problemer når LIDAR-sensoren og koden ble implementert med resten av roboten. Dette var problemer som tok tid å finne ut av og som forhindra at roboten kunne kjøre med den nye sensoren.

5.4.1 I^2C krasje

Når LIDAR-sensoren ble koblet til resten av roboten og implementert i koden, startet programvaren mellom roboten og serveren å hakke. Dette førte til at noen få målinger ble tatt før kommunikasjonen mellom serveren og roboten ble tapt. Hver gang dette hendte måtte hele kartleggingsprosessen startes på nytt.

Etter mye feilsøking viste det seg at freeRTOS gjorde det mulig å kjøre kompasset og LIDAR-sensoren samtidig. FreeRTOS blir brukt på roboten for å kjøre flere tasker samtidig ved å veksle mellom dem på prosessoren. Ettersom både LIDAR-sensoren og kompasset kommuniserer over I^2C ble det da et krasj i kommunikasjonen, da feil data blir sendt over SDA porten. I koden førte dette til at LIDAR-sensoren ikke fikk et klarsignal og fortsette å søke helt til en "*timer*" gikk ut.

Ettersom den delen av koden som omhandler innhenting av data fra IR-sensorene hengte seg opp i LIDAR-koden, førte dette til at nye målinger aldri ble gjennomført. Den delen av koden som starter oppdatering av informasjon til serveren ligger i samme "*task*" som innhenting av distansemåla. Dette betyr at serveren ikke fikk noe mer informasjon fra roboten og serveren tror derfor at den har mistet kontakt og henger seg opp.

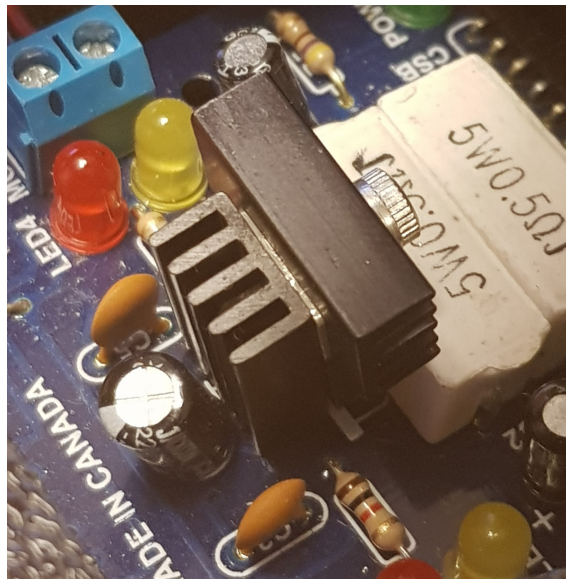
```
taskENTER_CRITICAL(); //Disable interrupt while LIDAR is talking
distLIDAR = LIDARdistance(0,LIDARLITE_ADDR_WRITE); //receive measurement in cm
taskEXIT_CRITICAL(); //Enable interrupt
```

Figur 5.6: Deaktivering av freeRTOS avbrudd

Dette ble enkelt ordnet ved å prioritere henting av data til LIDAR sensoren. Ved å gjøre den til en kritisk funksjon forhindrer det freeRTOS fra å bytte mellom "*tasker*" når LIDAR sensoren snakker over I^2C. Etter å ha gjort LIDAR-funksjonen kritisk klarte LIDAR-sensoren å gjennomføre målinger som ønsket, selv om de kunne være litt ustabile.

5.4.2 Varm transistor

Når roboten ble testet med IR-sensorer oppstod det et problem der transistoren på motorkontrolleren ble svært varm. Det ble først antatt å skyldes at motorene på denne roboten måtte styres med mer spenning enn på den gamle Arduino-roboten. Det ble derfor montert på to kjøleribber på denne transistoren for å redusere varmen. Dette løyste problemet så lenge roboten brukte IR-sensorer, men når LIDAR-sensoren ble koblet til kretskortet ble transistoren for varm, tross kjøleribbene.



Figur 5.7: Kjøleribber montert på transistor

LIDAR-sensoren ble koblet parallelt med IR-sensorene, og servoen til tårnet, med samme spenningskilde fra motorkontrolleren. Når LIDAR-sensoren ble koblet til kretskortet ble transistoren på motorkontrolleren enda varmere. Det viste seg da at LIDAR-sensoren trenger hele 135mA under kontinuerlig drift [23]. Dette øker betraktelig mengden strøm motorkontrolleren må levere til kretskortet. Dette førte igjen til at transistoren bygde opp mer varme enn før.

Dette problemet ble også løst lokalt på kretskortet til roboten. Ved å kutte kontakten mellom spenningskilden fra motorkontrolleren og isteden hente 5 volt driftsspenning fra Arduino-kortet. Dette førte også til at målingene utført fra LIDAR-sensoren ble mer stabile, ettersom det ikke var noen andre komponenter som drar strøm direkte fra denne spenningskilden.

5.5 Implementering av LIDAR

Initialisering av LIDAR-koden ble lagt helt i starten av robotens kode, like etter kompasset og IMU-en blir initialisert. Dette medfører å kjøre LIDARbegin() som er forklart under seksjon 5.3.2. Denne biten av koden ligger under "*#ifdef LIDAR_sensor*" som betyr at hvis LIDAR er definert i starten av koden så vil denne biten kjøre. Dette er den letteste måten å implementere LIDAR-sensoren på som fortsatt gjør det mulig å kjøre roboten ved kun å fjerne "*#define LIDAR_sensor*".

```
#define LIDAR_sensor
#ifdef LIDAR_sensor
LIDARbegin(0,LIDARLITE_ADDR_WRITE);
#endif
```

Figur 5.8: Initialisering av LIDAR-sensor

Selve avstandsmålingene med LIDAR skjer under programoppgava "*vMainSensorTowerTask*". Dette er samme programoppgave som håndterer avstandsmåling fra IR-sensorene. Målingen fra LIDAR-sensoren skjer like etter alle IR-sensorene har tatt målingene sine og i likhet med initialiseringen ligger denne delen av koden også under "*#ifdef*". For å sørge for at biasen blir oppdatert med jamne mellomrom, ligger det en teller i denne koden som sørger for at hver tiende gang det blir igjennomført en måling så vil det også regnes ut en ny bias. Dette gjøres enkelt ved å bytte ut 0 fra "*LIDARdistance(0, LIDARLITE_ADDR_WRITE)*" til 1 som forklart under seksjon 5.3.2.

```
if (biasCorrect=100)
{taskENTER_CRITICAL(); //Disable interrupt while LIDAR is talking
    distLIDAR = LIDARdistance(1,LIDARLITE_ADDR_WRITE); //receive measurement in cm
    taskEXIT_CRITICAL(); //Enable interrupt
    biasCorrect=0; //Count for next bias correction
}else{
//LIDAR handling
taskENTER_CRITICAL(); //Disable interrupt while LIDAR is talking
distLIDAR = LIDARdistance(0,LIDARLITE_ADDR_WRITE); //receive measurement in cm
taskEXIT_CRITICAL(); //Enable interrupt
biasCorrect++; //Count for next bias correction
}
```

Figur 5.9: Hente målinger fra LIDAR-sensor

Det viste seg fort at å ta målinger på 40 meter ikke var nyttig i praksis. Ettersom servoen roterer med 5° for hver måling, fører dette til at det er et stort sprang mellom hver måling LIDAR-sensoren utfører. Dette blir problematisk for kartlegging-systemet til serveren ettersom den fyller inn mellomrommet mellom hver måling med antatte verdier ut fra hva som er blitt målt tidligere. Ved å nytte grunnleggende trigonometri kan vi regne oss fram til hvor langt hver måling på 40 meter vil forskyve seg mellom hver måling. Hvis vinkel α er 5°, b er lengden på ene benet i en likebeinet trekant og β er en av de resterende vinklene.

$$\beta = \frac{180° - 5°}{2} = 87,5° \tag{5.3}$$

$$a = \frac{b \cdot \sin(\alpha)}{\sin(\beta)} = \frac{40m \cdot \sin(5°)}{\sin(87,5°)} = 3,45m \tag{5.4}$$

Avstanden LIDAR-sensoren klarer å måle ble dermed redusert til 2 meter. Dette er fortsatt fire ganger så stor avstand som IR-sensorene klarer å måle stabilt og sparer dermed mye tid under kartlegging. Denne reduksjonen av makslengde blir enkelt håndtert ved å bruke en if-løkke. Denne løkka sier at om distansen LIDAR-sensoren måler overstiger 200 cm vil den sette verdien fra målingen lik null, ellers vil målingen sendes videre. Ved å sette målingen lik null tolker serveren dette som at det ikke finnes noe objekt i søksområdet, ettersom IR-sensorene returnerer 0 når de ikke finner en hindring.

```
//Limit measurement taken by LIDAR
if (distLIDAR+LIDAR_OFFSET_CM>200)
{forwardSensor = 0;
vLED_singleHigh(ledRED);}
else
{forwardSensor = distLIDAR+LIDAR_OFFSET_CM;
vLED_singleLow(ledRED);}
```

Figur 5.10: Begrenser rekkevidden til sensoren

Når LIDAR sensoren først ble brukt sammen med serveren ble bare målinger opp til 40 cm målt. Dette kom av en begrensning i serveren som lager kartet der den kutter ned signalet til en avstand på 40 cm. Ettersom det fortsatt er nødvendig å bruke IR-sensorer, ble det gjort endringer på Serveren for sensor 1. Dette er sensoren som peker framover på roboten. Her ble avstanden økt til 200 cm, slik at vi oppnår det ønskede resultatet på 2 meter per måling.

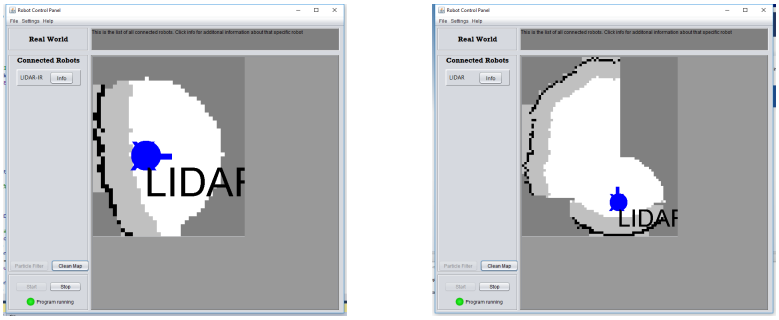
Siden flere roboter er ment å kunne kjøre på samme server, ble det tatt hensyn til dette da serveren ble endret. Når dataen fra IR-sensoren blir håndtert vil serveren nå sjekke navnet på roboten den snakker med. Om navnet på denne roboten er LIDAR, som er navnet på den LIDAR-baserte roboten, så vil den modifiserte versjonen av koden kjøre. Ellers vil koden kjøre akkurat som den har gjort før. Om en senere skulle ønske å bytte ut sensoren på flere roboter vil navnet på de nye robotene også måtte legges inn her for å oppnå bruken av LIDAR på serveren. Alle forandringene finner sted under ”*no.ntnu.et.mapping/MeasurementHandler.java*” i Netbeans.

Kapittel 6

Funksjonstest med LIDAR-sensor

6.1 Test på sirkelbane

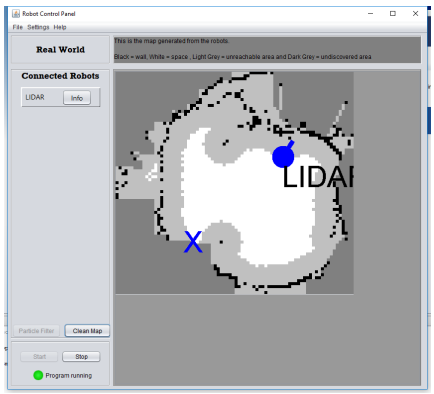
Mens det ble gjort forsøk på å implementere LIDAR-sensoren sammen med resten av roboten, ble sirkelbanen fra seksjon 4.2.1 brukt for testing. Dette var naturlig ettersom banen var enkel å sette opp og det var et bra sammenlikningsgrunnlag med IR-sensorer, fordi det allerede var gjort mange tester på denne banen.



(a) IR søk **(b)** LIDAR søk

Figur 6.1: Søk med ulike sensorer

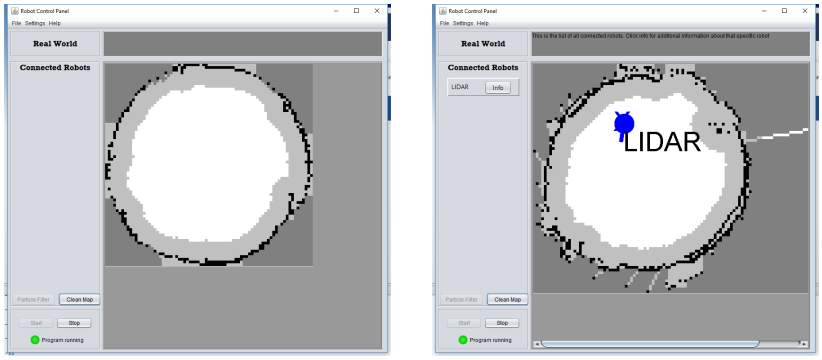
Når roboten starter kartlegging av testbanen får den kjempeutbytte av LIDAR-sensoren. På figur 6.1 ser vi at LIDAR-sensoren klarer å kartlegge helt fram til veggen med et enkelt søk, der IR-sensorene bare gjør søker fram til 40 cm. Dette avslørte et problem med roboten som ikke var kommet fram tidligere. Mens sensorene på roboten kartla området rundt den, var det en stor forskjell på avstanden IR-sensorene rapporterte og avstandene LIDAR-sensoren rapporterte. Det virket som avstanden LIDAR-sensoren fant var dobbelt så stor som avstanden IR-sensorene fant.



Figur 6.2: LIDAR med upresis enkoder

Etter hvert som roboten kjører rundt i sirkelbanen, skriver LIDAR-sensoren over verdiene IR-sensorene finner og IR-sensorene skriver over verdiene LIDAR-sensoren finner og resultatet blir som illustrert på figur 6.2. Dette oppstod som et resultat av at feil distanser var regnet ut for enkoderverdiene, som forklart under seksjon 3.3. Det var først antatt at enkoderene målte hver endring fra negativ til positiv puls, men dette var feil. Det er kun de positive pulsene som trigger et "*tick*".

Dette medførte at distansen som ble registrert av at hjulene roterte, ikke stemte med distansen de roterte i virkeligheten. Når roboten kun kjørte IR-sensorer på en sirkulær bane, var dette ikke et stort problem ettersom den eneste delen som ble feil var distansen på midten av sirkelen. Dermed såg resultatene fra IR-testen ut til å være bra, selv om denne feilen fortsatt eksisterte.



(a) Kartlegging med IR **(b)** Kartlegging med LIDAR

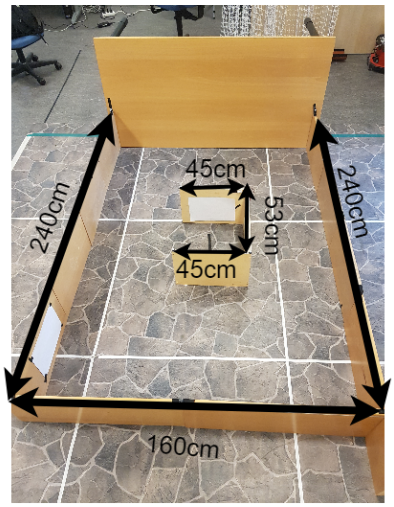
Figur 6.3: Kartlegging av sirkelbane

Med denne feilen retta, ble det gjort nye forsøk på sirkelbanen. Ettersom LIDAR-sensoren klarer å måle store avstander ganske nøyaktig fikk små avvik på robotens posisjon mer å si enn før. Det ble derfor nødvendig å være mer presis med utrekning av hvor langt hjulene har rotert. Ved å fysisk måle hjulene ble det funnet at de hadde en diameter på 78 mm ikke 80 mm som oppgitt av leverandøren [27]. Å endre dette var nok til at målingene fra LIDAR-sensoren i sirkelbanen ble tilnærmet de samme som tatt med IR-sensorene.

6.2 Stor bane

I samband med fellestesten, som er videre drøftet under seksjon 7.2, ble det bygget en mer komplisert testbane for robotene. Denne banen ble bygget på B333 hvor det var god plass og siden vi hadde tilgang på "*motion tracking*" utstyr der.

Denne banen er bygd opp av planker som danner et rektangel med 240 cm lengde og 160 cm bredde. I midten av banen finnes ladestasjonen til robotene. Det er planen at robotene skal starte fra denne ladestasjonen og så kartlegge hele kartet autonomt. Inne i ladestasjonen er det merket et kryss. Dette er for å gjøre startvinkelen og posisjonen, for robotene, mest mulig like for hver test. Siden ladestasjonen ligger i midten av banen, gir dette muligheten til å teste om roboten klarer å navigere rundt hinderet.

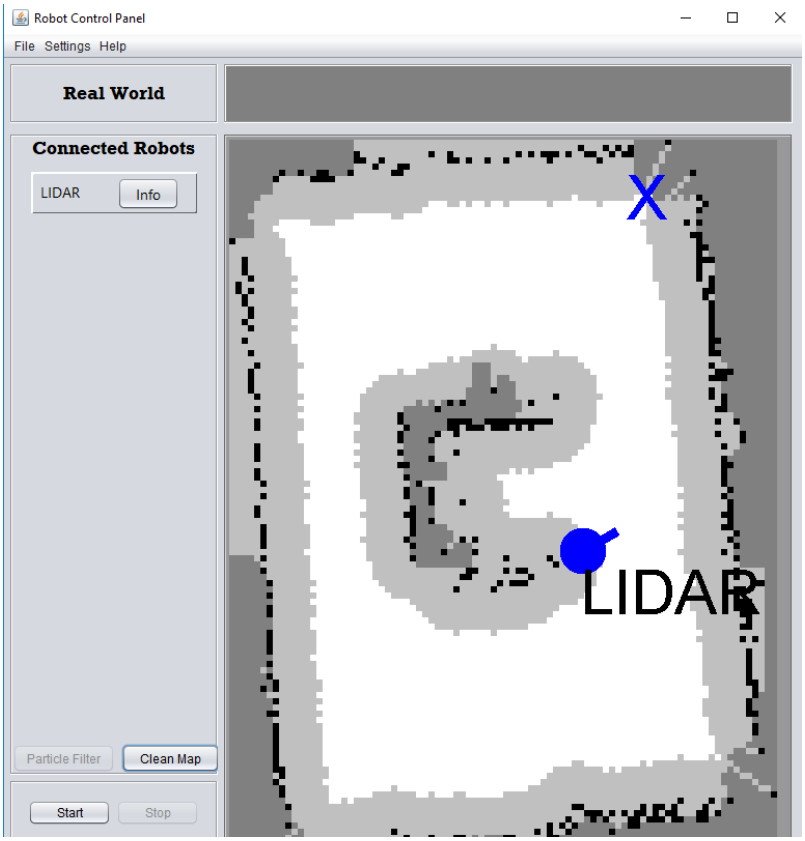


Figur 6.4: Stor bane

Denne banen er mye større en den roboten er blitt testet på tidligere og vil dermed bli en mer reell test av hvordan kartlegging med roboten blir. Dette vil være en mulighet til å kontrollere hvor presis roboten er over tid og kontrollerer hvilket sensoroppsett som kjører gjennom banen raskest. Ettersom veggene på denne banen ikke er hvite, vil dette også være en mulighet til å teste ut hvor bra målinger hver sensor er i stand til å gjøre under mer krevende forhold.

6.3 Test på stor bane

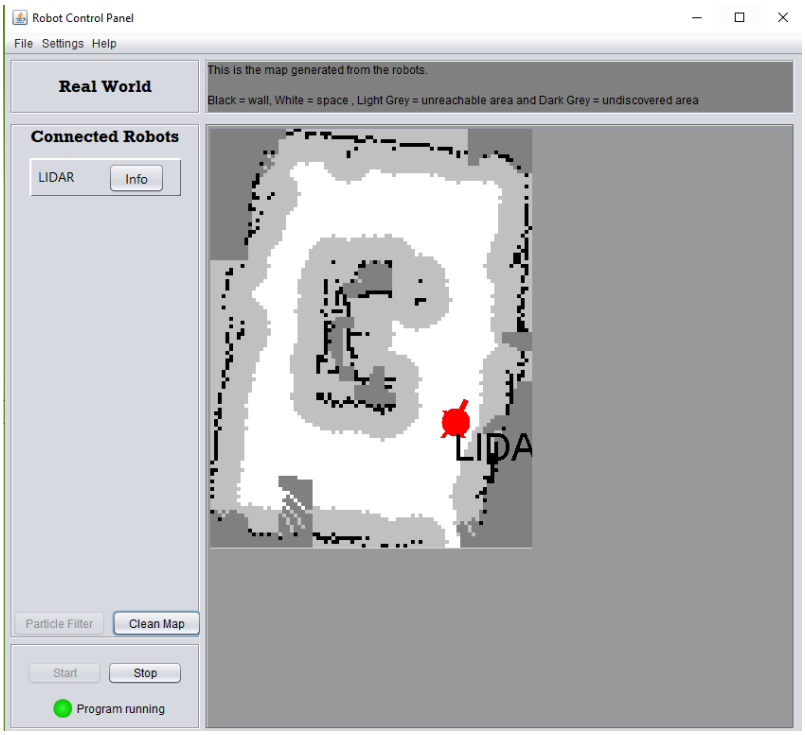
Før fellestesten, seksjon 7.2, ble det gjennomført en test med LIDAR-sensorer og IR-sensorer i den store banen. Dette ble gjort for å se om roboten ville være i stand til å kartlegge banen, og for å avdekke eventuelle feil som ikke kom fram i den lille sirkelbanen. Disse testene ble gjennomført i kartesiske koordinater heller enn polarkoordinater, ettersom kartesiske koordinater skulle bli brukt senere i fellestesten, se seksjon 7.1.



Figur 6.5: Kartlegging ved bruk av IR-sensorer

Først ble LIDAR-roboten sendt gjennom banen alene, der den kun brukte IR-sensorer til å kartlegge banen. Denne testen ble helt vellykket og ut fra figur 6.5 kan vi se at alle vegger ble funnet. Vi kan også se at punktskya generert av veggene er rette og at det hvite området roboten har lov å bevege seg på, er tilsvarende rett og jevnt overalt. Denne testen gikk bedre enn forventet og selv om roboten var nødt å bruke rundt 3 minutter på å kjøre fram og tilbake for å kartlegge hele labyrinten, så oppstod det ingen problemer ut fra dette.

Etter dette ble det gjennomført en test der roboten bruke LIDAR-sensoren til å kartlegge labyrinten. Som en ser ut fra resultatet vist på figur 6.6, ser en at kartet som ble generert her, er dårligere enn det som ble generert ut fra IR-sensorene. En ser at veggene ofte er skjeve og har hakk, noe som fører til at det hvite området roboten har lov å bevege seg på er ujevnt. Kartleggingen med LIDAR-sensoren ser også ut til å ha generert mere støy enn kartleggingen med IR-sensorer gjorde. Denne støyen består av enkelte punkter som ligger plassert hvor det ikke er vegger.



Figur 6.6: Kartlegging ved bruk av LIDAR-sensorer

Under testing virket det som LIDAR-sensoren målte nøyaktige og gode verdier så lenge roboten kjørte framover eller tårnet på roboten roterte. LIDAR-sensoren ble derimot mer unøyaktig når hele roboten roterte. Dette kan komme av unøyaktigheten diskutert under seksjon 4.1. Når roboten roterer legger den den estimerte rotasjonsvinkelen til den vinkelen på kartet hvor sensoren kartlegger. Ettersom sensoren måler store lengder under en slik rotasjon, og siden rotasjonen skjer fort, fører dette til at LIDAR-sensoren genererer noen sprette målinger som er mer unøyaktige enn resten av målingene. Disse unøyaktige målingene lager støy for roboten og gjør det vanskeligere å kartlegge banen skikkelig. Denne feilen kan trolig bli løst ved å deaktivere kartlegging på LIDAR-sensoren når roboten roterer. For at det skal være mulig å gjennomføre målinger rundt robåten, må det da implementere flere søk hvor tårnet på roboten roterer mens roboten står i ro.

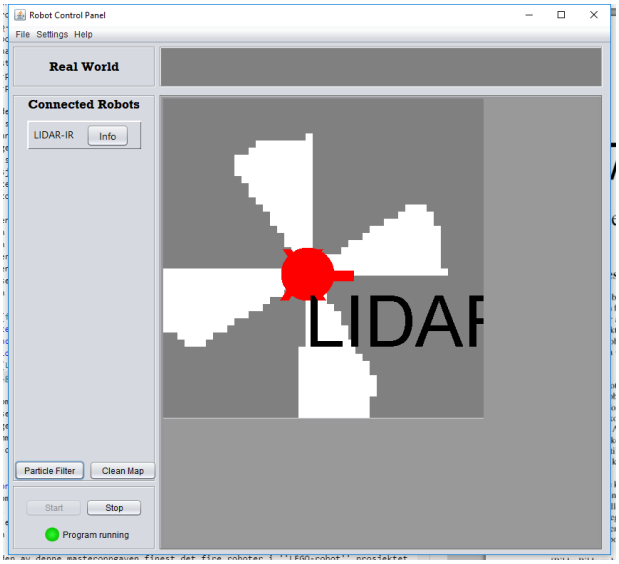
Ut fra denne testen med LIDAR-sensoren kom det også fram at koordinatene serveren regnet fram, og som roboten skulle bevege seg til, ikke var optimaliserte for rekkeviddet til LIDAR-sensoren. Flere ganger gjennom testen fikk roboten beskjed om å kjøre til et punkt den allerede hadde kartlagt ved hjelp av LIDAR-sensoren. Dette førte til at kartlegging av hele banen med LIDAR-sensor tok omtrent like lang tid som kartlegging med IR-sensor ettersom roboten brukte unødvendig lang tid på å kjøre gjennom deler av banen den allerede hadde kartlagt ved hjelp av LIDAR-sensoren. Mer om dette problemet og løsninger finnes under videre arbeid (8.2).

Kapittel 7

Fellestest

7.1 Kartesiske koordinater

LIDAR-roboten ble opprinnelig programmert i polarkoordinater. Dette ble gjort ettersom Arduino-roboten fra Andersen og Rødseth sitt prosjekt var programmert i polarkoordinater og prosjektet for å oversette robotens navigasjon til kartesiske koordinater gikk parallelt med dette prosjektet. Dette resulterte i at da roboten var ferdigstilt brukte den polarkoordinater, mens Arduino-roboten fra Nilsen sitt master-prosjekt [28] og NXT-roboten fra Geir sitt master-prosjekt [29] brukte kartesiske koordinater.



Figur 7.1: Start søk på kartesisk-server

Siden denne roboten brukte polarkoordinater, ville den ikke kunne kjøre på samme server som resten av robotene, og de andre robotene kunne ikke kjøres på samme, server som denne roboten. For å gjøre det mulig å kjøre alle robotene sammen ble det da gjort endringer på navigasjons-koden til LIDAR-roboten. Ettersom både LIDAR-roboten og Nilsen sin robot bruker et Arduino-kort med "*Atmega 2560*" til å styre roboten var navigasjons koden våre ganske like. Ved å bruke navigasjonsdelen fra koden hans, erstatte navnet på variabler fra $\theta.r$ til $x.y$ og endre spesifikasjonene for motorene, var det mulig gjøre roboten kompatibel med kartesiske koordinater.

Serveren med de kartesiske koordinatene viste seg å ha noen forandringer i forhold til den med polarkoordinater som var brukt før. Denne serveren lager mindre avstander roboten skal reise til mellom hver vellykka navigasjon enn den gamle serveren. Dette resulterer i at roboten beveger seg saktere, men mer presist. En annen forskjell er at serveren gir beskjed til roboten at den skal bytte til kjøre-modus fortere enn den gamle serveren. Dette resulterer i at roboten ikke rekker å kartlegge hele 360° før den begynner å kjøre.

Ettersom det nå ble benyttet en ny server for kartlegging, måtte endringene for å få LIDAR-sensoren til å funger også implementeres på denne serveren. Dette er de samme endringene som ble gjort til serveren under seksjon 5.5. Ved samme endringer fungerte kartlegging med LIDAR-sensor like bra på denne serveren som på den gamle.

7.2 Fellestest

Ettersom en del av dette prosjektet var å lage roboter som kan kartlegge et område sammen, ble det viktig å teste hvor bra dette fungerte i praksis. Det ble derfor utført en test med alle de eksisterende robotene i "*LEGO-robot*" prosjektet på B333, der den store testbanene fra seksjon 6.2 ble brukt.

Ved enden av denne masteroppgaven finest det fire roboter i "*LEGO-robot*" prosjektet. Disse robotene er:

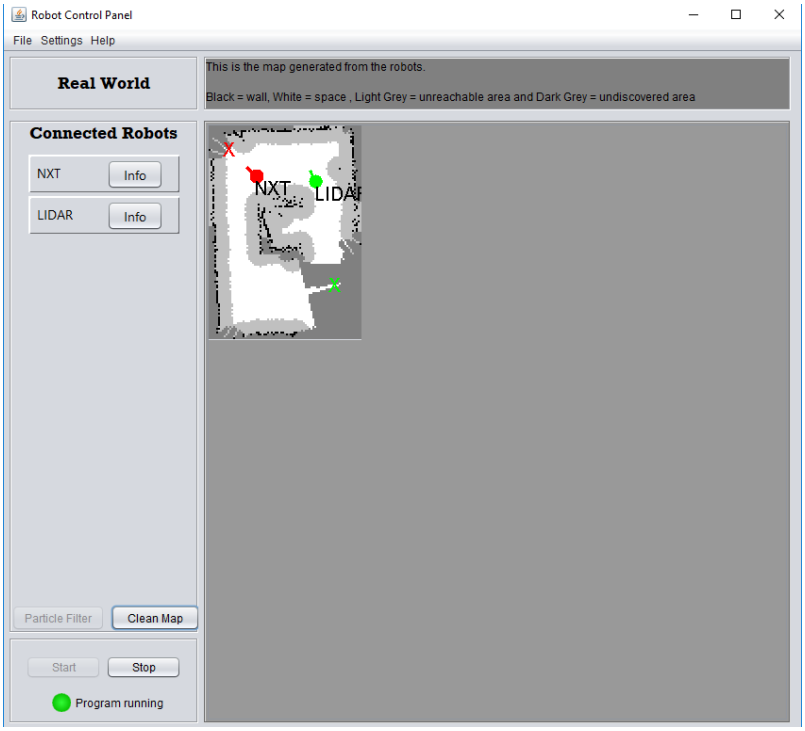
- LIDAR-robot (Fra denne masteren)
- ARD-robot (Fra Nilssen sin master) [28]
- NXT (Sist jobbet med under Eikeland sin master) [29]
- Arduino-robot (Fra Andersen og Rødseth sin master) [4]

På tidspunktet robotene skulle testes var Arduino-robot ødelagt og kunne derfor ikke være med på testen. Dermed var det bare tre av robotene som kunne testes sammen i banen.

Da robotene først skulle koples opp mot serveren oppsto en underlig feil. Både LIDAR-roboten og NXT roboten kunne kople seg opp på samme server, men ARD-roboten og LIDAR-roboten kunne ikke kople seg opp på serveren samtidig. Dette var først antatt å være en feil på Bluetooth-dongelen på roboten som gjorde at server-dongelen ikke klarte å skille disse to robotene fra hverandre. Det ble etter mye feilsøking avdekt å skyldes adressen til robotene. Serveren skiller robotene fra hverandre etter en 8-bits heltalls-variabel kalt adresse som settes i "*network.c*". Denne variablene var blitt satt til samme verdi på både LIDAR-roboten og ARD-roboten og serveren klarte derfor ikke å skille disse robotene fra hverandre.

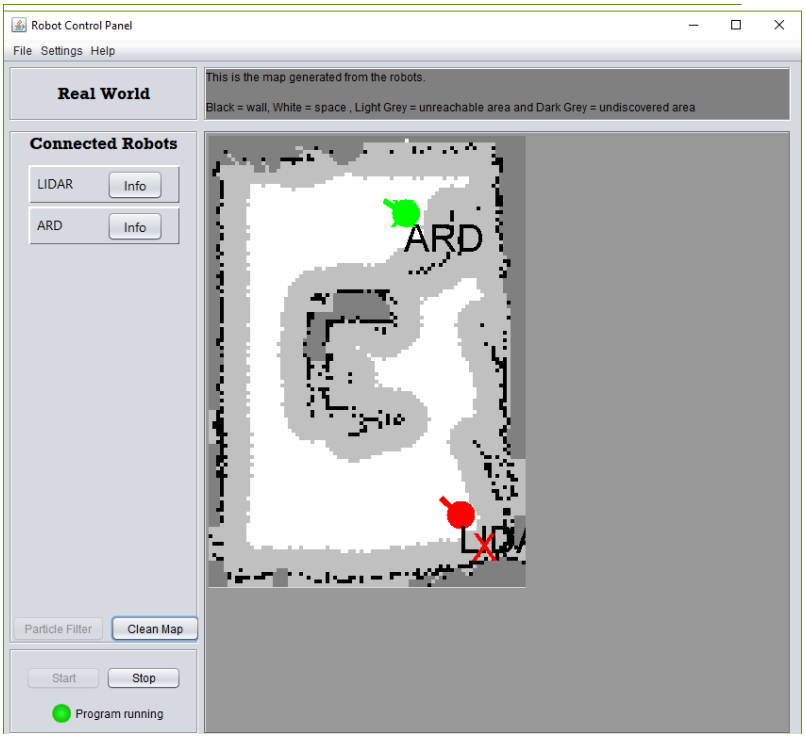
Da denne feilen ble retta, fikk alle robotene kontakt med serveren og kunne kjøres samtidig. Det viste seg etter litt testing at det nye kartesiske koordinatsystemet hadde en svakhet som polarkoordinater ikke hadde. Serveren som bruker kartesiske koordinater kan ikke starte roboten i andre koordinater enn 0.0. Om andre startkoordinater settes, tror roboten at den starter i 0.0 og prøver å kjøre mot de gitte koordinatene istedenfor å oppdatere startpunktet sitt. Dette gikk igjen på alle robotene. Når dette ble testet i polarkoordinater kunne startposisjon settes, og roboten ble flyttet til rett posisjon i kartet uten å bevege på seg i virkeligheten.

Ettersom ingen av de andre robotene kunne kjøres i polarkoordinater, måtte testene fortsette ved at vi startet alle robotene fra samme posisjon, altså posisjon 0.0. Dette gjorde at en robot måtte startes i 0.0 og kjøre 80 cm før neste robot kunne plasseres i 0.0 og kobles til serveren. Når robotene ble startet på denne måten, gjorde det at kartlegging ble litt redundant med 3 roboter ettersom 2 roboter kunne kjøre hver sin vei og på den måten kartlegge hele banen, mens den tredje roboten måtte kjøre samme vei som en annen robot hadde kjørt tidligere.



Figur 7.2: Kartlegging under felles test

Gjennom testen klarte LIDAR-roboten og ARD-roboten helt fint å kartlegge banen sammen, men da NXT-roboten ble koplet til resten, førte dette til at alle robotene mistet kontakt med serveren. Det viste seg etter litt testing at den samme feilen oppstod hver gang NXT-roboten skulle kjøre med en vilkårlig annen robot. Robotene kan enkelt skrues av og på for å få kontakt med serveren igjen, men må plasseres i posisjon 0.0 om de skal kartlegge korrekt videre. Ettersom denne feilen kun oppstod når NXT-roboten kjører sammen med en annen robot, så kan en anta at dette er et problem med NXT-roboten. Siden NXT-roboten ikke er en del av denne oppgaven, ble feilsøking av den ikke prioritert.



Figur 7.3: LIDAR problem under felles test

Selv om resultatet fra LIDAR-testen utført på den store banen, seksjon 6.3, var dårligere enn ønsket ble det gjort forsøk på å kjøre LIDAR-roboten med en LIDAR-sensor sammen med ARD-roboten. Dette avdekket nok et problem med LIDAR-sensoren som ikke var vurdert tidligere. Når robotene kjører sammen prøver serveren å lage mål robotene skal kjøre mot, der de ikke vil komme nærmere værandre enn 40 cm. Dette gjør at sensorene på robotene ikke vil detektere hverandre som hinder i banen. Som vist på figur 7.3 ser vi at LIDAR-sensoren, som har et rekkevidde på 2m i dette oppsettet, detekterte ARD-roboten. ARD-roboten ble så merket som et hinder i banen. Dette vil bli et større problem i en mer åpen bane hvor LIDAR-sensoren har lettere for å se andre roboter.

Kapittel 8

Diskusjon

8.1 Resultat

Ved enden av masterprosjektet er målene beskrevet under oppgavebeskrivelse nådd. Det nye kretskortet er printet, loddet, montert og alle feilene med det er rettet, slik at roboten er operativ. Alle portene som ble byttet på kretskortet er omprogrammert i koden til roboten, og drivere til de komponentene som er nye er kodet. Roboten er så blitt testet med IR-sensorer og feil som er avdekket under kartlegging er rettet, slik at roboten lykkes i å kartlegge en testbane.

Det er også designet et feste til LIDAR-sensoren slik at den kan brukes sammen med resten av sensorene på roboten. To fester ble da laget, et hvor sensoren erstatter den ene IR-sensoren og et hvor LIDAR-sensoren erstatter hele IR-tårnet. En driver er så laget for LIDAR-sensoren som gjør det mulig å bruke sensoren sammen med resten av C-koden til roboten. For å få LIDAR-sensoren til å virke er det også gjort endringer på serveren som styrer roboten. Disse er gjort på en måte som gjør at serveren fungerer som normalt for roboter uten LIDAR-sensor.

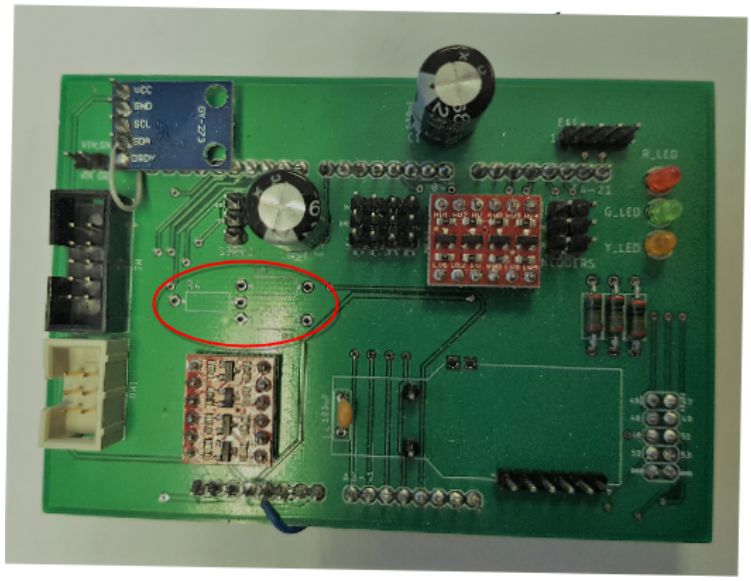
Gjennom tester hvor robotens prestasjon er blitt testet, er forskjellen på IR- og LIDAR-sensor blitt undersøkt. Det ble her avdekket at selv om LIDAR-sensoren fungerer sammen med resten av roboten, så lager IR-sensorene et mer presist kart over omgivelsene. Denne forskjellen blir større dess større kart roboten skal navigere gjennom. Dette skyldes komponenter som kan byttes på roboten og kalibrering som kan forbedres. Navigeringen serveren gir til roboten er heller ikke optimale når roboten bruker en LIDAR-sensor over IR-sensor.

Etter en fellestest, hvor flere roboter skulle kartlegge et rom sammen, viste det seg at når roboten bruker IR-sensoren fungere den godt sammen med andre roboter. Robotene klarer å kartlegge et område sammen og unngår å bevege seg i samme retning, om mulig. Det ble funnet en svakhet med systemet der hver robot må startes i samme posisjon, men kartlegging er fortsatt mulig tross dette problemet.

8.2 Videre arbeid

Digital batterinivå

En funksjon som behøves, men som ikke er implementert på de Arduino-baserte robotene, er en digital oversikt over batterinivået på roboten. Hvis det blir mulig å vise batteri-informasjon på roboten eller på serveren, så vil dette være de første stegene mot å implementere automatisk lading på roboten. Det vil også gjøre det lettere å bruke robotene når en kjenner ladingen på batteriet.



Figur 8.1: Spenningsdeler på kretskortet

På kretskortet fra seksjon 5.3 er det lagt til rette for å måle spenningen på batteriet. Her går det en direkte kobling fra batteriet til en spenningsdeler. Denne spenningsdeleren består av tre motstander hvilket gjør at bare 1/3 del av spenningen fra batteriet går videre til Arduino-kortet. Spenningsdeleren blir brukt ettersom porter på Arduino kortet ikke bør overstige 5V og spenningen fra batteriet er på 11.1V. Det er så tenkt å bruke en analoginngang på Arduino-kortet til å måle spenningen som blir gitt.

Ettersom denne funksjonen ikke er implementert ennå så er ikke motstandene som er nødvendig til spenningsdelingen loddet på kortet. Dermed er det et brudd fra spenningskilden og til analoginngangen A4 på Arduino-kortet. Denne inngangen A4 samt A5, A6 og A7 er de fire portene som brukes om en skal ta i bruk JTAG på ATmega 2560.

Autonom lading

Om en digital batterioversikt blir implementert, så er det et neste steg å implementere autonom lading av roboten. Her tenkes det at roboten skal enten ved lavt batterinivå, eller på kommando, finne veien tilbake til dokkingstasjonen og lades der. Både ladetårnet og dokkingstasjonen til robotene er laget under Strande sin masteroppgave [17].

Dette krever at bryteren som kobler ut ladetårnet på roboten blir gjort digital. Bryteren kan da byttes ut med et relé som kobles inn/ut ved en puls fra Arduino-kortet. Da kan Arduino kortet koble til tårnene etter roboten har navigert tilbake til dokkingstasjonen og kobles ut automatisk etter roboten er ladet helt opp.

I tillegg må det lages en funksjon som gjør det mulig for roboten å navigere til dokkingstasjonen. Roboten må da rygge inn i stasjonene slik at ladetårnet som er plassert bakerst på roboten kommer inn til kontaktflata på dokkingstasjonen. Dette var mulig på den gamle serveren med polarkoordinater ettersom roboten der kan kjøre i negativ retning, men ikke i kartesiske koordinater hvor roboten alltid snur seg mot målet den ønsker å kjøre til.

Rotterende tårn under kjøring

Når roboten kjører roterer den ikke på IR-tårnet. Dette er helt likt for alle de Arduino-basserte robotene, men NXT roboten har implementert denne funksjonen. Siden de Arduino-basserte robotene bare stiller tårnet til startposisjon før de begynner å kjøre, gjør det at de kun kan detektere hinder som står like foran dem. Dette betyr at om et hinder står i vegen på siden av roboten, vil denne ikke bli detektert før roboten treffer den. På serveren er dette prøvd å forebygges ved å gi robotene små avstander å traversere gjennom ikke-kartlagt terreng, men muligheter for en kollisjon er der fortsatt.

360° LIDAR-scann

For å få best nytte ut av LIDAR-sensorn kan det være nyttig å gjennomføre hele kartleggingssøket med den. Med tanke på dette ble det laget et feste til roboten, men ettersom servoen som ble brukt til å rotere tårnet bare kunne rotere 180° ble dette ikke gjort noe mer med.

Det er to måter dette kan bli implementert på. En kan enten kjøpe en ny servo, som kan rotere 360°, eller så kan en anskaffe en ekstra LIDAR-sensor som blir plassert 180° på den første sensoren. Dette vil gjøre at roboten kan kartlegge mye mer av banen med startsøket. Om dette blir implementert, må det også gjøres endringer i hvordan serveren sender koordinater til roboten. Da må serveren prioritere å gi roboten koordinater til en lokasjon hvor et nytt 360° søk vil avdekke mest nytt område på kartet.

Unøyaktighet med LIDAR

Under testen i seksjon 6.3 ble det observert at LIDAR-sensoren kunne bli unøyaktig mens roboten roterte. Dette skyldes trolig unøyaktigheter med robotens estimerte posisjon mens den roterer. Siden LIDAR-sensoren har så lang rekkevidde blir ikke disse verdiene oppdatert etter hvert som roboten kjører. Dette gjør at det oppstår støy på kartet som gjør resten av kartlegginga til roboten unøyaktig.

En løsning på dette kan være å ikke ta målinger med LIDAR-sensoren mens roboten roterer. Da må en heller rotere tårnet på roboten for å oppdatere kartet etter de nye omgivelsene. Tårnet på roboten roterer saktere og det er lettere å kontrollere hastigheten slik at avstanden mellom hver måling blir mindre. Når tårnet på roboten rotere vet roboten også hvor mange grader den har rotert og slipper derfor å estimere seg fram til en verdi som kan være feil.

En annen løysing kan være å forbedre estimata av rotasjonen roboten gjennomfører. Slik roboten er nå så er disse nøyaktige nok til at roboten klarer å kartlegge et lukket område, men sammen med LIDAR-sensoren blir dette unøyaktig. En forbedring her kan være å bytte ut kompasset på roboten, skjerme kompasset mer for støy, eller å bruke mer presise komponenter til å måle rotasjon.

Startkoordinater

Under felles testen som er beskrevet under seksjon 7.2 ble det oppdaget at alle robotene som kjøres med kartesiske koordinater ikke får oppdatert startkoordinatene sine. Dette ser ut til å være et problem under initialiseringen mellom serveren og roboten. Istedenfor at koordinatene som blir satt i serveren blir gjort til en ny startkoordinater, så virker det som det skjer en glipp i kommunikasjonen, og roboten blir heller bedt om å kjøre til disse koordinatene.

Dette er ikke noe problem når bare en robot skal kjøre, men når flere roboter skal samarbeide for å kartlegge et område blir dette fort et stort problem. Det krever endringer på både serveren og roboten for å ordne denne feilen.

Optimaliserings algoritme

Da LIDAR-sensoren ble testet på den store testbanen, viste det seg at algoritmen, som regner ut nye koordinater robotens skal kjøre til, ikke er godt egnet sammen med LIDAR-sensoren. Roboten hadde da tendenser til å kjøre gjennom områder sensoren allerede hadde kartlagt. Dette er en endring som må gjøres i serveren, men må kun gjøres for roboter som bruker LIDAR-sensorer. Det kan da være en god løsning å skille mellom roboter som bruker LIDAR-sensorer og de som bruker IR-senorer ved å bruke navnet til roboten.

Bibliografi

[1] Amsen, J. Ø. (Juli 2017) *Improving Navigation and Mapping with Arduino robot*. Masteroppgave. NTNU.

[2] SHARP (2014) *Datablad for IR-sensorene*. [Online] URL: `https://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf`
(Sist besøkt: 31.05.2018)

[3] Jensen, S. M. K. (Desember 2017) *Arduino-robot med LIDAR-sensor*. Prosjektoppgave. NTNU.

[4] Andersen, T. og Rødseth, M. (Juni 2016) *System for Self-Navigating Autonomous Robots*. Masteroppgave. NTNU.

[5] Andersen, T. og Rødseth, M. (Desember 2017) *Arduino SLAM robot*. Prosjektoppgave. NTNU.

[6] Garmin (2018) *Bibliotek til LIDAR-sensoren*. [Online] URL: `https://github.com/garmin/LIDARLite_Arduino_Library`
(Sist besøkt: 31.05.2018)

[7] Seeed Studio (2018) *PCB/PCBA*. [Online] URL: `https://www.seeedstudio.io/fusion_pcb.html?gclid=CjwKCAjwur7YBRA_EiwASXqIHDWWv5EM2hxziVDtvgk1bjcl6dxUOqNMxaB8P1U-pBGVMtH2eZC35BoCpSwQABwE`
(Sist besøkt: 31.05.2018)

[8] Sparkfun (2017) *Hookup Guide for LIDAR lite v3*. [Online] URL: `https://learn.sparkfun.com/tutorials/lidar-lite-v3-hookup-guide?_ga=2.16203710.945091813.1527751433-401373336.1504450833`
(Sist besøkt: 31.05.2018)

[9] Arduino (2018) *PWM*. [Online] URL: `https://www.arduino.cc/en/Tutorial/PWM`
(Sist besøkt: 31.05.2018)

[10] Atmel (2014) *ATmega2560 datasheet*. [Online] URL: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf
(Sist besøkt: 31.05.2018)

[11] DBServo (2014) *S05NF servo*. [Online] URL: https://cdn.sparkfun.com/datasheets/Robotics/S05NF%20STD.pdf
(Sist besøkt: 31.05.2018)

[12] Hitec (2009) *HS-5925MG servo*. [Online] URL: https://servodatabase.com/servo/hitec/hs-5925mg
(Sist besøkt: 31.05.2018)

[13] STMicroelectronics (2015) *LSM6DS3 IMU*. [Online] URL: https://cdn.sparkfun.com/assets/learn_tutorials/4/1/6/DM00133076.pdf
(Sist besøkt: 31.05.2018)

[14] Sparkfun (2014) *Arduino Mega 2560*. [Online] URL: https://www.sparkfun.com/products/11061
(Sist besøkt: 31.05.2018)

[15] Cana Kit (2015) *L298 H-Bridge*. [Online] URL: https://www.canakit.com/Media/Manuals/UK1122.pdf
(Sist besøkt: 31.05.2018)

[16] Elfa Distrelec (2014) *Li-ion-batteri*. [Online] URL: https://www.elfadistrelec.no/no/li-ion-batteri-11-4600-mah-hy-line-h2b181/p/16901653
(Sist besøkt: 31.05.2018)

[17] Strande, L. (Juni 2017) *Autonom retur og dokking av AVR robot*. Masteroppgave. NTNU.

[18] Honeywell (2013) *HMC5883L Datasheet*. [Online] URL: https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf
(Sist besøkt: 31.05.2018)

[19] DAGU (2014) *Encoder Kit*. [Online] URL: https://cdn.sparkfun.com/datasheets/Robotics/multi-chassis%20encoder001.pdf
(Sist besøkt: 31.05.2018)

[20] DAGU (2018) *120:1 gir motor*. [Online] URL: https://www.exp-tech.de/en/7045/dagu-dc-gear-motor-paar-90-degree-shaft-120-1
(Sist besøkt: 31.05.2018)

[21] FreeRTOS (2018) *The FreeRTOS Kernel*. [Online] URL: https://www.freertos.org/
(Sist besøkt: 31.05.2018)

[22] Ese, E. (Juni 2016) *Sanntidsprogrammering på samarbeidande mobil-robotar*. Masteroppgave. NTNU.

[23] GARMIN (2017) *Lidar Lite v3 Operation Manual and Technical Specifications*. [Online] URL: http://static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf (Sist besøkt: 31.05.2018)

[24] Statens strålevern (2014) *Laserklasser*. [Online] URL: https://www.nrpa.no/fakta/90813/laserklasser (Sist besøkt: 31.05.2018)

[25] Sparkfun (2017) *LIDAR-Lite v3*. [Online] URL: https://www.sparkfun.com/products/14032 (Sist besøkt: 31.05.2018)

[26] Code-N-Logic (2015) *I2C communication using ATMEGA16*. [Online] URL: https://www.youtube.com/watch?v=si5HhCWp6qI (Sist besøkt: 31.05.2018)

[27] DAGU (2018) *80mm Wheel*. [Online] URL: http://www.dagurobot.com/80mm_Wheel?search=wheels&category_id=0 (Sist besøkt: 31.05.2018)

[28] Nilsen, S. (Juni 2018) *Navigation improvement on SLAM robot*. Masteroppgave. NTNU.

[29] Eikeland, G. (Mars 2018) *Kartlegging ved NXT-robot*. Masteroppgave. NTNU.

Kapittel 9

Vedlegg

(a) Vedlegg 1 - Bluetooth Dongel

(b) Vedlegg 2 - PCB- og Gerber-filer

(c) Vedlegg 3 - Robot koder

(d) Vedlegg 4 - Server koder

(e) Vedlegg 5 - Datablad

(f) Vedlegg 6 - Viktige Rapporter

(g) Vedlegg 7 - Koblingskjema

(h) Vedlegg 8 - Test-koder

(i) Vedlegg 9 - Videoer og Bilder