LIDARINO:

EN REISE MOT SELVKJØRING

ROBDIG 2025

Thomas E. Ortynski

Karl Magnus Myrvold

Erik Magnussen

Odd Levi Skare

Forord

Denne rapporten er utarbeidet av fire avgangsstudenter ved Fagskulen Vestland, studieretningen for robotteknologi og digital produksjon i industrien. Rapporten dokumenterer utviklingen av en autonom minirobot kalt «Lidarino».

Prosjektet er tildelt gruppen av oppdragsgiver Gunnar Storebø, faglærer ved Fagskulen Vestland. Oppgaven omfatter montering av en minirobot basert på et ferdig chassis, integrering av nødvendige komponenter ved hjelp av egenutviklede holdere, samt programmering for selvkjørende funksjonalitet.

Målet for roboten er å navigere autonomt gjennom en korridor eller et rom og parkere med en nøyaktighet på ±1 cm fra et angitt hjørne (nord, sør, øst eller vest), før den roterte slik at fronten vendte ut.

Vi vil takke oppdragsgiver Gunnar Storebø for muligheten til å få arbeide med et utfordrende og lærerikt prosjekt. Hans veiledning og ekspertise har bidratt til deler av prosessen. Kunnskapen og erfaringene fra dette arbeidet har styrket vår faglige kompetanse og vil være sentral i våre fremtidige prosjekter og karriere.

Videre rettes en stor takk til både familie, venner og bekjente for deres bistand, forståelse og fraværende sosial kontakt gjennom en krevende prosjektperiode.

Sammendrag

Dette hovedprosjektet har hatt som mål å utvikle en autonom minirobot, Lidarino, som kan navigere selvstendig og parkere med høy presisjon ved hjelp av integrerte sensorer og avanserte styringssystemer. Roboten er basert på et ferdig chassis, hvor det har blitt gjennomført nødvendige maskinvarekomponenter som mikrokontrollere, motorer med enkoder, Lidar, kompass og ultralydsensorer. I tillegg har det blitt utviklet programvare for sanntidsprosessering og kontroll av robotens bevegelser.

Gjennom prosjektet har teori og praksis innen robotikk vært i sentrum, elektronikk og programmering. Arbeidet har inkludert design av mekaniske og elektroniske komponenter, utvikling av styringsalgoritmer, feilsøking og optimalisering av systemet. Roboten benytter Omni drive-teknologi for fleksibel bevegelse i alle retninger, styrt ved hjelp av en ESP32-mikrokontroller og en Raspberry Pi 5 for mer avansert databehandling.

For å forbedre robotens navigasjon er det gjennomført sensorfusjon mellom Lidar og ultralydsensorer, samt algoritmer for ruteplanlegging og hinderunngåelse. Det har også blitt utforsket sanntidskartlegging (SLAM) og inverse kinematiske beregninger for best mulig styring av Omni drive-hjulene.

Underveis i prosjektet har det oppstått flere tekniske utfordringer, blant annet knyttet til motorstyring, kommunikasjon mellom komponenter og stabil strømforsyning. Gjennom systematisk feilsøking og testing har det blitt funnet løsninger som sikrer en mer stabil og effektiv robotdrift.

Resultatene viser at Lidarino er i stand til å navigere autonomt og unngå hindringer ved hjelp av sanntidsdata fra sensorene. Robotens parkeringsevne og presisjon har blitt testet og analysert for å vurdere ytelsen. Videre arbeid skulle konsentreres om å optimalisere programvaren ytterligere, samt utforske muligheten for å ta i bruk maskinsyn for mer avansert navigasjon, men tiden strekte ikke til for å få dette gjennomført.

Prosjektet har gitt verdifull erfaring innen robotteknologi, systemutvikling og problemløsning, og de tekniske løsningene som er utviklet kan videreføres og forbedres i fremtidige prosjekter.

Innholdsfortegnelse

1. Problemstilling 7

2. Målsetting 7

3. Teori og prinsipper 8

3.1. Omni-hjul system 9

3.2. Mikrokontroller 10

3.2.1. ESP32-Wroom 32 11

3.2.2. Raspberry Pi 13

3.3. Gyroskop 15

3.4. Akselerometre 16

3.5. Lidar 18

3.6. Motordriver MX1616H 19

3.7. DC-Motor med enkoder 22

3.8. Ultralydsensor 24

3.9. Kompass 27

4. Metode 29

4.1. Git og GitHub 30

4.2. Konstruksjon og oppbygning av nedre del 31

4.3. Drivsystem og motorstyring 32

4.3.1. PWM (Pulse Width Modulation) 33

4.3.2. Generering av PWM-signaler 35

4.3.3. Kiwidrive 35

4.4. IR-Mottaker og fjernkontroll: 38

4.5. PID-regulator 39

4.5.1. Implementering av PID-regulering 42

4.6. Konstruksjon og oppbygging (Øvre del) 45

4.7. Systemintegrasjon 46
4.7.1. Konfigurering av Raspberry Pi 5: Enhetsnavn og LiDAR-stopp 47
4.7.2. Batteripakke 53
4.7.3. Step-down DC-DC omformer 55
4.7.4. Skybasert Logging og Datainnsamling 56
5. Resultat 57
6. Drøfting 61
6.1. Inndeling av prosjektet 61
6.2. Nedre del av roboten 61
6.2.1. Oppbygning og plassering av komponenter 62
6.3. Den øvre delen av roboten 65
6.4. Kommunikasjon mellom nedre og øvre del 68
6.5. Avsluttende refleksjon 71
7. Konklusjon 72
8. Litteraturliste 74
8.1. Definisjoner og forkortelser 77
8.2. Figurliste 80
9. Vedlegg 1 Prosjektets GitHub 82
10. Vedlegg 2 Prosjektets Google drive 83
11. Vedlegg 3 koblingsskjema til Raspberry og ESP 32 84

Presentasjon av medlemmer

Thomas Eiksund Ortynski:

Prosjektleder:

Fagbrev som elektriker og er ansatt hos FLM Elektro. Jobber som montør på små og store prosjekter innen industri, bolig og næring i privat og offentlig sektor

Karl Magnus Myrvold:

Teknisk Ansvarlig:

Fagbrev som sveiser og ansatt som platearbeider. Har ansvar for CNC-maskin og reiser offshore for serviceoppdrag på plattformer og fartøy.

Erik Magnussen:

Prosjektmedarbeider.

Fagbrev som serviceelektroniker og svakstrømsmontør. Jobber i dag som selvstendig næringsdrivende i firma kalt Smart Lys AS, som tar oppdrag innen svakstrøm.

Odd Levi Skare:

Prosjektmedarbeider.

Fagbrev som serviceelektroniker og ansatt i Forsvarets logistikkorganisasjon som systemtekniker for sjøforsvarets fartøyer

Innledning

Robotteknologi har de siste årene sett en kraftig vekst innen både forskning og kommersiell bruk. Fra å utføre repeterende oppgaver innen industrien, har roboter utviklet seg til å navigere autonomt i komplekse miljøer og er i dag blitt en viktig del av det moderne samfunnet. Denne utviklingen har ført til økt etterspørsel etter kompetanse innen robotdesign, programmering og systemintegrasjon.

Prosjektet omfatter utviklet en autonom minirobot basert på kunnskap innen elektronikk, mekanikk og programmering. Arbeidet har gitt verdifull erfaring med bruk av teoretiske prinsipper i praksis og har bidratt til økt innsikt i programmering, sensorteknologi og robotikk.

Rapporten dokumenterer hele utviklingen av den autonome miniroboten, fra design og gjennomføring til testing. Det blir redegjort for de tekniske løsningene som har blitt valgt, samt en analyse av robotens ytelse basert på disse valgene.

Prosjektet har vært både utfordrende og lærerikt, og har krevd tett samarbeid og kreativ problemløsning fra alle medlemmer av teamet.

1. Problemstilling

Målet med dette prosjektet er å bygge og programmere en autonom robot basert på en forhåndsdefinert plattform. Roboten skal kunne navigere selvstendig i et kontrollert miljø, identifisere og unngå hindringer samt parkere med en nøyaktighet på ±1 cm fra et angitt hjørne.

For å oppnå dette må det brukes en rekke tekniske komponenter, inkludert mikrokontrollere, motorer med enkoder, Lidar, kompass og ultralydsensorer. I tillegg må det utvikles programvare for presis styring og effektiv samhandling mellom komponentene.

2. Målsetting

Målet med dette prosjektet er å undersøke mulighetene for å utvikle et karosseri fra en robot til å kunne bli en selvkjørende minirobot som kan navigere autonomt, unngå hindringer ved hjelp av integrerte sensorer som Lidar, ultralydsensorer og kompass som illustrert i figur 1. Lidar (Light Detection and Ranging) er en sensor som sender ut modulert laserpulser som måler tiden og vinkelen til den reflekterte strålen for å bygge opp et todimensjonalt eller tredimensjonalt punktkart av omgivelsene.

Roboten skal kunne parkere med en nøyaktighet på ±1cm fra valgt hjørne av et rom eller en korridor, og demonstrere pålitelig samhandling mellom maskinvare og programvare. Dette inkluderer mikrokontrollere, motorer med enkoder og sensorteknologi.

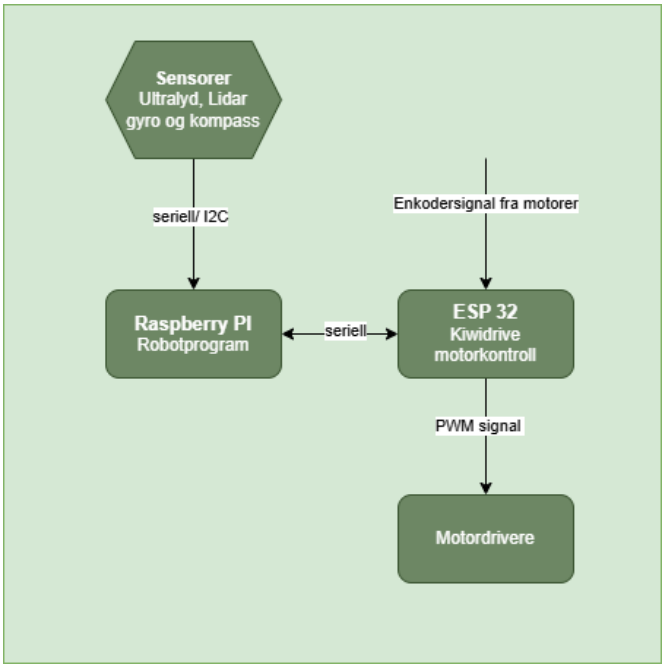
Gjennom prosjektet forventes det at eksisterende kunnskap innen programmering, sensorteknologi og robotikk videreutvikles. Dersom tiden strekker til, vil det bli sett på muligheten for å ta i bruk ytterlige teknologier som maskinsyn og sanntids kartlegging for å forbedre robotens autonomi og funksjonalitet.

3. Teori og prinsipper

Valget av komponenter har vært preget av omfattende drøftinger og evalueringer, basert på både tidligere erfaringer og kunnskap. Flere alternativer har blitt vurdert, og det er benyttet tidligere kjennskap og praktiske erfaringer fra lignende prosjekter for å sikre de beste tekniske og økonomiske valgene.

For å legge til rette for en solid og fremtidsrettet løsning har er det lagt vekt på flere nøkkelaspekter. Teknisk kompatibilitet er sørget for ved å se på hvordan de valgte komponentene passer inn med eksisterende systemer og teknologi, samt deres potensial for oppgradering og utvidelse. I tillegg er det, basert på tidligere erfaringer, valgt komponenter som har vist seg å være stabile og effektive, noe som gir trygghet for langsiktig drift. Videre er det også tatt høyde for innovasjon og skalering, slik at løsningene kan oppgraderes eller utvides etter hvert som kravene endres.

Sammenfattende er valget basert på en grundig vurdering av tekniske kriterier, praktiske erfaringer og en helhetlig tilnærming til prosjektets målsettinger. Dette gir ikke bare best mulig løsning i dag, men legger også grunnlaget for videre innovasjon.



Figur 1 Prinsippskisse over roboten

3.1. Omni-hjul system

Omni-hjul, eller polyhjul, er hjul med små «ruller» rundt omkretsen som står vinkelrett på dreieretningen. Prinsippet bak disse hjulene er illustrert i Figur 3. Effekten er at hjulet kan drives forover, men også gli sidelengs. Disse hjulene brukes ofte i holonomiske kjøresystemer, som kjennetegnes ved at de kan bevege seg fritt i alle retninger uavhengig av tidligere bevegelse. Holonomisk bevegelse betyr at alle begrensninger i systemets bevegelse kan uttrykkes som matematiske relasjoner mellom posisjonsvariabler, som gjør at systemet kan bevege seg direkte i ønsket retning uten å måtte endre orientering først. Et eksempel er en plattform som kan bevege seg rett sidelengs uten å måtte endre retning, i motsetning til en vanlig bil som har begrensninger på hvordan den kan snu og kjøre (Wikipedia, u.d.).

Det første omni-direksjonelle hjulet som ble patentert, er designet av J. Grabowiecki i 1919. Hjuldesignet blir fortsatt ofte brukt i dag, og det består av et hovedhjul med tverrgående ruller rundt hovedhjulet.

En plattform som bruker tre Omni-hjul i en triangulær konfigurasjon kalles generelt Kiwi Drive. Killough-plattformen er lik; oppkalt etter Stephen Killoughs arbeid med omni-direksjonale plattformen ved Oak Ridge National Laboratory. Killoughs design fra 1994 brukte par med hjul montert i bur i rette vinkler til hverandre og oppnådde dermed holonomisk bevegelse uten å bruke Omni-hjul (Wikipedia, 2025)

Figur 2 Viser prinsippet av et omni-hjul. Illustrasjon laget av prosjektgruppen.

3.2. Mikrokontroller

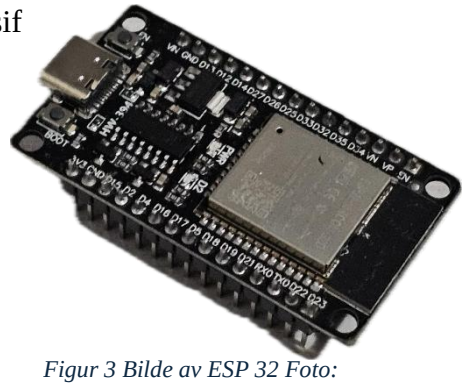
En mikrokontroller som ESP WROOM 32, benyttet i dette prosjektet, er en type integrert krets (IC) som fungerer som en kompakt datamaskin på én enkelt brikke. Den inneholder prosessor (CPU), ulike typer minne (som RAM for midlertidige data og Flash-minne for permanent lagring av programvaren), samt inn- og utganger (I/O) for kommunikasjon med øvrige komponenter.

Mikrokontrollere anvendes primært i innebygde systemer der formålet er å overvåke, styre eller regulere funksjoner, ofte i sanntid. Programvaren (koden) lagres i det ikke-flyktige minnet og kjøres automatisk hver gang mikrokontrolleren får spenning. Dette gjøres vanligvis under montering av mikrokontrolleren på det endelige kretskortet den skal inngå i.

De brukes i alt fra små husholdningsapparater til komplekse systemer. Et vanlig eksempel er i bilindustrien, hvor en bil kan inneholde flere hundre mikrokontrollere som styrer funksjoner som motor, sikkerhet, dekktrykk og lading i elbiler (Larsen, 2025).

3.2.1. ESP32-Wroom 32

ESP32-WROOM-32 er en allsidig mikrokontroller utviklet av espressif systems, se figur 4. Den kombinerer både Wi-Fi og bluetooth, både klassisk bluetooth (Classic) og bluetooth Low Energy, (BLE) som gjør den svært godt egnet for applikasjoner innen internet of things (IoT), trådløs kommunikasjon og sanntids datainnsamling.



Figur 3 Bilde av ESP 32 Foto: Prosjektgruppen

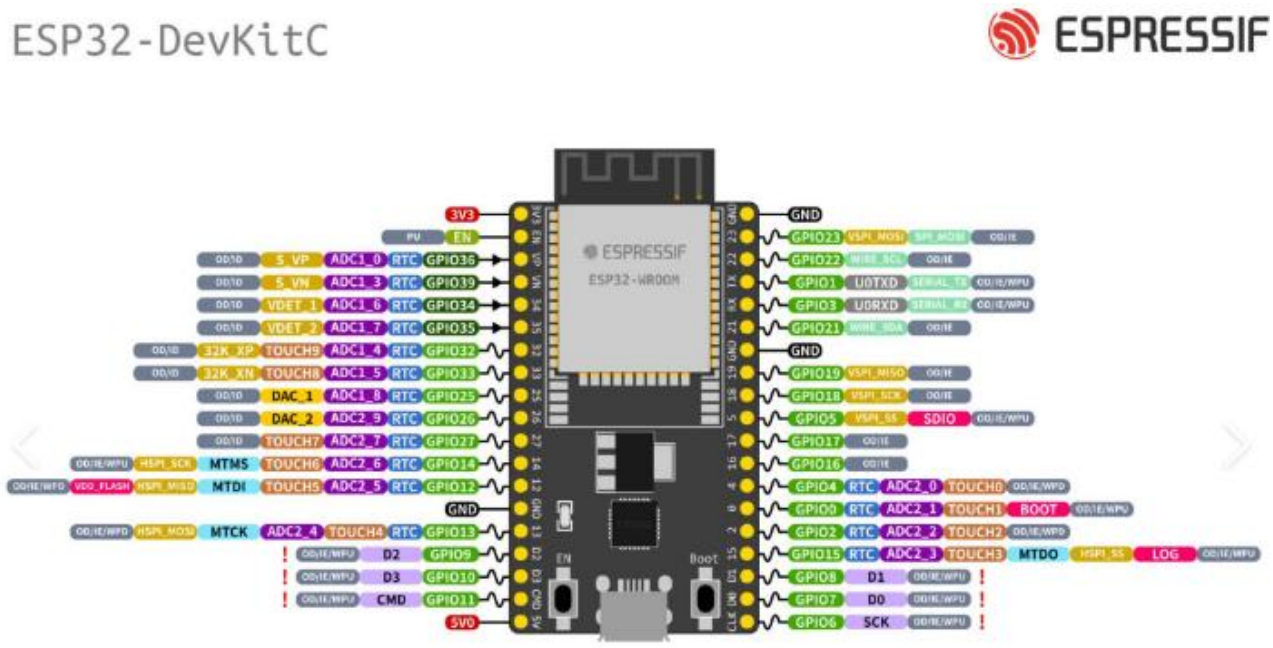
Modulen er basert på ESP32-D9WDQ6 System-on-chip (SoC), som inneholder to 32-biters Tensilica Xtensa Lx6-prosessorer. Disse kan kjøre uavhengig eller parallellt, med en maksimal klokkehastighet på 240 Mhz. ESP32 har 520 KB SRAM og finnes også i varianter med ekstra pseudostatisk RAM (psRAM) for mer krevende oppgaver.

På kommunikasjonsfronten støtter modulen Wi-Fi (802.11 b/g/n) og Bluetooth 4.2, inkludert både klassisk Bluetooth og BLE (Bluetooth Low Energy). Denne doble protokollstøtten gir stor fleksibilitet i enheter som krever pålitelig trådløs kommunikasjon, som smarthusenheter, bærbare sensorer og trådløse styresystemer.

Takket være høy ytelse kombinert med lavt strømforbruk er ESP32 WROOM utbredt i smarthusløsninger, bærbare enheter, industriell automatisering og sanntids datasystemer. Den har også støtte for avanserte funksjoner som stemmegjenkjenning og mediebehandling, noe som utvider bruksområdene ytterligere (Nabto, u.d.).

Tilkoblinger

GPIO'ene (General Purpose Input/Output) har forskjellige funksjoner, som vist i illustrasjonen på figur 5. Ikke alle utgangene kan programmeres fritt da noen av de har dedikerte oppgaver.



Figur 4 Oversikt GPIO hentet fra espressif.com

Brukte GPIO ifra arduino koden

```
// ==================== ENCODER PINS ====================
#define ENCODER_A1_PIN 18
#define ENCODER_B1_PIN 17

#define ENCODER_A2_PIN 16
#define ENCODER_B2_PIN 23

#define ENCODER_A3_PIN 4
#define ENCODER_B3_PIN 19

// ==================== MOTOR PINS ====================
#define MOTOR_1_FORWARD_PIN 14
#define MOTOR_1_BACKWARD_PIN 27

#define MOTOR_2_FORWARD_PIN 25
#define MOTOR_2_BACKWARD_PIN 26

#define MOTOR_3_FORWARD_PIN 33
#define MOTOR_3_BACKWARD_PIN 32

// ==================== IR SENSOR PIN ====================
#define IR_RECEIVE_PIN 13
```

3.2.2. Raspberry Pi

Raspberry Pi er en ettkortdatamaskin utviklet av det britiske selskapet Rasberry Pi Foundation, med det opprinnelige målet å fremme informatikkundervisning. Siden lanseringen i 2012 har den utviklet seg betydelig, og den nyeste modellen, Raspberry Pi 5, representerer et teknologisk sprang fremover, illustrert i figur 6. Den er basert på en mikroprosessor (SoC), og tilbyr forbedret prosessorytelse, økt minnekapasitet og et bredt spekter av tilkoblingsmuligheter. Dette gjør den velegnet for både hobbyprosjekter og industrielle applikasjoner.
Raspberry Pi 5 gir mulighet for å håndtere mer komplekse oppgaver enn mange andre mikrokontrollere. Den er spesielt egnet for prosjekter som krever høy ytelse, flerbrukerstøtte og omfattende I/O-kontroller. Modellen gir mulighet for å kjøre kraftigere operativsystemer som Raspberry Pi OS, og støtte for høyere prosesseringshastigheter og mer avanserte tilkoblinger (Raspberry Pi Foundation, u.d.).

På nettsiden til Raspberry Pi Foundation er følgende spesifikasjoner nevnt:

Maskinvare

- Broadcom BCM2712 quad-core Arm Cortex A76 processor 2.4 GHz
- 8 GB RAM (Random Access Memory)

Tilkoblinger

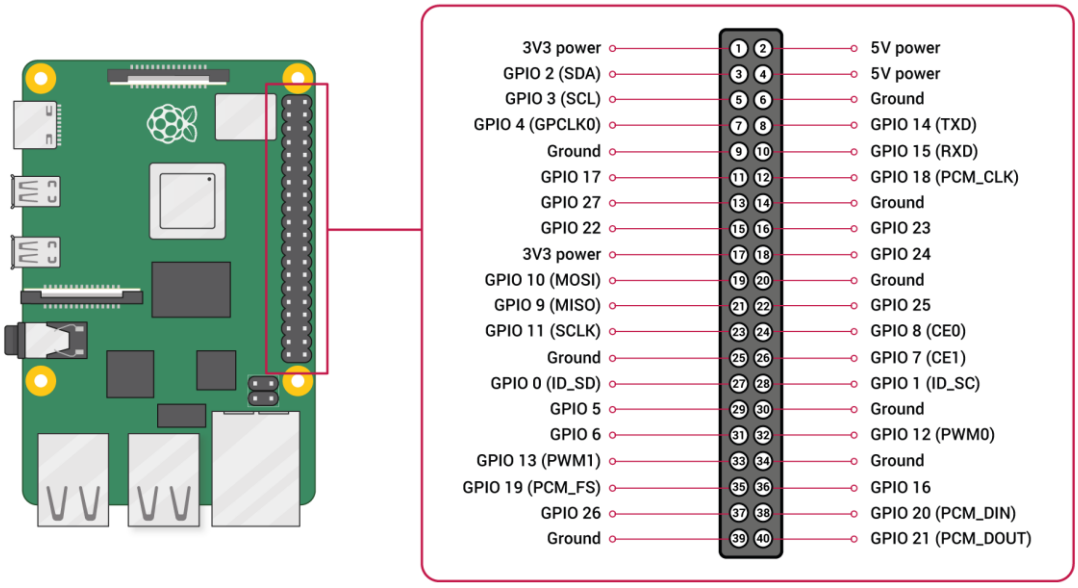
- 40x GPIO – kontakter, som kan konfigrueres og programmeres
- 4x USB A porter
- 1 Gb Ethernet port
- USB C
- 2 HDMI mini utganger

Figur 5 Raspberry Pi 5 Foto: Prosjektgruppen

Lagring

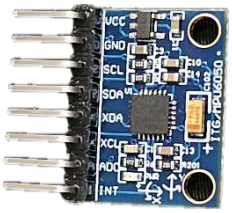
- microSD-kort for OS og lagring, men har også støtte for NVMe SSD-lagring via et PCIe- M.2 grensesnitt, som gir raskere lese- og skrivehastigheter
- Operativsystem (tidligere kjent som Raspbian), basert på Debian Linux. Andre operativsystemer og distribusjoner kan også benyttes etter behov.

Raspberry Pi 5 har støtte for HAT (Hardware Attached on Top)-moduler, som kan kobles til via GPIO-pinnene eller PCIe-kontakten, illustrert i figur 7, som gir bedre fleksibilitet i tilkoblingen av eksterne enheter som sensorer, motorer eller skjermer. Funksjonaliteten er spesielt nyttig i prosjekter som krever integrasjon med et bredt spekter av maskinvare.



Figur 6 Raspberry med GPIO, hentet fra pinout.ai/raspberry-pi-5

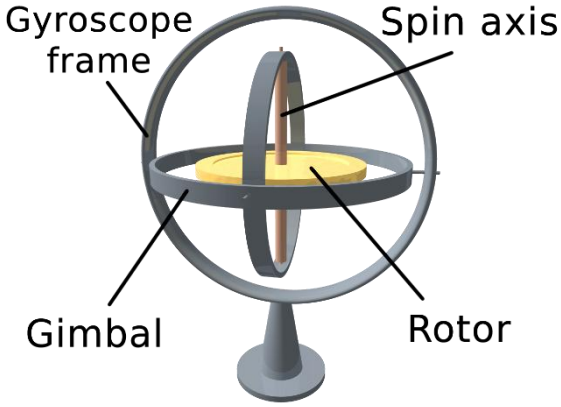
3.3. Gyroskop



Figur 7 Gyroskopet
Foto: Prosjektgruppen

Det finnes seks eller flere typer gyroskoper, men det fokuseres bare på den gamle, mekaniske typen og MEMS-gyroskoper (Mikro-elektronisk-mekanisk system), som er en av de mest anvendte typene ettersom de er billige å masseprodusere og krever lite energi (Watson, 2016).

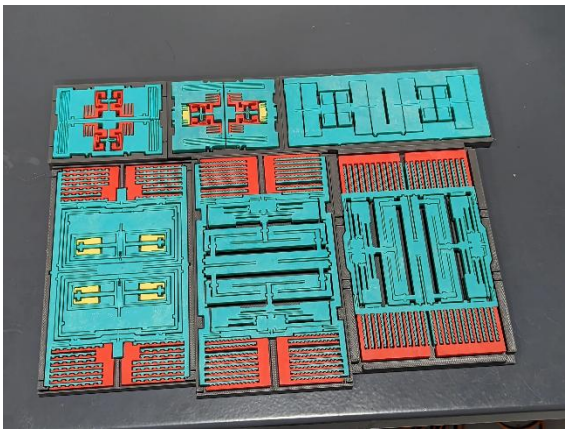
Mekaniske gyroskoper bruker et hjul eller en rotor som roterer i høy hastighet, vist i figur 7. Rotoren er montert i et oppheng som lar den rotere fritt i flere akser. Når rotoren roterer raskt nok, skapes det et vinkelmoment som opprettholder rotorens orientering i rommet. Denne effekten gjør at rotoren motstår endringer i orientering, unntatt langs sin egen rotasjonsakse (Brain & Bowie, 2023). Selv om mekaniske gyroskoper er svært nøyaktige og stabile over tid, har de noen ulemper. De er store og tunge, noe som gjør dem upraktiske for moderne, kompakte enheter, og de bevegelige delene er utsatt for slitasje. De krever også regelmessig vedlikehold og kalibrering for å opprettholde nøyaktighet.



Figur 8 Gyroskop med kardansk oppheng. Hentet fra snl.no

Likevel er mekaniske gyroskoper fortsatt mye brukt for å holde orienteringen til blant annet raketter og satellitter over lengre tid, fremfor presis måling og instrumentering (Brain & Bowie, 2023).

MEMS-gyroskoper er derimot svært små, mikroskopiske enheter, som vist i figur 8. De fungerer ved at interne strukturer vibrerer ved en gitt frekvens. Når gyroskopet roteres rundt en akse, oppstår Corioliseffekten, som beskriver hvordan en rettlinjet bevegelse i forhold til et koordinatsystem som er i ro, avbøyes dersom koordinatsystemet roterer (Store norske leksikon, 2005-2007).



Figur 9 Innsiden av MEMS-gyroskopet MPU- 6050, her representert av en 3D printbar model fra printables.com

Da MEMS-gyroskoper har lav kostnad, størrelse og energieffektivitet er de vanlige i alt fra smarttelefoner, droner og biler til spillkontrollere og VR-utstyr. Til tross for fordelene har de også noen begrensninger. De er mindre presise enn mekaniske gyroskoper og kan være utsatt for drift over tid, noe som påvirker nøyaktigheten. Eksterne faktorer som temperaturendringer og vibrasjoner kan dessuten påvirke ytelsen negativt, noe som gjør dem mindre egnet for applikasjoner som krever ekstrem nøyaktighet eller langvarig stabilitet (Watson, 2016).

Selv om mekaniske gyroskoper er mer driftssikre og presise over lengre tid, har MEMS-gyroskoper revolusjonert hvordan man bruker gyroskoper i hverdagslige applikasjoner. Teknologien utvikler seg fortsatt, med stadig mer presise og effektive varianter som kan dekke flere behov i fremtiden.

3.4. Akselerometre

Akselerometre er sensorer som reagerer på, eller registrerer endringer i fart eller retning, og de brukes ofte sammen med gyroskoper for navigasjon og styring av fly, raketter og ubåter. I tillegg anvendes de i vibrasjonsmåling i bygninger, som utløsere for kollisjonsputer i biler, og i spesialiserte formål som jordskjelvmåling via seismometre (Lenzi & Giorgio, 2022; Andrejašič, 2008).

Et akselerometer består av en bevegelig masse, som er suspendert i en ramme og koblet til fjærer. Når enheten utsettes for akselerasjon, beveger massen seg i forhold til rammen, og denne bevegelsen registreres. MEMS-akselerometre, som er blant de mest brukte typene, benytter ofte endringer i kapasitans for å måle denne forskyvningen. Kapasitansen endres når massen beveger seg, og dette signalet brukes til å beregne akselerasjonen (Andrejašič, 2008; Kistler, u.d.).

Moderne varianter er svært små og energieffektive, og finnes i alt fra smarttelefoner og spillkontrollere til droner og biler, der de brukes til å registrere bevegelser og rotasjon eller skifte mellom portrett- og landskapsmodus. MEMS-kapasitansbaserte akselerometre er også egnet for lavfrekvente vibrasjonsmålinger, noe som gjør dem verdifulle i strukturell helseovervåking og andre ingeniørapplikasjoner.

Begrensninger med MEMS-akselerometre inkluderer følsomhet for støy og miljøfaktorer som temperaturendringer og vibrasjoner. Selv om de er svært nøyaktige for mange formål, kan ekstreme påkjenninger påvirke ytelsen negativt, og de er mindre presise enn avanserte optiske eller laserbaserte systemer.

I prosjekter som involverer mobile roboter – som i dette tilfellet en minirobot med Kiwi Drive kan akselerometre benyttes for å overvåke robotens bevegelser. Sensorene gir informasjon om hvorvidt roboten beveger seg som forventet, har kjørt seg fast eller har veltet. Dette gir grunnlag for videre beslutninger i styringssystemet, enten det gjelder korrigerende tiltak eller feilhåndtering.

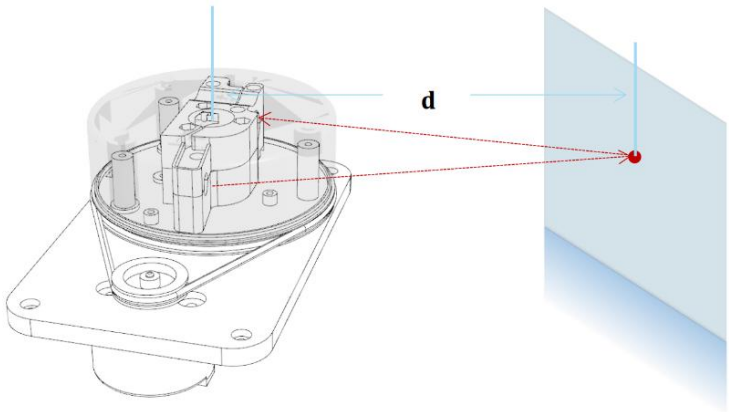
3.5. Lidar

Lidar (Light Detection and Ranging) som illustrert i figur 9, er en teknologi som bruker laserlys til å måle avstander og lage detaljerte 3D-modeller av omgivelsene. Dette oppnås ved å sende ut laserimpulser og måle tiden det tar før lyset reflekteres tilbake til sensoren, illustrert i figur 10. Avstanden beregnes ved hjelp av lysets hastighet og tiden det bruker på å reise frem og tilbake. Lidar kan operere i faste retninger eller rotere for å skanne et bredt område, noe som gir et komplett bilde av miljøet rundt sensoren (Slamtec, 2020).

Figur 10 Bilde av lidaren. Foto: Prosjektgruppen

Lidar sin evne til å levere nøyaktige målinger gjør den ideell for autonome systemer. Den brukes i alt fra kartlegging og terrengmodellering til navigasjon i selvkjørende biler og roboter. For roboten som skal produsere, vil Lidar spille en avgjørende rolle i sanntidsregistrering av miljøet. Den vil identifisere hindringer, kartlegge omgivelsene, og bidra til presis navigasjon i dynamiske og komplekse omgivelser (Wikipedia, u.d.).

Ved å bruke Lidar kan roboten oppnå høy grad av autonomi. Teknologien gir roboten mulighet til å oppfatte både små og store objekter med høy nøyaktighet, noe som er essensielt for å sikre effektiv drift og unngå kollisjoner. Lidar sin evne til å fungere uavhengig av lysforhold er også en stor fordel i varierende miljøer.



Figur 11 Lidarens virkemåte hentet fra Slamtec.com

Figur 11 i resultater viser et bilde av både lidar avlesninger, markert i blå linjer og ultralydsensor i røde linjer, overlappet med et bilde av miljøet hvor roboten navigerte.

3.6. Motordriver MX1616H

MX1616H er en avansert motordriver som er designet for å styre børstede DC-motorer med høy presisjon og effektivitet, se figur 12. Den benytter en H-bro-arkitektur som gjør det mulig å kontrollere både hastighet og rotasjonsretning på en stabil og energieffektiv måte (SinotechMixic, 2022).

En H-bro er en elektrisk krets som lar strømmen flyte i begge retninger gjennom motoren, noe som er nødvendig for å skifte mellom forover- og bakover bevegelse. Driveren omfatter flere funksjoner for gunstig motorstyring, inkludert justerbar PWM (Pulse Width Modulation) for presis hastighetsregulering. MX1616H har innebygd overstrømsbeskyttelse, og termisk beskyttelse for å forhindre overoppheting.

MX1616H støtter en bred spenningsrekkevidde og kan levere høy strømkapasitet, noe som gjør den egnet for motorer med større effektbehov sammenlignet med eldre løsninger som L298N.

Den har også høy effektivitet takket være bruken av moderne MOSFET-transistorer, som reduserer varmeutvikling og energitap.

Figur 12 Motordiver MX16161H
Foto: Prosjektgruppen

Fordeler med MX1616H

- **Høyere effektivitet** – Bruker moderne MOSFET-teknologi for lavere energitap og mindre varmeutvikling.
- **Høy strømkapasitet** – Kan levere mer strøm til motorene enn eldre motordrivere som L298N.
- **Justerbar PWM** – Gir presis kontroll over motorhastighet og respons.
- **Beskyttelsesmekanismer** – Innebygd termisk beskyttelse og overstrømsvern for økt pålitelighet.
- **Kompatibilitet** – Støtter en rekke mikrokontrollere, inkludert Arduino og Raspberry Pi.

Ulemper med MX1616H

- **Kompleksitet** – Kan kreve mer avansert konfigurering sammenlignet med enklere drivere.
- **Pris** – Kan være dyrere enn eldre alternativer som L298N.
- **Kjølebehov** – Selv om den er mer effektiv, kan den fortsatt kreve ekstra kjøling ved høy belastning.

Testing av MX1616H

For å teste om MX1616H fungerer som den skal, kan man koble den til en strømkilde og en DC-motor. Deretter måles spenningen på utgangene for å bekrefte riktig funksjon. En enkel testkode kan brukes til å styre motoren i begge retninger og variere hastigheten med PWM-signaler. Dette vil bekrefte at både retnings- og hastighetskontrollen fungerer som forventet.

I stedet for to digitale pinner for retning og én PWM-pinne for hastighet, bruker MX1616H to PWM-pinner (én per retningsinngang). Dette reduserer antall I/O-pinner som trengs for å kontrollere en DC-motor.

Ved å bruke MX1616H i prosjektet sikres bedre ytelse, pålitelighet og energieffektivitet sammenlignet med eldre motordrivere.

For å teste driveren på en enkel motor kan en testkode som vist under brukes.

Her defineres hvilke GPIO-pinner som brukes til å sende PWM-signal til motordriveren.

```
const int MOTOR_IN1 = 14;  // Fremover-retning
const int MOTOR_IN2 = 27;  // Bakover-retning
```

Pinnene settes som OUTPUT for å kunne sende signaler til motorstyringen.

```
pinMode(MOTOR_IN1, OUTPUT);
pinMode(MOTOR_IN2, OUTPUT);
```

Motorens hastighet styres med en PWM-verdi mellom 0 og 255.

PWM = 0 betyr at motorene står stille

PWM = 255 tilsvarer maksimal hastighet

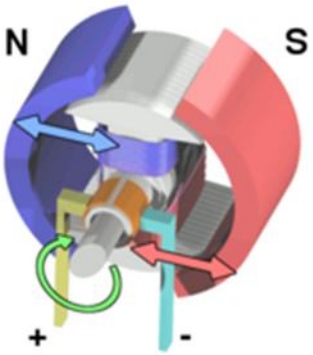
Verdien kan justeres for å øke eller redusere hastigheten trinnvis

```
    analogWrite(MOTOR_IN1, 0);    // PWM-verdi fra 0–255
    analogWrite(MOTOR_IN2, 0);    // Motsatt retning av
```

3.7. DC-Motor med enkoder

En likestrøms motor (ofte kalt en DC-motor, Direct Current) illustrert i figur 13, er en elektromagnetisk maskin som konverterer elektrisk energi til mekanisk energi ved hjelp av magnetiske krefter. Hovedkomponentene i denne motoren er statoren, armaturen, kommutatoren og børstene. (Wikipedia, 2025).

Den elektromagnetiske delen av motoren fungerer etter prinsippet når det går elektrisk strøm gjennom en leder, skapes et magnetisk felt rundt lederen. I motorens armatur er spoler viklet rundt en jernkjerne, og når strømmen går gjennom disse viklingene skapes et elektromagnetisk felt. Dette magnetiske feltet samhandler med det magnetiske feltet fra statormagnetene og skaper et dreiemoment som får armaturen til å rotere.

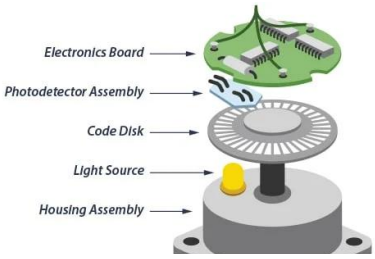


Figur 13 Prinsippskisse av en DC-motor. Hentet fra Wikipedia.

For at rotasjonen skal opprettholdes kontinuerlig, må strømretningen i viklingene endres periodisk. Dette gjøres av kommutatoren som er koblet til armaturen. Kommutatoren er en roterende bryter som bytter retningen på strømmen på viklingene slik at motoren fortsetter å rotere i en retning. Børstene fungerer som elektriske kontakter og overfører strømmen fra den stasjonære delen til den roterende kommutatoren.

En DC-motor med enkoder kombinerer den grunnleggende funksjonaliteten til en DC-motor med enkoder som gir posisjonstilbakemelding. Slik motor brukes ofte i robotikk, automatiseringssystemer og andre områder der nøyaktig posisjonering er nødvendig.

Motoren består av to hoveddeler: selve motoren og koderen. Motoren drives ved å tilføre spenning til terminalene, noe som får rotoren til å rotere. Koderen registrerer derimot rotasjonen av motorakselen og gir tilbakemelding om posisjonen (Veishida, 2023).



Figur 14 Prinsippskisse av enkoder. Hentet fra encoder.com

En typisk enkoder består av en disk med et mønster eller spor og en sensor som registrerer bevegelsen når disken roterer. Under rotasjon danner sensoren elektriske signaler basert på diskens mønster, som angir motorakselens posisjon, vist i figur 14.

Oppløsningen til koderen bestemmes av antallet spor på disken. Signalene fra koderen behandles av et kontrollsystem som bruker denne tilbakemeldingen til å justere spenningen som påføres motoren.

Dette lukkede sløyfekontrollsystemet gjør det mulig å kontrollere motorens posisjon, hastighet og akselerasjon med god presisjon (Veishida, 2023).

DC-motorer med enkoder, illustrert i figur 15, har flere fordeler sammenlignet med andre motorer. Den største fordelen er deres evne til å oppnå nøyaktig posisjonering, som er avgjørende i mange bruksområder. I tillegg er de svært effektive, har lang levetid og krever lite vedlikehold.

Figur 15 Motoren med enkoder og omni-hjul.. Foto: Prosjektgruppen

3.8. Ultralydsensor

HC-SR04 er en ultralydsensor som benytter lydpulser for å måle avstanden til objekter uten fysisk kontakt. Den opererer ved å sende ut en ultralydbølge med en frekvens på 40 kHz og måle tiden det tar før ekkoet returnerer etter å ha blitt reflektert fra et objekt. Basert på denne tidsmålingen kan avstanden beregnes, ettersom lydens hastighet i luft er kjent (Handson Technology).

Sensoren har fire tilkoblingspinner: VCC, GND, Trig og Echo. For å initiere en måling sendes en kort puls på 10 mikrosekunder til Trig-pinnen. Dette får sensoren til å sende ut en serie på åtte ultralydpulser. Når disse pulsene treffer et objekt, reflekteres de tilbake, og Echo-pinnen registrerer tiden det tar før ekkoet returnerer. Denne tidsforsinkelsen brukes deretter til å beregne avstanden til objektet ved hjelp av formelen:

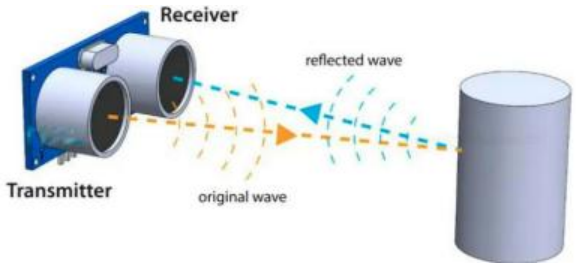
$$Avstand = (\frac{\text{Tidsforsinkelse x Lysets hastighet}}{2})$$

Lydens hastighet i luft er omtrent 343 meter per sekund, eller 0,0343 cm/µs. Ved å måle tidsforsinkelsen i mikrosekunder kan man dermed beregne avstanden i centimeter. For eksempel, hvis tidsforsinkelsen er 500 mikrosekunder, blir avstanden:

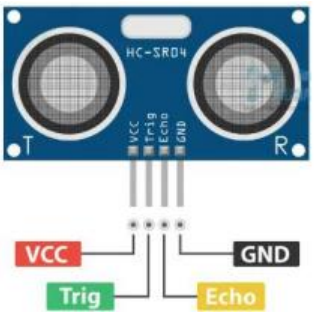
$$Avstand = \frac{(500\ \mu s\ \times\ 0{,}0343\ cm/\mu s)}{2}$$

HC-SR04 har et målområde fra 2cm til 400 cm med en nøyaktighet på ±3 mm. Den effektive målevinkelen er omtrent 15 grader, noe som betyr at sensoren er mest nøyaktig innenfor denne vinkelen. Den drives med en spenning på mellom 3,3 V og 5 V DC og har en hvilestrøm på mindre enn 2 mA, med en arbeidsstrøm på 15 mA (Dejan, u.d.).

I praktiske anvendelser brukes HC-SR04 ofte i robotikk for hindringsunngåelse, avstandsmåling og nivådeteksjon, for eksempel i parkeringssensorer eller væskenivåmåling. Dens enkle grensesnitt og pålitelige ytelse gjør den til et populært valg for mange elektronikkprosjekter.



Figur 16 Prinsippskisse av ultralydsensoren. Hentet fra howtomechatronics.com



Figur 17 Kobling av ultralydsensoren. Hentet fra howtomechatronics.com

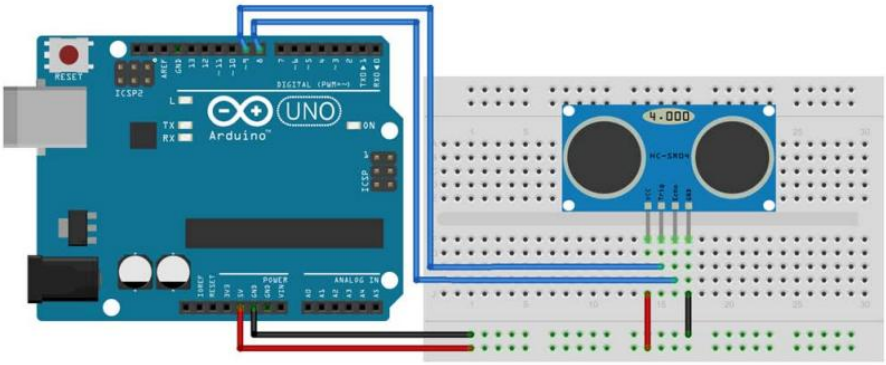
Fordeler:

- Er rimelig og tilgjengelig, noe som gjør den ideell for budsjettvennlige prosjekter.
- Nøyaktighet ved måling av avstander opptil ± 3mm under best mulige forhold.
- Fungerer godt med vanlige mikrokontroller som arduino og raspberry Pi.
- Fungerer godt i variert lysforhold, ikke avhengig av synlig lys som IR sensorer.
- Godt egnet til batteridrevet systemer grunnet lavt energibehov.
- Kan detektere objekter uavhengig av overflate og refleksivitet.

Ulemper:

- Begrenset synsvinkel på ca.15 grader, som gjør den mindre egnet for breddeovervåkning.
- Kan være følsom for støy, som igjen kan forstyrre målingene.
- Er ikke vanntett eller designet for ekstreme miljøer uten modifikasjon.
- Kan ha en begrenset oppdateringshastighet, som begrenser bruk ved raske applikasjoner.
- Under 2 cm kan sensoren gi upålitelige målinger.

Oppkobling og test av ultralydsensor:



Figur 18 Oppkobling av ultralydsensoren mot Arduino Uno. Hanson Technology

Viktigste deler av koden:

Setter trig-pinnen i høy, 5 V, så ultralydmodulen sender ut en kort lydpuls.

```
18  digitalWrite(trig, HIGH);
```

Venter 10 mikrosekunder for å sikre at pulsen er lang nok til å trigge sensoren.

```
19  delayMicroseconds(10);
```

Setter trig-pinne signalet tilbake til lav, 0 V, stopper pulsen og venter på tilbakemelding.

```
20  digitalWrite(trig, LOW);
```

Måler hvor lenge echo pinne signalet er i høy i mikrosekunder, tilsvarer tiden fra utsendelse til retur.

```
21  duration = pulseIn(echo, HIGH);
```

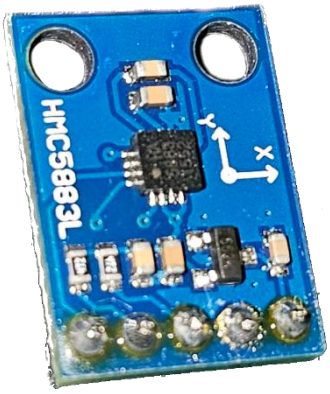
Når denne koden kjøres på en Arduino mikrokontroller og man observerer resultatet i «Serial monitor», kan man se at avstanden endres når et objekt for eksempel blir flyttet foran sensoren. Verdiene som vises, representerer den målte avstanden.

3.9. Kompass

HMC5883L er en tre-akset magnetisk sensormodul utviklet av Honeywell, vist i figur 19, primært designet for bruk i digitale kompasser og andre applikasjoner som krever nøyaktig måling av magnetfelt. Modulen benytter anisotrop magnetoresistiv (AMR) teknologi for å oppnå høy presisjon og følsomhet ved måling av jordens magnetfelt. Dette gjør den velegnet for navigasjonssystemer, robotikk og posisjoneringsapplikasjoner.

Den fungerer basert på prinsippet om anisotrop magnetoresistiv (AMR) effekt, hvor materialers elektriske motstand varierer i respons til ytre magnetiske felt. Sensoren består av tre AMR-elementer, hver plassert ortogonalt for å måle magnetfelt langs X-, Y- og Z-aksene. Når magnetfelt påvirker disse elementene, endres motstanden deres, og denne endringen omformes til en målbar spenning. Spenningen konverteres deretter til digitale verdier via en 12-bits ADC, noe som gir høy presisjon i feltmålingen (Honeywell).

For å beregne orientering benytter HMC5883L vektorsummen av de magnetiske komponentene. Ved å sammenligne målingene med jordens magnetfelt kan enheten bestemme sin retning relativt til magnetisk nord. I praksis benyttes ofte filtreringsteknikker og kalibreringsalgoritmer for å korrigere for forstyrrelser fra nærliggende metallstrukturer og elektriske komponenter. I kombinasjon med akselerometer og gyroskop kan disse dataene gi en god og stabil retningsbestemmelse.



Figur 19 Kompasmodul HMC5883L Foto: Prosjektgrupen

HMC5883L opererer innenfor et spenningsområde på 2,16 V til 3,6 V og kommuniserer via en I2C-grensesnitt, noe som muliggjør enkel integrasjon med mikrokontrollere som Arduino og Raspberry Pi. Sensoren har en 12-bits ADC (Analog-Digital Converter), som gir en oppløsning på to milligauss per LSB. Dette gir en vinkeloppløsning på mellom en og to grader avhengig av kalibrering og miljøforhold. Modulen består av tre magnetiske sensorer som måler X-, Y- og Z-komponentene av det omgivende magnetfeltet. Disse verdiene kan benyttes til å beregne en enhets orientering ved hjelp av kompassalgoritmer, ofte i kombinasjon med akselerometer og gyroskop for forbedret stabilitet og presisjon.

Fordeler: modulen har flere fordeler, inkludert høy nøyaktighet og følsomhet. Den 12-bits ADC-en gir en feltoppløsning på to milligauss, noe som resulterer i en nøyaktig retningsbestemmelse. Modulen har lavt strømforbruk,med et gjennomsnittlig forbruk på 100 μA i målemodus, noe som gjør den godt egnet for batteridrevne systemer. Den kompakte utformingen, med en fysisk størrelse på 3,0 × 3,0 × 0,9 mm, gjør den enkel å innlemme i små enheter.

Ulemper: Modulen har også noen ulemper. Den er følsom for eksterne magnetiske forstyrrelser, og nøyaktigheten kan påvirkes av nærliggende elektriske komponenter og metallstrukturer. I tillegg har Honeywell avsluttet produksjonen av HMC5883L, noe som kan begrense tilgjengelighet og fremtidig support.

4. Metode

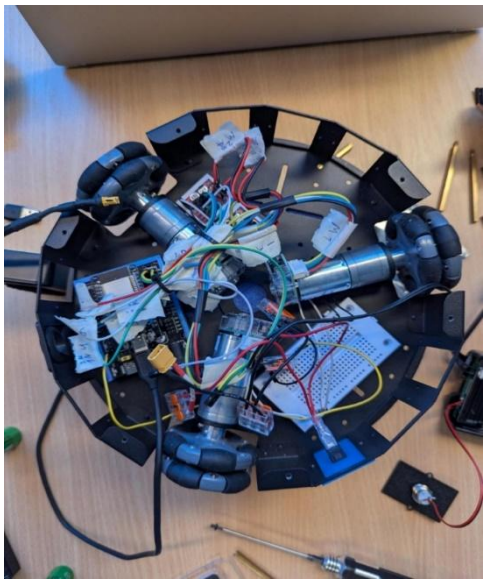
I dette kapittelet beskrives metodikken som ble brukt for å utvikle Lidarino-roboten, fra den strukturelle oppbygningen til realiseringen av styringssystemer. Prosjektarbeidet omfattet karosseriets utforming, valg og integrasjon av elektroniske komponenter, samt utvikling av programvare for den autonome navigasjonen og motorstyringen.

Et sentralt aspekt ved arbeidet, var oppdelingen av konstruksjonsarbeidet i en nedre og en øvre del fordelt på medlemmene i prosjektgruppen. Disse to delene av robotens karosseri og komponentplassering ble utført uavhengig av hverandre før den endelige sammensettingen. Arbeidsdelingen gjenspeiles i strukturen til dette kapittelet, hvor byggingen og komponentplasseringen for den nedre delen og den øvre delen, beskrives separat i kapittel 5.2 og 5.4.

Videre i kapittelet beskrives systemintegrasjonen og styringen av robotens funksjonalitet, som hvordan nøkkelsensorer som Lidar, ultralydsensor, IMU og kompass er integrert og samhandler for å samle inn data fra omgivelsene og robotens posisjon.

Dataene fra disse sensorene analyseres av Raspberry Pi 5 for å fastsette navigasjonsbanen til roboten, der kontrollsignaler sendes til ESP 32 for regulering av motorhastigheten. Styringen av motorene, herunder bruk av PWM-signaler, integreringen av PID-regulering og Kiwidrive-systemets kinematikk, blir gått gjennom for å redegjøre for hvordan man oppnår nøyaktig og stabil bevegelse.

Videre forklares kommunikasjonen mellom ESP 32 og Raspberry Pi 5 via serielle grensesnitt (UART) og I2C, i tillegg til andre aspekter ved systemintegrasjonen som strømforsyning (BMS, step-down omformer) og logging/datainnsamling.



Figur 20 Viser roboten under bygging. Foto: Prosjektgruppen

4.1. Git og GitHub

For å sikre en trygg lagring av kildekode og effektivt samarbeid er Git benyttet som versjonskontrollsystem og GitHub som skybasert kodelager. Git gir mulighet til å spore alle endringer, jobbe parallelt uten å overskrive hverandres arbeid, og enkelt rulle tilbake til tidligere versjoner om det skulle bli nødvendig. Dette gir både oversikt og trygghet i utviklingsprosessen.

Et kodelager i Git er stedet der all kode og dens historikk lagres samlet. Når et kodelager klones, lagres en full kopi av både filene og alle versjonsendringer. Det muliggjør tilbakerulling, sammenligning og sporing av utviklingen over tid. Et kodelager kan ligge lokalt på en maskin eller eksternt på en tjeneste som GitHub, og synkronisering mellom dem gjør samarbeid enkelt.

GitHub-plattformen fungerer som en nettskytjeneste for Git, hvor det kan opprettes, deles og samarbeides om lagrene. Her kan det følges med på hvem som har gjort hvilke endringer, administrere ulike grener og pull forespørsler, og legge til beskrivelser og dokumentasjon som gjør prosjektet mer oversiktlig for alle bidragsytere.

En gren i Git er en uavhengig utviklingslinje i et kodelager der man kan teste nye funksjoner eller fikse feil uten å berøre hovedgrenen/hovedkoden. Når en gren opprettes, tar Git et øyeblikksbilde av hovedgrenens tilstand, og all videre utvikling på grenen lagres isolert til man velger å slå dem sammen.

Pull forespørsel er den strukturerte forespørselen som sendes via GitHub for å slå sammen endringene med hovedkoden. Den viser alle modifikasjoner slik at teamet kan gjennomgå, diskutere og be om justeringer. Etter at endringene er godkjent, sammenføyes grenen i hovedlinjen og blir en permanent del av prosjektets historikk.

Som klient er GitHub Desktop, et grafisk grensesnitt, benyttet da det forenkler arbeidsflyten ved å muliggjøre opplasting og nedhenting av kodens nyeste versjon direkte fra GitHub uten behov for kommandolinje. Dette verktøyet har vært aktivt brukt i arbeidet med å synkronisere all kode for både ESP32- og Raspberry Pi 5-delene av prosjektet.

4.2. Konstruksjon og oppbygning av nedre del

Prosessen med konstruksjon og oppbygning av Lidarino-roboten tok utgangspunkt i et kit bestående av ferdigproduserte deler levert av oppdragsgiver. Dette ble et startpunkt for å bygge sammen roboten. For å fullføre systemet, måtte enkelte sentrale komponenter anskaffes av prosjektgruppen; dette inkluderte blant annet en Power hatt og BMS (Battery management system)

Karosseriet i den nedre delen av roboten har en rund modulær form. Det er konstruert av to hovedlag i metall: et bunnlag og et midtlag. Hvert av disse lagene dannes av en plate som er sammensatt av tre separate metalldeler, totalt seks deler. Rundt omkretsen av disse platene ble det satt inn 3 veggplater for å kapsle inn og beskytte robotens indre komponenter. Denne strukturen ble valgt for å skape en stødig plattform for montering av de viktigste fremdrifts- og elektronikkomponentene.

Selve drivsystemet er sentralt plassert i den nedre delen. Motorene, utstyrt med Omni-hjul for holonomisk bevegelse, er montert på bunnplaten med 120-graders mellomrom for å få en jevn fordeling. Motordriverne og ESP32 er også plassert her for enkel tilgang.

For å få god hinderdeteksjon i lav høyde, er det montert fire ultralydsensorer rundt kapslingen på roboten. To av disse sensorene er plassert i fronten med en 90-graders vinkel for å dekke et bredt synsfelt, og en på hver side bak for å oppdage hindringer som kan dukke opp der. Da ultralydsensorene gir ut et ekkosignal på 5 V, mens Raspberry Pi 5 krever 3,3 V måtte spenningsdeler installeres.

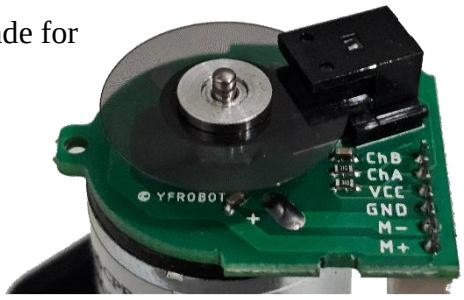
For å få best mulig stabil og funksjonell montering av de elektroniske komponentene, har gruppen selv modellert og designet alle nødvendige holdere for mikrokontrollere, motordrivere og øvrige komponenter som IR-mottaker og lignende i Autodesk Inventor og printet ut disse. Holderne ble så festet til karosseriet på de mest gunstige stedene som sørger for enkel tilgang samtidig som de gir beskyttelse mot eksterne påvirkninger.

4.3. Drivsystem og motorstyring

Det ble besluttet tidlig i prosjektet at det første som måtte gjøres var å få hjulene til å drive roboten for å kunne gå videre med avansert sensor styring.

Drivsystemet er bygd rundt en ESP32 som danner PWM-signaler til tre MX1616H-motordrivere. Hver av disse driverne styrer hver sin DC-motor med innebygd enkoder. Enkoderne sender kontinuerlig tilbake pulssignaler som måler hjulenes rotasjonshastighet, slik at ESP32 kan beregne sanntidshastighet. Basert på denne tilbakemeldingen justerer ESP32 PWM-arbeidssyklusen for å holde ønsket hastighet, noe som gir lukket kontrollstyring uten at andre komponenter er nødvendig.

Oppsettet med en motordriver per motor og dedikerte PWM-kanaler for hver motor sørger for trinnløs styring. Ved å bruke de innebygde enkoderne får man nøyaktige hastighetsmålinger direkte fra hjulene—noe som er helt avgjørende for presis navigasjon og jevn bevegelse. Dette enkle, men effektive samspillet mellom ESP32, MX1616H og de enkodede DC-motorene sikrer at roboten responderer raskt og pålitelig på alle bevegelseskommandoer. Videre i underkapitlene beskrives detaljert hvordan dette systemet er realisert og bygd opp.



Figur 21 Viser enkoderene som er koblet til motorene. Foto: Proskjetgruppen

4.3.1. PWM (Pulse Width Modulation)

Pulse-Width Modulation, er en metode for å styre effekten til elektriske laster ved å variere de aktive pulsene i et firkantpulstog. Periodetiden forblir konstant, mens forholdet mellom den aktive tiden og den totale periodetiden (arbeidssyklus), bestemmer hvor mye effekt som leveres til lasten.

Arbeidssyklusen «D» uttrykkes matematisk som:

$$\boldsymbol{D = \frac{T_o n}{T} \; x \; 100\%}$$

Hvor:

- Ton er tiden signalet er aktivt (høy tilstand) i en periode
- T er den totale periodetiden for signalet

Ved å integrere pulstoget oppstår en gjennomsnittlig spenning som tilsvarer prosentandelen til den aktive spenningen:

$$\boldsymbol{V_{avg} = D \times V_{max}}$$

Hvor:

- V_{max} er den maksimale spenningen tilført systemet.
- D er arbeidssyklus uttrykt som en desimal mellom 0 og 1

For eksempel, hvis et PWM-signal har en duty cycle på 50% og en forsyningsspenning på 12 V, blir den gjennomsnittlige spenningen som motoren mottar:

$$\boldsymbol{V_{avg} = 0,5 \; x \; 12 \; V = 6 \; V}$$

Dette betyr at motoren vil motta en effektiv spenning på 6 V, noe som reduserer hastigheten sammenlignet med full spenning.

Hvis arbeidssyklus økes til 75%, blir spenningen:

$$\boldsymbol{V_{avg} = 0,75 \; x \; 12 \; V = 9 \; V}$$

Ved å justere arbeidssyklusen kan vi variere motorhastigheten uten å kaste bort energi som varme, slik en lineær spenningsregulator ville gjort. MX1616H-motordriveren bruker PWM for hastighetsregulering, men fordi PWM alene ikke gir tilstrekkelig presisjon, benyttes en innebygd enkoder for sanntidsmåling av rotasjonshastigheten. En PID-kontroller justerer deretter kontinuerlig PWM-arbeidssyklusen for å holde ønsket hastighet.

Moderne mikrokontrollere som ESP32 har egne maskinvaremoduler for å danne PWM-signaler, noe som forenkler både maskin- og programvaredesignet. Før enkoderne ble tatt i bruk, testet vi motorene med IR-fjernkontroll og enkel testkode. Deretter ble programvare for Kiwidrive, som beregner og justerer PWM-verdiene for hvert av de tre hjulene (plassert med 120° mellomrom) uviklet, ved hjelp av en formel som gir en tilnærmet korrekt hastighet før PID-kontrollen trer inn.

Styringssignalet sendes som seriell data over USB, noe som legger til rette for fremtidig utvikling og forenkler kodeoppdatering fra Raspberry Pi 5 til ESP32. Selve signalet spesifiserer ønsket hastighet i X-, Y- og ω-koordinater for å realisere holonomisk bevegelse.

4.3.2. Generering av PWM-signaler

En enkel måte å danne et PWM-signal på er å bruke en sagtanngenerator sammen med en komparator. Referansespenningen som styrer pulsbredde sammenlignes kontinuerlig med en rampespenning. Hver gang rampen krysser referansen, skifter komparatorutgangen mellom høy og lav, og du får et firkantpulstog med konstant periode og justerbar "på-tid". Moderne mikroprosessorer som ESP32 har innebygde PWM-moduler der du bare angir ønsket frekvens og pulsbredde, så tar maskinvaren seg av resten uten ekstra kretser eller behov for CPU-styrte avbrudd.

For å sikre nøyaktige og stabile bevegelser overvåkes og justeres hjulenes hastighet kontinuerlig. Den maskinvare-genererte PWM-kontrollen kombineres med sanntidsmålinger fra enkodere, og en PID-kontroller oppdaterer pulsbredden fortløpende. Resultatet er presis, effektiv og lite prosessor-krevende motorstyring, som er avgjørende for jevn navigasjon i holonomiske robotsystemer. (Wikipedia, u.d.)

4.3.3. Kiwidrive

Det dukket opp flere problemer og utfordringer Under utviklingen av kiwidrive-programvaren, spesielt med å finne eksisterende Arduino-kode som passet prosjektets behov. Det fantes få relevante eksempler, noe som gjorde det nødvendig å komme frem til en egen løsning. Det ble søk på nettet og i andre forum etter matematiske modeller for kiwidrive som kunne brukes for å kontrollere motorene på riktig måte.

Kiwidrive-systemet mottar verdier for X-, Y- og rotasjonsbevegelse, og konverterer dem til hastigheter for hvert hjul i mm/s. Disse sendes videre til en PID-regulator (Proportional-Integral-Derivative), som justerer motorresponsen basert på avvik målt med hjulenes enkodere. Denne prosessen er beskrevet i detalj senere i rapporten.

Før enkoderene ble tatt i bruk, ble det laget et program som kontrollerte bilen via en IR-fjernkontroll. Testing av motorene ble utført enkeltvis, før kiwidrive-logikken ble innlemmet. Denne logikken bestemmer X-, Y- og omega-koordinater PWM-signalet til hvert hjul.

Hjulene er plassert med 120 graders mellomrom, og hver hastighet beregnes ut fra både translasjons- og rotasjonskomponentene.

Nedenfor vises en forenklet kiwidrive-formel for beregning av ønsket hjulhastighet (i = 1, 2, 3):

$$\boldsymbol{T_i = -\sin(\theta_i)\, v_x + \cos(\theta_i)\left(-v_y\right) + R \cdot \omega}$$

hvor:

- v_x er hastigheten i X-retning (mm/s)
- v_y er hastigheten i Y-retning (mm/s)
- ω er vinkelhastigheten (rad/s)
- R er avstanden fra senteret til hjulet
- θ_i er vinkelen til hvert hjul i forhold til X-aksen

For hjulene med vinkler $\theta_i = -30°, \theta_2 = 90° \; og \; \theta_3 = 210°$

kan følgende uttrykk utledes:

$$\boldsymbol{T_1 = \frac{1}{2} v_x - \frac{\sqrt{3}}{2} v_y + R \cdot \omega}$$

$$T_2 = -v_2 + R \cdot \omega$$

$$\boldsymbol{T_3 = \frac{1}{2} v_x + \frac{\sqrt{3}}{2} v_y + R \cdot \omega}$$

Arduino kode for kiwidrive

Trinnvis forklaring:

Skalering av lineær hastighet:

```
float vx = speedX * KIWI_SCALE_LINEAR;
float vy = speedY * KIWI_SCALE_LINEAR;
```

Konvertering av rotasjon fra grader/s til radianer/s:

```
float vr = rotation_dps * DEG_TO_RAD * KIWI_SCALE_ROTATION;
```

Beregning av translasjonskomponenter for hvert hjul:

```
float t1 = -sin(ANGLE_1) * vx + cos(ANGLE_1) * -vy;
float t2 = -sin(ANGLE_2) * vx + cos(ANGLE_2) * -vy;
float t3 = -sin(ANGLE_3) * vx + cos(ANGLE_3) * -vy;
```

Rotasjonshastighet (rad/s) konverteres til lineær hastighet (mm/s):

```
float rotation_linear = vr * ROTATION_RADIUS;
```

Translasjon og rotasjon kombineres til en endelig målhastighet for hvert hjul:

```
setpointSpeed[0] = t1 - rotation_linear;
setpointSpeed[1] = t2 - rotation_linear;
setpointSpeed[2] = t3 - rotation_linear;
```

4.4. IR-Mottaker og fjernkontroll:

IR-mottakeren og fjernkontrollen ble konfigurert ved hjelp av testkode funnet på Arduino sine forumer. Denne koden ble brukt som et utgangspunkt for å identifisere hvilke signalverdier som ble sendt fra hver enkelt tast på fjernkontrollen. Når en knapp ble trykket, viste den serielle monitoren i Arduino en 32-bits heksadesimal verdi. Deretter justerte vi koden slik at vi fikk ut data som representerer den konverterte IR-koden(kommandoverdien) som ble brukt i programmet. Disse verdiene ble notert ned i et tekstdokument og brukt videre i programmet for å styre bilens funksjoner under testingen.

For å hente ut rådata

```
Serial.println(IrReceiver.decodedIRData.decodedRawData, HEX);  // 32-bit råverdi
```

Konvertert signalkode eksempel:

Knapper på kontroll	Rådata	Konvertert
1	BA45FF00	0x45
2	B946FF00	0x46
*	E916FF00	0x16

Figur 22 Bildet viser fjernkontrollen som ble brukt. Foto: Prosjektgruppen

Figur 23 Bilde viser IR mottaker for mottak av fjernkontollkommandoer. Foto: Prosjektgruppen

4.5. PID-regulator

PID-regulatorer (Proporsjonal-Integral-Derivat) er en av de mest brukte kontrollmetodene for prosesskontroll. Hovedideen bak en PID-regulator er å kombinere tre kontrollstrategier: en proporsjonal del (P), en integrerende del (I) og en deriverende del (D), for å gi et kontrollsignal som er mer presist og stabilt. Målet er å redusere feilen mellom ønsket verdi (setpoint) og den faktiske verdien (målt verdi).

1. **Proporsjonal del (P)**

Den proporsjonale delen er basert på feilen mellom ønsket og faktisk verdi. Den gir en rask respons på feil, men kan føre til et steady-state-feil. Formel for proporsjonal kontroll er:

$$P_{ut} = K_p \cdot e(t)$$

hvor:

- P_{ut} er kontrollsignal fra proporsjonal del,
- K_p er proporsjonalforsterkeren,
- $e(t) = r(t) - y(t)$ er feilen, der $r(t)$ er ønsket verdi og $y(t)$ er målt verdi.

2. **Integrerende del (I)**

Den integrerende delen av PID-regulatoren akkumulerer feilen over tid og gir et kontrollsignal basert på summen av tidligere feil. Dette bidrar til å eliminere steady-state-feil. Formel for den integrerende kontrollen er:

$$I_{ut} = K_i \int_0^t e(\tau) d\tau$$

hvor:

- I_{ut} er kontrollsignal fra integrerende del,
- K_i er integrasjonsforsterkeren,
- $\int_0^t e(t)dt$ er den akkumulerte feilen over tid.

3. **Deriverende del (D)**

Den deriverende delen bruker hastigheten på endring av feilen for å forutsi fremtidige feil, og bidrar dermed til å redusere overskyting og forbedre stabiliteten. Formel for deriverende kontroll er:

$$D_{ut} = K_d \frac{de(t)}{dt}$$

hvor:

- D_{ut} er kontrollsignal fra deriverende del,
- K_d er derivasjonsforsterkeren,
- $\frac{d}{dt} e(t)$ er den tidsderiverte av feilen.

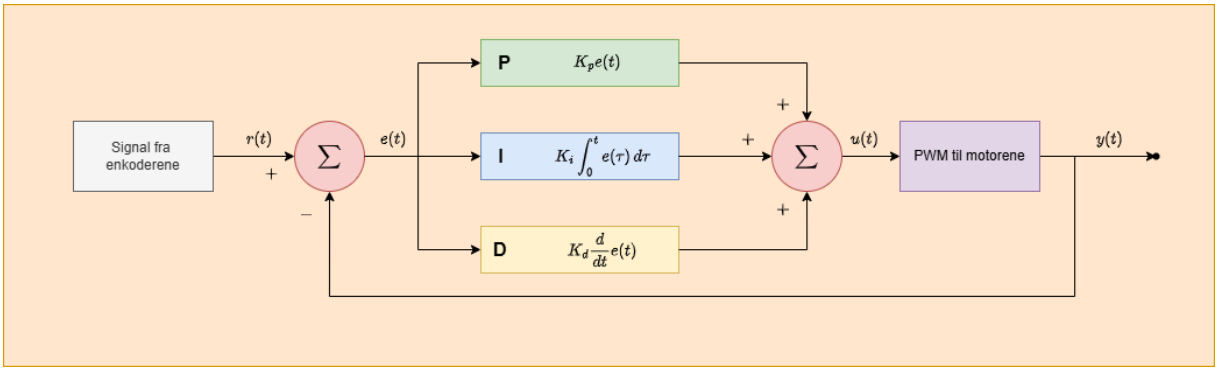
4. Total PID-kontrollsignal

Det totale PID-kontrollsignal er summen av de tre delene:

$$u(t) = P_u(t) + I_u(t) + D_u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

hvor u(t) er det totale kontrollsignal som sendes til systemet.

- K_p er den **proporsjonale delen** – en justerbar parameter som bestemmer hvor sterkt regulatoren reagerer på den nåværende feilen.
- K_i er den **integrerende delen** – en justerbar parameter som styrer hvor mye regulatoren reagerer på den akkumulerte feilen over tid.
- K_d er den **deriverende delen** – en justerbar parameter som påvirker hvor mye regulatoren reagerer på hvor raskt feilen endrer seg.
- $e(t) = SP - PV(t)$ er **feilen** ved tidspunktet t, der:
  - SP (Set Point) er ønsket verdi
  - $PV(t)$ (process varivable) er faktisk målt verdi i prosessen ved t.
- t er **tidspunktet nå** – det aktuelle tidspunktet regulatoren opererer på.
- τ er **integrasjonsvariabelen**, som representerer alle tidligere tidspunkter fra 0 til nåværende tid t i integrasjonsleddet (Wikipedia, u.d.).



Figur 24 Prinsippskisse av PID prosessen. Illustrasjon latet av prosjektgruppen.

4.5.1. Implementering av PID-regulering

Definere PID Gain for hvert hjull

```
float pid_kp[3] = {1.32, 1.32, 1.32};
float pid_ki[3] = {3.5, 3.5, 3.5};
float pid_kd[3] = {0.028, 0.028, 0.028};
```

PID-verdiene justeres individuelt for hvert hjul (indeks 0–2), og består av:

pid_kp: Forsterkning for P-leddet

pid_ki: Forsterkning for I-leddet

pid_kd: Forsterkning for D-leddet

Beregning av P, I og D ledd

```
float error = setpoint - actual;
float integral = pid_integral[i] += error * dt;
float derivative = (error - pid_lastError[i]) / dt;
```

P-ledd avvik mellom ønsket og målt hastighet (error = setpoint - actual)

I-ledd akkumulerer feil over tid (integral)

D-ledd endring i feil siden forrige måling (derivative)

De tre leddene kombineres for å beregne en reguleringsverdi (output), som deretter brukes til å styre motoren.

```
float output = (pid_kp[i] * error)
             + (pid_ki[i] * pid_integral[i])
             + (pid_kd[i] * derivative);
```

For å hindre at utgangsverdien overskrider tillatt PWM-område, begrenses den til [-255, 255].

```
output = constrain(output, -255, 255);
```

Hvis integratoren overstiger en definert grense (±100), kuttes den for å unngå at regulatoren blir ustabil over tid

```
pid_integral[i] += error * dt;

const float integralLimit = 100.0;
if (pid_integral[i] > integralLimit)  pid_integral[i] = integralLimit;
if (pid_integral[i] < -integralLimit) pid_integral[i] = -integralLimit;
```

Dette hindrer I delen i å bygge seg høyere en gitt verdi i integralLimit

Dersom utgangen når maksimal verdi (±255), trekkes siste oppdatering av I-leddet tilbake. Dette sørger for at integratoren ikke fortsetter å vokse unødvendig under metning.

Se Figur 29 Resultat for graffing av PID loop

Implementeringen av PID-reguleringen hadde flere utfordringer. I første fase ble en enkelt DC-motor med innebygd enkoder benyttet for testing. Avlesning av hastighet fra enkoderen utgjorde settpunktet i PID-loopen, og gjorde det mulig å måle avviket mellom ønsket og faktisk rotasjonshastighet. Testing og utveksling av kode ble gjennomført som fjernarbeid over nett der to gruppemedlemmer samarbeidet via kamera og mikrofon. Motoren i denne fasen støttet kun rotasjon forover og bakover, og manglet evne til diagonal- eller roterende bevegelser slik det kreves i Kiwi Drive. Planen var å overføre det fungerende oppsettet til robotplattformen når grunnleggende motorstyring var verifisert.

Under utviklingen av PID-løkken erfarte vi imidlertid at tilbakemeldingen fra hjulet oscillerte selv ved lave verdier av P, I og D. Feilsøking av både programkode og oppkobling ble utført for å identifisere kilden til svingningene. Etter at en fungerende PID-løkke var etablert, ble det klart at integratordelen kunne akkumulere vedvarende avvik – en effekt kjent som integrator-oppbygging – slik at når avviket senere gikk mot null, førte den oppbygde I-verdien til langvarig oversving inntil integratoren normaliserte seg. For å forhindre dette ble en mot-oppbyggingsmekanisme innført, der integratordelen begrenses slik at I-summen ikke overskrider forhåndsdefinerte grenser.

Da PID-kontrolleren skulle overføres til samordnet styring av tre Omni-hjul plassert med 120° rotasjonsforskyvning, oppsto det behov for omfattende omstrukturering av eksisterende kodebase. Hver PID-løkkes beregninger måtte kjøres parallelt og synkroniseres slik at alle tre motorene responderte korrekt på felles referansepunktkommandoer. I denne fasen benyttet vi eksterne ressurser, blant annet ChatGPT og faglitteratur, for å finjustere algoritmene og sikre at koden fungerte som ønsket på tvers av alle hjul. Til slutt ble overføringen og samkjøringen av motorstyringen vellykket, og Kiwi Drive-funksjonaliteten kunne utnyttes i systemet.

4.6. Konstruksjon og oppbygging (Øvre del)

På den øvre delen av robotplattformen er det installert en LIDAR-sensor, et elektronisk kompass, en gyrosensor og en Raspberry Pi 5, samt en IR-mottaker for manuell styring via fjernkontroll. Disse komponentene utgjør det primære sensor- og kontrollsystemet som muliggjør dynamisk navigasjon ved å formidle styringssignaler til ESP32-enheten gjennom en seriell kommunikasjonsprotokoll.

LIDAR-enheten er plassert sentrert på en spesiallaget plate, da robotens originale karosseri ikke hadde festemekanisme for komponenten. Denne platen ble tegnet i Inventor og 3D-printet, for å sikre at LIDAR gir nøyaktig avstandsmåling av omgivelsene. Autonom navigasjon og effektiv hindringsunngåelse blir så utført av roboten under kjøring.

I den nedre del av roboten ble ultralydsensorene montert, for å bedre deteksjon av mindre hindringer som ikke ville bli oppdaget av LIDAR grunnet sin plassering. Kompassmodul ble montert i senter av roboten sin konstruksjon for best beskyttelse mot ytre elementer. IR mottaker ble plassert i toppen av roboten, sammen med Lidar for å sikre at den er i et åpent område for å motta signal fra fjernkontrollen for manuell styring. Kommunikasjonen mellom komponentene kobles sammen via serielle grensesnitt UART og I2C som bidrar til effektiv dataoverføring og koordinert styring.

Spesialtilpassede vegger ble konstruert for roboten som et supplement til toppdekselet. Veggene ble designet i Inventor og 3D-printet før montering. Deres funksjon er å optimalisere luftsirkulasjonen internt, i samspill med en toppmontert vifte som fjerner varm luft. Denne løsningen var for å forhindre overoppheting av komponenter, spesielt Raspberry Pi, basert på observerte tendenser i tidligere testfaser.

4.7. Systemintegrasjon

I denne delen tar vi kort for oss hvordan de ulike komponentene kommuniserer med hverandre, det brukes ulike kommunikasjonsprotokoller for å utveksle data mellom mikrokontroller, ultralydsensor, kompass, gyro og lidar. De vanligste protokollene inkluderer UART og I2C, disse formene har vær sine egenskaper, fordeler og bruksområder.

UART

UART er en seriell kommunikasjonsprotokoll som benytter to ledninger (TX og RX) for å sende og motta data mellom enheter. Det fungerer asynkront, noe som betyr at ingen felles klokke brukes – i stedet synkroniseres dataoverføringen med spesifikke baud-rater (f.eks. 115200 bps). UART blir brukt av komponentene ESP32 og LIDAR som kommunikasjonsprotokoll mot Raspberry Pi 5. (Siebeneicher, 2024).

I2C

I2C er en seriell kommunikasjonsprotokoll som bruker to ledninger (SCL – Serial Clock, SDA – Serial Data) for å koble flere enheter til samme buss. I2C er synkron, noe som betyr at en master-enhet kontrollerer overføringen via et felles klokkesignal (Irazabal & Blozis, 2003). De komponenter som bruker I2C protokollen i prosjektet er kompass modulen HMC5883L og Gyroskopet MPU-6050.

For å få frem adressene til enhetene som er koblet til kommunikasjonsbussen kjøres følgende kommando i terminalen på Raspberry Pi:

```
i2cdetect -y 1
```

Adressene blir vist som i tabellen under.

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                          -- -- -- -- -- 0d -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

GPIO

Ultralydsensorene HC-SR04 ultralydsensorene bruker ikke en standard kommunikasjonsprotokoll som de nevnt ovenfor, i stedet kommuniserer de ved hjelp av digitale pulser på to pinner, trig og echo.

4.7.1. Konfigurering av Raspberry Pi 5: Enhetsnavn og LiDAR-stopp

I dette kapitlet beskrives et samlet oppsett på Raspberry Pi 5 som sikrer at ESP32 og RPLidar alltid tilordnes de samme dev-portene, og at LiDAR-motoren stoppes automatisk ved oppstart. Følgende trinn gjennomføres:

Opprett prosjektmappe og Python-miljø:

Lag katalogen for RPLidar-scriptet og bibliotekene:

```
mkdir -p /home/admin/Rplidar
```

Endrer gjeldende arbeidsmappe til prosjektkatalogen der alle filer skal samles.

```
cd /home/admin/Rplidar
```

python3 -m venv venv Genererer en undermappe venv med isolert Python-installasjon for prosjektet, slik at globale biblioteker ikke påvirkes.

```
python3 -m venv venv
```

Aktiverer det virtuelle miljøet ved å laste inn miljøets variabler fra activate-skriptet. Terminalprompten vil nå vise (venv).

```
source venv/bin/activate
```

Installerer nødvendige Python-biblioteker i det virtuelle miljøet: rplidar for LiDAR-kontroll, pyserial for seriell kommunikasjon, smbus2 for I²C og RPi.GPIO for GPIO-styring.

```
pip install rplidar pyserial smbus2 RPi.GPIO
```

Deaktiverer det virtuelle miljøet, slik at påfølgende Python-kommandoer bruker systemets standardinstallasjon.

```
deactivate
```

Dette gir et isolert Python-miljø (venv) der nødvendige pakker installeres uten å forstyrre systemets globale bibliotek.

Definer symbolske lenker med udev-regler:

Finn USB-enhetenes unike ID-er:

Lister alle enheter under /dev som inneholder "USB" i navnet (for eksempel /dev/ttyUSB0), slik at du kan identifisere hvilken port enheten bruker.

```
ls /dev/*USB*
```

Henter de seks første linjene med attributter idVendor, idProduct og serial for enheten tilknyttet /dev/ttyUSB0, slik at unike ID-verdier kan lagres.

```
udevadm info -a -n /dev/ttyUSB0 | grep -E 'idVendor|idProduct|serial' |
head -n6
```

Opprett regelfil /etc/udev/rules.d/99-usb-serial.rules med innhold:

Åpner en ny udev-regelfil med Nano under root, der regler for symbolske lenker basert på enhetens ID-er blir definert.

```
sudo nano /etc/udev/rules.d/99-usb-serial.rules
```

To linjer som sjekker om enhetens idVendor og idProduct samsvarer, og oppretter symbolske lenker /dev/rplidar og /dev/esp32.

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60",
SYMLINK+="rplidar"

SUBSYSTEM=="tty", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="7523",
SYMLINK+="esp32"
```

Last inn og anvend reglene:

```
sudo udevadm control --reload-rules
```

Anvender de nye reglene på eksisterende enheter, slik at symbolske lenker blir generert umiddelbart.

```
sudo udevadm trigger
```

Viser detaljer om de symbolske lenkene, slik at du kan bekrefte at de peker til riktige ttyUSB-porter.

```
ls -l /dev/rplidar /dev/esp32
```

Nå vil /dev/rplidar og /dev/esp32 peke riktig ved hver oppstart.

Eksempel på nøyaktige utdata som viser at /dev/esp32 lenker til ttyUSB1 og /dev/rplidar til ttyUSB0.

```
lrwxrwxrwx 1 root root 7 Apr 21 13:46 /dev/esp32 -> ttyUSB1
lrwxrwxrwx 1 root root 7 Apr 21 13:46 /dev/rplidar -> ttyUSB0
```

Lag Python-skript for automatisk stopp av LiDAR:

Opprett filen /home/admin/Rplidar/lidar_controller.py:

```
nano /home/admin/Rplidar/lidar_controller.py
```

Åpner en ny fil i Nano i prosjektmappen, der du limer inn Python-scriptet.

```
from rplidar import RPLidar
lidar = RPLidar('/dev/rplidar')
lidar.stop()
lidar.stop_motor()
```

Dette korte scriptet initialiserer RPLidar-klienten på /dev/rplidar, sender kommandoen stop() for å stoppe dataflyt, og stop_motor() for å slå av motoren.

Gjør den kjørbar:

```
chmod +x /home/admin/Rplidar/lidar_controller.py
```

Sett opp systemd-tjeneste for oppstart:

Opprett /etc/systemd/system/lidar-controller.service:

```
sudo nano /etc/systemd/system/lidar-controller.service
```

Oppretter en ny systemd-servicefil under /etc/systemd/system/ for å definere hvordan lidar_controller.py skal håndteres som en bakgrunnstjeneste.

```
[Unit]
Description=RPLidar & Robot control daemon
After=dev-rplidar.device dev-esp32.device network.target

[Service]
Type=simple
User=admin
WorkingDirectory=/home/admin/Rplidar
ExecStartPre=/bin/sleep 5
ExecStart=/home/admin/Rplidar/venv/bin/python3
/home/admin/Rplidar/lidar_controller.py
Restart=always
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
```

I [Unit]-seksjonen spesifiserer After= at tjenesten skal vente til symbolske lenker og nettverk er klare. ExecStartPre=/bin/sleep 5 gir en pause for å sikre at USB-enhetene initialiseres før skriptet kjøres. I [Service] kjører ExecStart scriptet i det virtuelle miljøet, mens Restart=always sørger for å gjenstarte tjenesten ved eventuelle feil. Utdata logges i journalen.

Aktiver og start tjenesten:

Oppdaterer system med nye/endrede tjenestedefinisjoner.

```
sudo systemctl daemon-reload
```

Setter tjenesten til å starte automatisk ved systemstart.

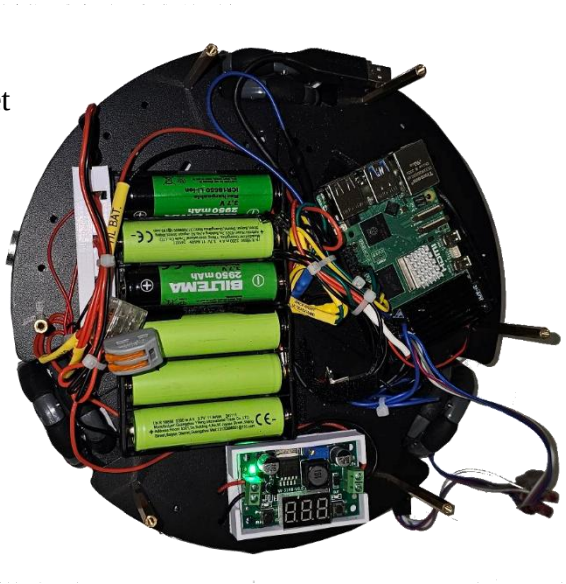
```
sudo systemctl enable lidar-controller.service
```

Starter tjenesten umiddelbart uten behov for omstart.

```
sudo systemctl start lidar-controller.service
```

Ved neste omstart vil LiDAR-motoren automatisk stoppes, og de faste enhetsnavnene vil være på plass, noe som sikrer konsistent og driftssikkert oppsett uten behov for manuell inngripen.

4.7.2. Batteripakke



Figur 25 Viser roboten med batteriene, DC-DC konverter og Raspberry Pi. Foto: Proskjetgruppen

Innledningsvis ble det forsøkt å utvikle en egendesignet batteripakke for roboten. Denne ble designet ved hjelp av Autodesk Inventor og deretter 3D printet en prototype med mål om å sikre tilstrekkelig driftsspenning for alle tilknyttede komponenter. Imidlertid ble det identifisert utfordringer knyttet til pålitelig elektrisk tilkobling av batteripolene, noe som resulterte i en beslutning om å anskaffe kommersielt tilgjengelige batteriholdere. Innledende ytelsestesting avdekket at robotens strømforsyningskapasitet var utilstrekkelig for den tiltenkte driften. For å adressere denne begrensningen ble gjort et redesign av batterisystemet. Den endelige konfigurasjonen bestod av en batteripakke basert på seks 18650-battericeller, arrangert i to parallelle koblinger av tre seriekoblede celler (3S2P). Denne konfigurasjonen sikret tilstrekkelig spenning og strøm for å drive roboten gjennom testperioden.

For å regulere og beskytte batteriene benyttes et Battery Management System (BMS) sammen med en spenningsomformer. BMS spiller en avgjørende rolle i å overvåke og optimalisere batteriets ytelse, da det hindrer overlading, overutlading og kortslutning, samt sørger for cellebalansering for jevn spenning mellom cellene. Dette forlenger batteriets levetid og øker sikkerheten (MPS, 2025).

I tillegg gjør BMS det mulig å lade batteripakken uten å demontere roboten, noe som forenkler vedlikeholdet. Sammen med spenningsomformeren sikrer det at Lidarino får stabil og riktig strømtilførsel til alle komponenter, noe som er essensielt for driftssikkerheten.

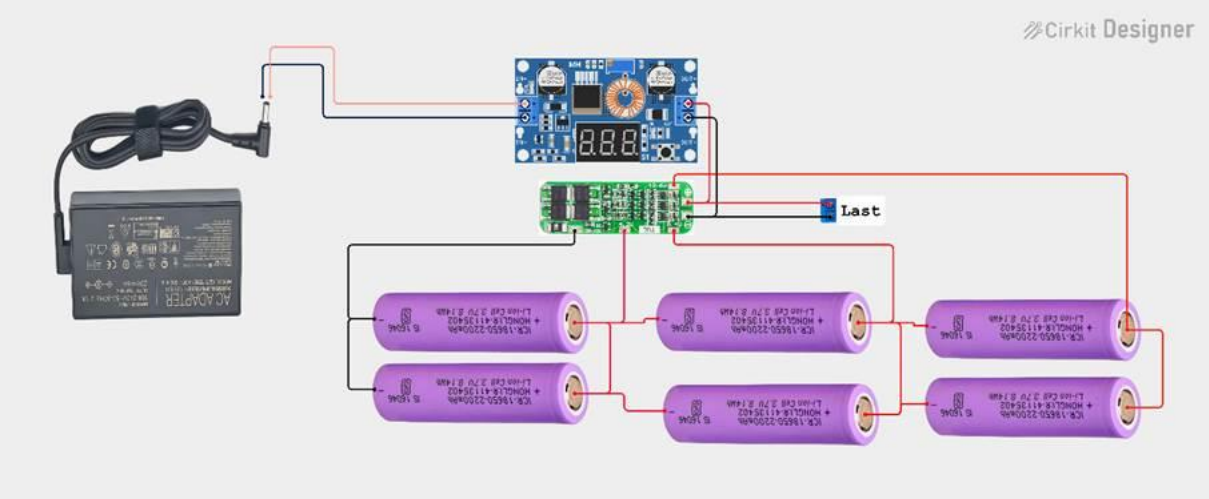
Komponentene som er brukt for strømkilden er:

XL4015 5A DC Buck Step-down

Beskrivelse: En DC-DC step-down (buck) omformer som kan håndtere opptil 5 A strøm.

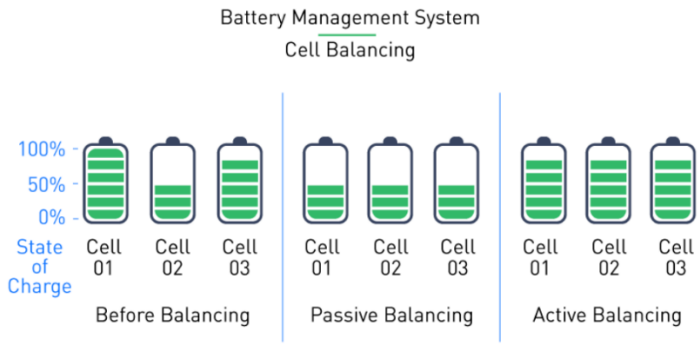
3s 20 A BMS

Beskrivelse: Et batteristyringssystem for 3-serie (3s) litiumionceller, med en maksimal strømhåndtering på 20 A.



Figur 26 Batterisystemet tegnet ved hjelp av Cirkit Designer

BMS som er brukt bruker passiv balansering. Passiv balansering innebærer å spre overskuddsenergi fra celler med høyere spenning som varme, vanligvis ved hjelp av motstander koblet parallelt med hver celle. Denne metoden har som mål å redusere ladningen i de sterkere cellene for å matche de svakere.



Figur 27 Viser prinsippet for balansering av batterier (MPS, 2025).

Balansering sikrer at den totale kapasiteten ikke begrenses av at en enkelt celle når sine spenningsgrenser for tidlig. For det andre øker balansering levetiden til hver celle ved å forhindre overlading og overutlading. Disse forholdene kan forårsake ytelsesforringelse og forkorte batteriets levetid. Ved å utjevne ladenivåene, opererer alle cellene innenfor et tryggere spenningsvindu. For det tredje maksimerer det den brukelige kapasiteten til batteriet. Uten balansering kan en batteripakke slutte å lade når den høyest ladede cellen når sin spenningsgrense, og etterlater andre celler underladet. Balansering gjør at alle cellene kan bidra med sitt fulle potensial, og øker dermed den ladbare kapasiteten og forbedrer den brukelige kapasiteten.

I praksis opplevde vi at batteriene kunne brukes lengre før de måtte lades. Samtidig sørget BMS for å beskytte batteriene ved å kutte spenningen når den blir for lav, som forhindrer skader og bidrar til lengre levetid.

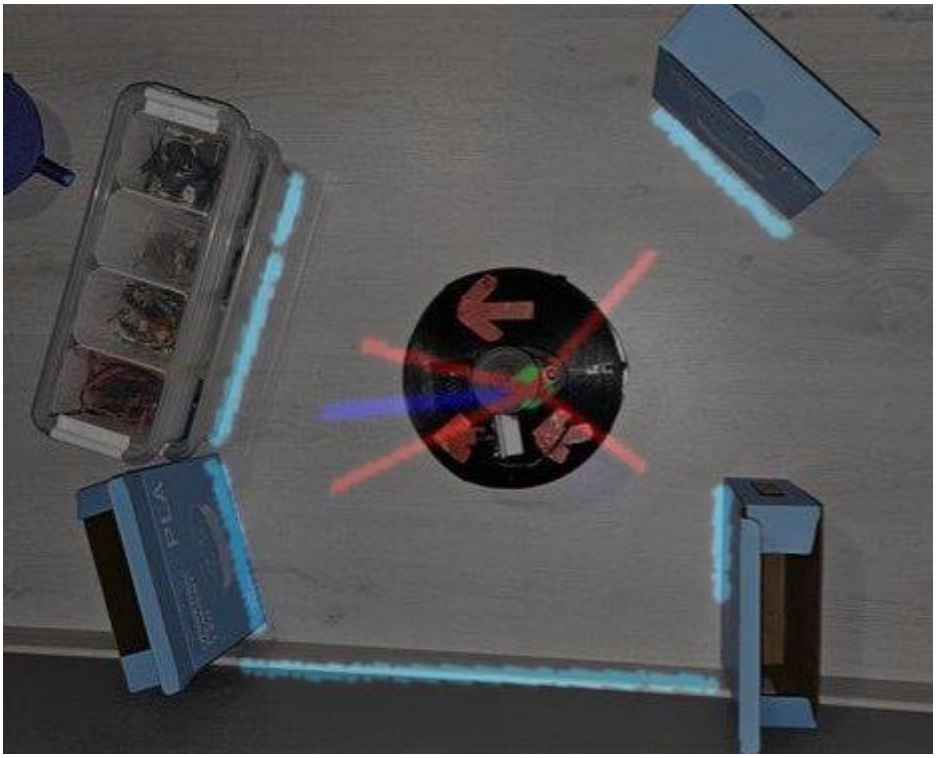
4.7.3. Step-down DC-DC omformer

For å sikre riktig spenningsnivå til roboten og dens komponenter, er det brukt en step-down omformer. Spesifikt er XL4015 en DC-DC buck-spenningsregulator som omdanner en høyere inngangsspenning (opptil 36 V) til en justerbar og stabil lavere utgangsspenning (mellom 1,25 V og 32 V), med en nominell utgangsstrøm på opptil 5A. Enheten benytter en effektiv svitsjebasert teknologi der den interne transistoren raskt kobler inngangsspenningen av og på, mens et lavpassfilter bestående av spole og kondensator glatter ut pulsmodulert strøm for å oppnå jevn likespenning. Videre er XL4015 utstyrt med flere sikkerhetsmekanismer som overstrøms-, overtemperatur- og kortslutningsbeskyttelse, noe som sikrer en trygg og pålitelig drift i kravstore applikasjoner. Denne løsningen øker robotens fleksibilitet ved å sikre kontinuerlig strømforsyning, kombinert med en ladeenhet som eliminerer behovet for frakopling av batteriet ved ladning. Videre forbedrer denne metoden systemets energieffektivitet, da buck-konvertering minimerer effekttap sammenlignet med lineære regulatoren (ELECROW).

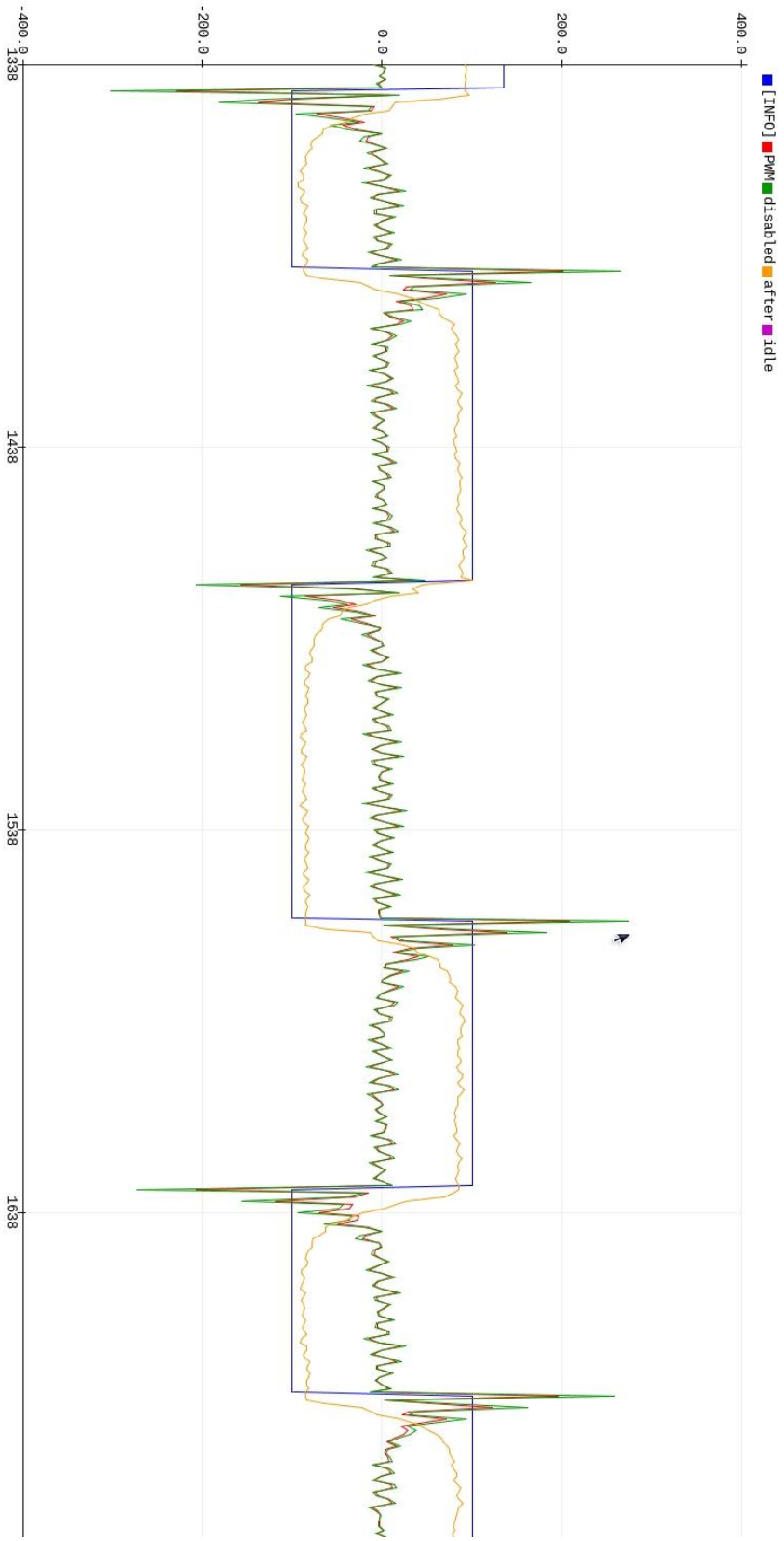
4.7.4. Skybasert Logging og Datainnsamling

For å overvåke systemets ytelse og gjennomføre en mulig analyse av operasjonelle data, er InfluxDB Cloud benyttet som en skybasert tidsstempeldatabase. Denne løsningen muliggjør kontinuerlig logging av sanntidsdata fra RPLidar, ultralydsensorer, ESP32, kompass og gyro. Den lagrede informasjonen benyttes for å evaluere systemets dynamikk, optimalisere respons ved hindringsdeteksjon og tilrettelegge for videre utvikling og feilsøking. Den skybaserte loggingen reduserer behovet for lokal infrastruktur og gir en skalerbar plattform for fremtidige utvidelser,se Figur 32.

5. Resultat



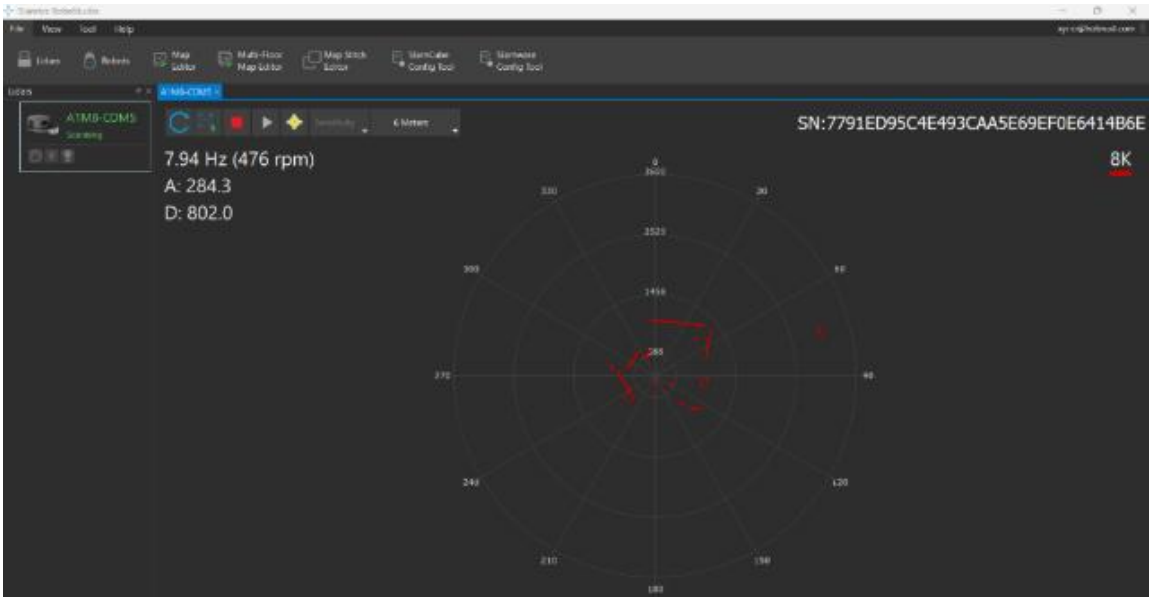
Figur 28 Bildet over viser en overlapp av miljøet og Lidar avlesning med blå strek, og med ultralydsensorer med røde streker. Foto: Prosjektgruppen



Figur 29 Graff av PID loop kjøring på 1 hjul

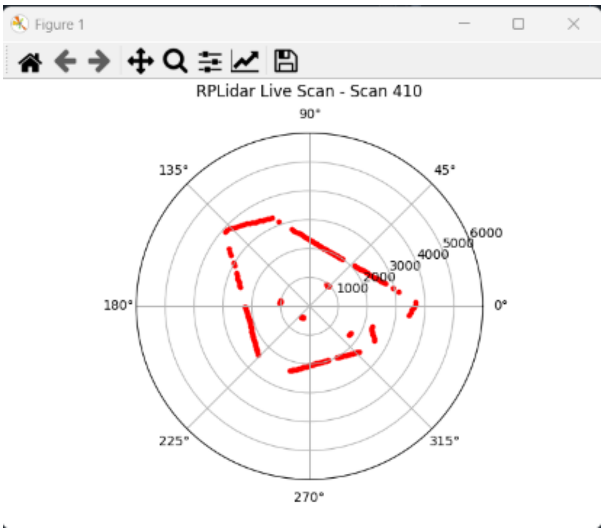
Lidar:

Etter anskaffelse av Lidar testet vi selve komponenten ved å kjøre den i produsentens egne testprogram Robostudio, dette var for å verifisere at modulen presterte slik vi forventet.

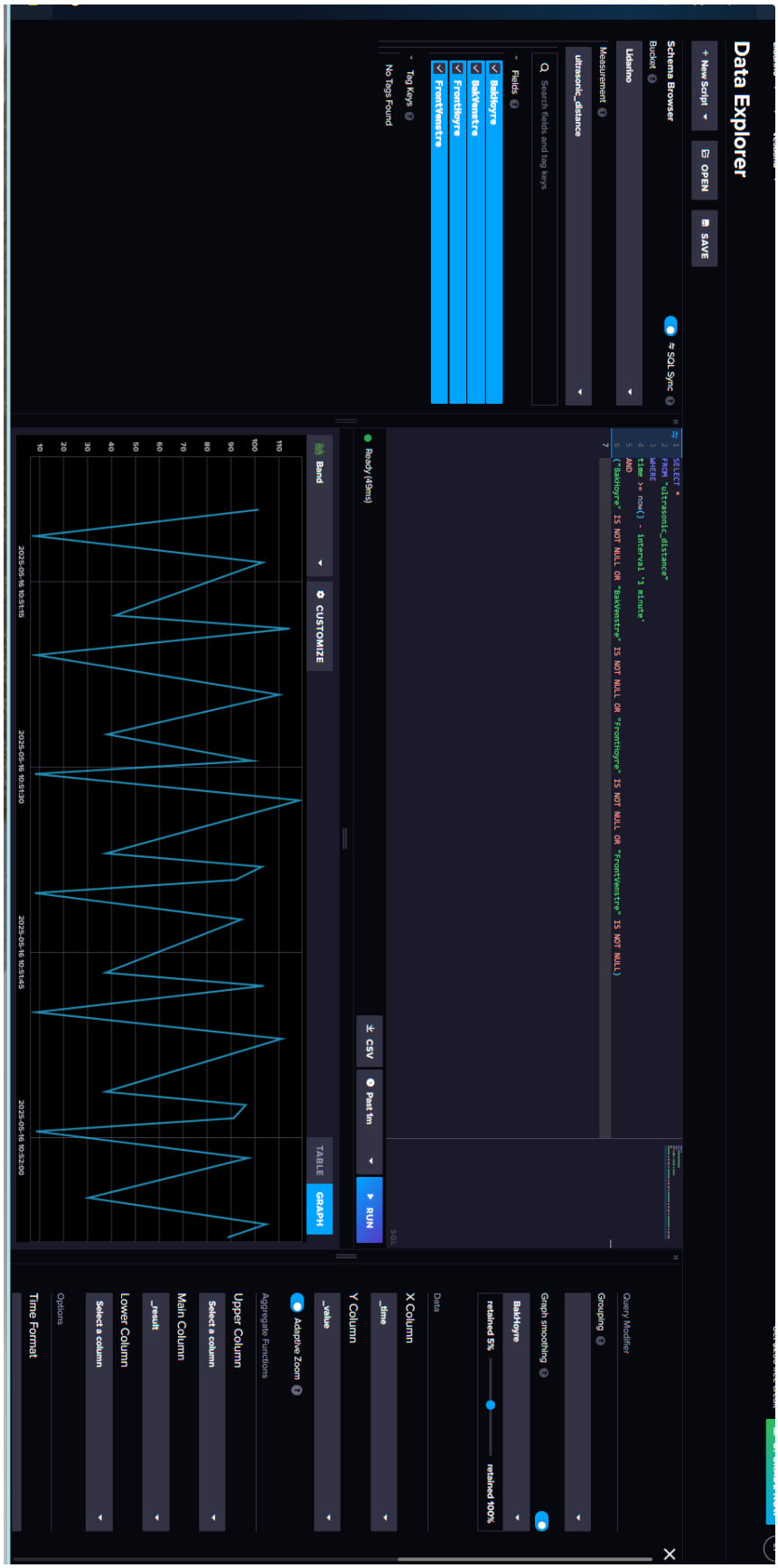


Figur 30 testbilde fra Lidar i robostudio:

Etter testen, utviklet vi en kode i Spyder som presentert et sanntidsbilde av omgivelsene:



Figur 31 Scan gjort med python i rom på skolen.



Figur 32 Skjermdump av Influx db under testing av ultralydsensorene.

6. Drøfting

I løpet av prosjektet har gruppen opparbeidet seg faglig utvikling innen robotikk, sensorteknikk og programmering. Arbeidet har vært utfordrende, og vi har støtt på flere problemer som har krevd både kreativitet, samarbeid og tålmodighet. Begrenset forkunnskap, manglende erfaring med programmering, robotikk, samt tidspress, ble ikke målet om presis parkering i et utvalgt hjørne nådd. Årsakene til dette og vurderingene som ble gjort underveis drøftes videre i de følgende kapitlene.

6.1. Inndeling av prosjektet

For å håndtere prosjektets ulike deler på en effektiv måte, fordelte vi ansvaret i prosjektgruppen. Enkelte deltok mer i den nedre delen som omfattet drivsystemet med tilhørende komponenter som ESP32, motordrivere, kiwidrive og mekanismen for bevegelse samt IR-mottaker og fjernkontroll og tilhørende kode. Andre fokuserte på den øvre delen som omfattet Raspberry Pi 5, sensorer som Lidar, ultralydsensor, kompass, gyro og programmering av disse.

For å gjøre drøftingen mest mulig oversiktlig og tydelig, er arbeidet med den nedre og øvre delen av roboten behandlet separat i henholdsvis kapittel 7.2 og 7.3. Disse delene er skrevet hver for seg. I etterfølgende kapitler samles erfaringer i en felles drøfting av prosjektets helhet, der blant annet kommunikasjon mellom delene og samarbeidsutfordringer blir belyst.

6.2. Nedre del av roboten

Som en del av utviklingsarbeidet ble den nedre delen av roboten planlagt og bygget med fokus på bevegelse og motorstyring. Denne delen inkluderte drivsystemet, motorstyring, bevegelse og fjernkontroll. I det følgende kapittelet drøftes de tekniske utfordringene, og løsningene som ble utviklet underveis i arbeidet før drøftingen går videre til den øvre delen av roboten.

6.2.1. Oppbygning og plassering av komponenter

Etter planlegging og oppsett valgte vi å plassere komponentene i den nedre delen av roboten ut ifra brukermanualen som kom med robot-kittet. Dette bestod av monteringsplater, festemateriell og tre DC-motorer med Omni-hjul som ble montert med 120 graders vinkel mellom hvert hjul for å kunne realisere et kiwidrive-oppsett.

ESP32-kontrolleren, sammen med motordriverne, ble plassert i bunnen av roboten ved hjelp av selvlagde holdere som ble 3D-printet og limt fast. Plasseringen ble valgt basert på tilkomst, tilkobling og begrensningen med plass etter vi hadde koblet til alt av ledninger og drivsystem.

I ettertid ser vi at det kunne være hensiktsmessig å lage en løsning med kabelfester for å få en mer ryddig intern struktur i den nedre delen. Dette ble valgte bort på grunn av behovet for hyppig demontering og remontering underveis i prosjektet, som gjorde en enkel løsning mer praktisk.

Tidlig i prosjektet oppstod det problemer med den første ESP32-enheten, som sluttet å kommunisere over USB. Dette ble sannsynligvis forårsaket av at motordriverne trakk strøm gjennom USB-porten, noe som førte til en belastning høyere enn det ESP32-en er konstruert for å tåle. Dette avdekket en svakhet i strømforsyningsdesignet, men siden systemet uansett ikke var ment å drives via USB alene, ble det vurdert som unødvendig å gjennomføre ytterligere beskyttelsestiltak på dette området.

Valget av MX1616h som motordriver ble gjort med hensyn til effektivitet og utvidbarhet. Den krever kun to PWM-signaler per motor for både hastighet og retningsstyring, noe som reduserer antall nødvendige tilkoblinger for implementering av kiwidrive. Dette valget frigjorde i tillegg flere GPIO-pinner på ESP32 for mulig videre funksjonalitet.

Motorene som ble brukt hadde 6-pins kontakt med separate ledninger for både motordrift og enkoder. Dette inkluderte egne tilkoblinger for motorens strømforsyning, jord og A- og B-signaler fra enkoderen. Denne løsningen bidro til en mer ryddig kabling, og gjorde det enkelt å koble fra enkeltmotorer ved behov. Det forenklet både montering og feilsøking, spesielt når det var nødvendig å verifisere at riktige motorer var koblet til riktige utganger på ESP32.

Drivsystemet og motorstyring

En av de største utfordringene gruppen møtte, var å få drivsystemet til å fungere optimalt. Vi hadde i utgangspunktet veldig begrenset erfaring med bruk av PID-regulator, som førte til at de første versjonene av programmene som ble testet resulterte i oscillerende og ustabil hjulbevegelse på roboten. Årsaken til dette fant vi ut etter en god stund var at vi forsøkte å bruke alle tre verdiene (PID) på en gang heller enn å starte med en og en verdi og jobbe oss ut fra dette. Løsningen ble å dele opp problemet og arbeide stegvis om hverandre for å få til et fungerende system.

PWM-signaler fra ESP32 ble brukt for å styre tre MX1616H-motordrivere, som igjen regulerte hver sin DC-motor med enkoder. Etter mye feilsøking og testing oppnådde vi en jevn og stabil bevegelse med lav feilmargin, men akselerasjonen forble fortsatt ustabil fra starten av kjøringen. Dette systemet fungerte godt innenfor rammene til prosjektet, men der var fortsatt potensial til forbedringer ved mer avanserte regulatorer og bedre filtrering av måledata.

Kiwidrive og bevegelsesmekanisme

Valget om å bruke kiwidrive ble tatt for å gi roboten full bevegelsesfrihet i alle retninger, men dette krevde imidlertid omfattende forståelse for prinsippene bak invers kinematikk og koordinering av hjulbevegelse. Det eksisterte lite tilgjengelig kode som passet vårt spesifikke oppsett, og det meste måtte derfor utvikles fra bunnen av med hjelp fra eksterne kilder og verktøy som ChatGPT.

Til slutt fikk vi systemet til å fungere noenlunde tilfredsstillende. Koden ble så kombinert med PID-regulering og kunne da tolke og utføre bevegelser basert på X-, Y- og rotasjonskommandoer.

IR-mottaker og fjernkontroll

Siden det var lite intuitivt å utføre testing og forbedring av kode ved å la roboten kjøre av seg selv, fant vi ut at det måtte utvikles kode for IR-mottaker og en fjernkontroll for å enklere kunne utføre tester på drivsystemet til roboten. Plasseringen til IR-mottakeren var i utgangspunktet ikke gunstig, og den ble flyttet underveis i prosjektet etter hvert som roboten ble utvidet med Lidar. Til slutt ble den plassert oppe på topplaten sammen med Lidar.

Koden vi brukte for å identifisere knappene på fjernkontrollen, fant vi på Arduino sine forum og brukte så ChatGPT for å forbedre denne koden til våres formål. Det ble laget en tabell i et tekstdokument der knappene ble notert ned, samt rådataen som ble sendt til den serielle monitoren i Arduino og deretter konvertert til brukbar kode som vist i teoridelen for IR-mottaker og fjernkontroll se IR-Mottaker og fjernkontroll:

Etter at de sentrale komponentene i den nedre delen av roboten og tilhørende kode var kommet til et fungerende og testbart nivå, ble selve roboten overlevert til gruppemedlemmene som arbeidet med den øvre delen. Dette ga samtidig rom for at de som hadde arbeidet med drivsystemet kunne begynne å dokumentere fremgangsmåten, erfaringene og de tekniske løsningene som var utviklet så langt i prosjektet.

6.3. Den øvre delen av roboten

I denne delen ser vi nærmere på LIDAR, ultralydsensor og kompass. Vi deler erfaringene våre med de tekniske utfordringer vi støttet på, både med komponentene og koden, og forklarer hvordan vi forsøkte å løse dem etter hvert.

Lidar

Implementeringen av Lidar mot en Raspberry pi 5 avdekket flere tekniske utfordringer knyttet til programvareavhengigheter og sanntidsdatavisualisering. Innledende ble det brukt produsentens eget program, Slamtec robostudio med drivere for å funksjonsteste Lidar før den ble installert på roboten, illustrert i Figur 30.

Da det var verifisert at den virket ble den koblet opp mot Raspberry Pi. Det ble opprettet et virtuelt miljø i terminalen og alle bibliotek ble lastet inn. For at vi skulle kunne ta lidar i bruk, var vi avhengig av å lage eget Python program i Spyder for lesing av data fra enheten.

En utfordring vi fikk var at scannet som ble returnert fra Lidar var speilvendt, illustrert i Figur 31. Men dette hindret oss ikke videre i prosjektet da den kun brukt til å måle avstanden til objekter rett fremfor roboten. Etter hvert som utviklingen av koden fortsatte, fikk vi til slutt LIDAR til å fungere med roboten og representere et bilde av omgivelsene, illustrert i figur 12, i resultatdelen Til tross for at vi hadde planlagt å bruke SLAM (Simultaneous Localization And Mapping), kom vi oss ikke i mål med denne funksjonen.

Strømforsyning

På den øvre delen ble det også plass til batterier og DC-DC konverter som vi har gått igjennom i teoridelen, se 4.7.2 -4.7.3. Ved test i begynnelsen var det store problemer med strøm til roboten, der den stoppet fort opp eller Raspberry slo seg av. Det ble brukt mye tid på å finne løsning til dette, og resultatet ble at det ble bestilt inn BMS system for å styre og overvåke batteriene. At vi måtte vente på deler gjorde at roboten ble stående uvirksom i en periode. Det ble i denne perioden koblet strøm kun til Raspberry for arbeid i operativsystemet og opp mot de andre sensorene.

Ultralydsensor

Selv om ultralydsensorene befinner seg på nedre del, ble det utført av prosjektmedlemmene med ansvar for øvre del, installasjon og testing av disse. Et problem som vi møtte på, var at echo-signalet fra ultralydsensorene er på 5 V og inngangene til Raspberry støtter kun 3.3 V. Det ble derfor gjort en beregning for å redusere spenningen og valgt motstander deretter. Ved drift av roboten kom det frem at den reagerte ustabilt når den skulle snu seg etter å ha møtt en hindring, det ble oppdaget at koblingen mellom den ene sensoren og Raspberry Pi 5 var feilkoblet med tanke på hvor roboten skulle snu ved hindrings deteksjon. Dette ble tatt tak i ved å gjøre om i koden slik at når roboten fikk sensor input, så trigget den en riktig respons som gjorde at den kjørte vekk fra hindringen, istedenfor mot den. Begrensninger i programmerings kunnskaper hos gruppe medlemmene, og bratt læringskurve, gjorde at dette tok lang tid å finne ut.

Linux

Raspbian som operativsystemet på Raspberry Pi heter, er som nevnt tidligere, bygget på en Linux-distribusjon. Dette var noe som var nytt for flere i prosjektet og tok tid å sette seg inn i. Men det er vanskelig å seg for seg at det var noe vei utenom Linux i et slikt oppsett.

Programmering

Ved programmering av Python-kode til roboten viste det seg raskt at det ble meget komplekst med kode til alle sensorene og samspillet mellom disse. Her kom vi fort til kort med våre kunnskaper innen programmering. Løsningen ble å bruke KI (Kunstig Intelligens) til kodehjelp, men det ble raskt klart at denne har sine begrensninger. Det ble mye prøving og feiling, og når man har samtale med KI over en lengre periode så blander den sammen og glemmer deler av tidligere problemstillinger. Dette førte til at den endret til tider ting som allerede fungerte. I begynnelsen ble alt laget i et enkelt Python-skript. Vi fikk roboten til å stoppe ved hindring for så å endre retning. Men når vi skulle legge til noen funksjoner, var det andre ting som ikke lengre fungerte. Senere ble det prøvd med å stykke opp programmet i flere Python-skript, men kompleksiteten gjorde at det ble vanskelig å få alle delene til å spille sammen. Programmeringen var en ganske sentral del av oppgaven og det ble tatt en avgjørelse samlet i gruppen at de andre prosjektmedlemmene skulle få teste ut sine ideer.

Kompass

Etter at roboten var kjørbar, ble kompass modulen bakt inn i selve koden. Kompasset vår, en HMC5883L skulle ifølge dokumentasjonen ha en adresse 0x1E, noe som ikke samsvarte med verdiene som vi fikk. Adressen som vi fikk var 0x0d, som vist i tabellen under kapitel om systemintegrasjon. Etter et kjapt søk på nettet fant vi at vi ikke var alene om denne erfaringen. Det viste seg at kompasset vårt ikke var orginalt kompass fra Honeywell, men en kopi. Vi fant at kompass QMC5883L hadde adressen 0x0d, når vi brukte den adressen til denne modellen virket kompasset. Etter at den var registrert med riktig adresse, måtte den kalibres slik at kompasset gav riktig retning når vi kjørte roboten. Kalibrering av kompass er viktig for å kompensere for magnetiske forstyrrelser og gi pålitelige resultater, den mest vanlige måten er å kjøre et eget testprogram for deretter bevege sensoren i en åttetallsbevegelse slik at den opplever alle retninger, dette var metoden vi landet på.

6.4. Kommunikasjon mellom nedre og øvre del

I dette kapittelet drøftes samspillet mellom de to delene av roboten. Den nedre, som stod for bevegelse og motorstyring, og den øvre, som styrte sensorer og den overordnede logikken. God kommunikasjon mellom disse var avgjørende for at roboten skulle kunne navigere presist og reagere hensiktsmessig på omgivelsene.

Samarbeid og overlevering mellom gruppene

Etter at arbeidet med ferdigstillingen av begge delene og roboten ble satt sammen fysisk, ble den overlevert til de som hadde jobbet mest med den øvre delen. På dette tidspunktet måtte begge gruppene ha innsikt i hverandres arbeid for å forstå hvordan samhandlingen skulle være. Dette innebar at koden for motorstyring og sensorhåndtering måtte forklares og forstås på tvers av gruppene før integrasjonsarbeidet kunne starte.

Teknisk sett måtte ESP32 og Raspberry Pi 5 kunne kommunisere, og det ble nødvendig å sette seg inn i hvordan serielle tilkoblinger og datadeling mellom enhetene fungerer. Denne kompetansen hadde ingen av gruppemedlemmene på forhånd, og det ble brukt en del tid på å tilegne seg kunnskap om dette gjennom nettressurser og verktøy som ChatGPT.

Utfordringer med fremdrift og organisering

I denne fasen oppstod flere praktiske utfordringer. Én gruppe tok med seg roboten hjem for å begynne på integrasjonsarbeidet. Samtidig oppstod det uventet arbeid og fravær blant medlemmene, som gjorde at fremdriften ble en del forsinket. Én måtte delta på et tre ukers kurs i utlandet, og en annen hadde arbeidstimer som begrenset kapasiteten.

Siden gruppemedlemmene bodde langt fra hverandre, var det heller ikke mulig å overlevere roboten fysisk før neste skolesamling. Dette førte til at gruppen uten roboten ikke kunne avslutte arbeidet med rapporten og ta over roboten.

Integrasjon og kodeutvikling

Koden som ble utviklet i første omgang bestod i større grad av testkode som var satt sammen fra ulike forsøk. Resultatet var en kodebase som ikke var oversiktlig, men klarte til dels å få roboten til å kjøre, unngå hindringer og stanse en gitt avstand fra en vegg. Det største problemet var at den var ustabil og utfordrende å videreutvikle.

Ved neste samling tok den andre gruppen over roboten og startet arbeidet med å rydde opp og omstrukturere koden. Det ble brukt verktøy som ChatGPT og Gemini aktivt i dette arbeidet, blant annet til å organisere koden i flere faner (filer) for hver enkelt komponent som motorstyring, ultralydsensor, Lidar og seriell kommunikasjon. Dette ble også gjort for å enklere holde oversikt over hvilken kode som tilhørte hva, heller enn å ha en hel kode på flere tusen linjer i en og samme fil.

Feil i orientering: IMU og kompass

En vesentlig feilkilde som tok lang tid å oppdage underveis. Var at både IMU og kompassmodulen var montert opp-ned. Dette påvirket hele bevegelsessystemet, ettersom alle orienteringsdata ble feiltolket. I tillegg ble kompasset i første omgang kalibrert og tatt i bruk uten å koordinere det med IMU-dataene.

Denne feilen førte til omfattende feilsøking og bortkastet tid i en allerede presset fase av prosjektet. Da den ble oppdaget, måtte all retningstenkning og deler av koden skrives om for å korrigere orienteringen.

Problemet med komponentene som var montert opp-ned ble løst ved at vi undersøkte hvordan dette skulle gjøres og byttet om på kode og fortegn i den matematiske delen av koden?

Kode for å invertere gyrosignalet

```
corrected_gyro_z = -(raw_gyro_z_deg_s - gyro_bias_z)
```

Videre i feilsøkingen og finjusteringen av IMU og kompass ble det valgt at roboten skulle orientere seg mot nord og deretter fortsette kjøringen samtidig som den detekterte og unnamanøvrerte når den møtte hindringer. Her ble det brukt en del tid på å legge inn og forandre på formler og verdier for en offset for at roboten skulle kjøre mot det virkelige nord og ikke det roboten selv mente var nord.

```
final_x_for_atan2 = compensated_x # Bruk kompensert +X som Nord-komponent
final_y_for_atan2 = -compensated_y # Bruk kompensert -Y som øst-komponent
```

Retningen nord ble valgt som utgangspunkt fordi problemstillingen for prosjektet var å få roboten til å finne et hjørne og parkere med en nøyaktighet på ±1 cm. Selv om funksjonene for sensorer, bevegelse og hinderdeteksjon etter hvert fungerte tilfredsstillende, ble det klart at tiden ikke ville strekke til for å finjustere systemet godt nok, eller få inn den resterende koden for å nå målet.

Oppsummering av integrasjonsarbeidet

Arbeidet med å få de to delene av roboten til å kommunisere og fungere sammen ble mer tidkrevende enn forventet. Dette skyldtes både manglende forkunnskap, tekniske utfordringer og uforutsette praktiske hindringer. Robotens funksjoner ble etter hvert samkjørt i et visst omfang, men presisjonskravet i prosjektbeskrivelsen ble ikke oppnådd.

6.5. Avsluttende refleksjon

Gjennom arbeidet med prosjektet opplevde vi at teori og praksis sjelden går hånd i hånd uten betydelige justeringer underveis. Selv med et tydelig definert mål og delmål, viste det seg at begrenset erfaring, tekniske utfordringer og praktiske forhold gjorde det vanskelig å oppnå dette innenfor prosjektets rammer.

Vi fikk systemene til å fungere hver for seg, og etter hvert samhandle i et visst omfang. Roboten ble autonom og kunne detektere hindringer og unngå disse, men kjøringen ble uten mål og mening. Likevel opplevde vi hvor krevende det er å utvikle en autonom robot som skal håndtere både navigasjon, sanntidsdata og motorstyring.

Selv om vi ikke oppnådde den ønskede presisjonen, har prosjektet gitt oss verdifull læring innen praktisk problemløsing, tverrfaglig samarbeid og utvikling av autonome systemer. Denne erfaringen danner grunnlaget for konklusjonen som følger.

7. Konklusjon

Hensikten med hovedprosjektet var å utvikle og programmere en autonom robotplattform, Lidarino, som skulle navigere i et kontrollert miljø, detektere og unngå hindringer, og til slutt parkere med en presisjon på ±1 cm i et utvalgt hjørne. Dette viste seg å være langt mer komplekst enn først antatt. Både tekniske og organisatoriske utfordringer gjorde at prosjektets fulle mål ikke ble realisert, men vi oppnådde likevel gode resultater.

Roboten ble bygget opp fra bunnen med både maskinvare og programvare utviklet internt i prosjektgruppen. Den fikk implementert funksjonalitet for autonom kjøring, hinderdeteksjon og unnamanøvrering. Vi fikk testet og brukt sensorer som Lidar, ultralyd, kompass og IMU, og benyttet PID-regulering for motorstyring. En del sentrale konsepter, som SLAM og presisjonsparkering, ble imidlertid ikke ferdigstilt. Dette skyldtes blant annet at vi undervurderte både tid og kompleksitet knyttet til integrasjon av alle komponentene – spesielt i samspillet mellom Raspberry Pi og ESP32 og medfølgende kode.

Gjennom prosjektet erfarte vi at feilsøking, testing og videreutvikling av autonome systemer krevde betydelig med tid og innsats. Begrensede forkunnskaper innen programmering og robotikk gjorde at store deler av arbeidet måtte baseres på selvstudier og iterativ problemløsing. Likevel tilegnet vi oss mye kunnskap om sensorteknologi, dataintegrasjon og sanntidsrespons, samt hvordan teori fra fagene faktisk anvendes i praksis.

Samarbeidet i prosjektgruppen var preget av både styrker og utfordringer. Til tider fungerte samarbeidet godt med åpen kommunikasjon og gjensidig støtte. Samtidig slet vi med å opprettholde felles fokus og jevn innsats. Enkelte perioder var preget av lav prioritering, ulik tilgjengelighet og lite fremdrift. Prosjektleder opplevde til tider at ansvaret ikke ble tatt like alvorlig av alle, og at mange avgjørelser måtte gjentas før de ble fulgt opp. Dette skapte frustrasjon og økt press, og i noen situasjoner påvirket det stemningen negativt. I ettertid ser vi at en tydeligere ansvarsfordeling og hyppigere statusmøter kunne bedret progresjonen.

Mot slutten av prosjektperioden ble innsatsen mer balansert, og samarbeidet styrket seg betraktelig. Gruppen samlet seg om å rydde opp i koden, feilsøkte robotens oppførsel og få systemene til å samhandle. Erfaringene vi har gjort oss, både når det gjelder prosjektledelse, teknisk problemløsning og teamarbeid, har gitt oss en ny forståelse for hvor krevende tverrfaglige utviklingsprosjekter kan være, og hvor viktig det er med struktur, åpenhet og kontinuerlig oppfølging.

Prosjektet har lagt et solid grunnlag for videre arbeid. Lidarino-plattformen kan videreutvikles av fremtidige studenter, for eksempel gjennom implementering av SLAM, forbedret posisjonskalibrering og navigasjon, eller kamera for visuell objektdeteksjon. I tillegg egner prosjektet seg godt som læringsplattform for forståelse av PID-regulering, sensordata og praktisk programmering.

Alt i alt har prosjektet gitt oss en dypere innsikt i hvordan autonome roboter bygges opp, ikke bare teknisk, men også organisatorisk. Vi har lært verdifulle ferdigheter vi tar med oss videre inn i utdanning og yrkesliv, spesielt innen teknologi, automatisering, systemintegrasjon og samarbeid.

8. Litteraturliste

Andrejašič, M. (2008). *MEMS Accelerometers. Seminaroppgave,*. Hentet fra Universitetet i Ljubljana: https://faculty.uml.edu/xwang/16.541/2010/MEMS_accelerometers.pdf

Brain, M., & Bowie, D. (2023, september 7). *How the Gyroscope Works*. Hentet mars 2025 fra HowStuffWorks: https://science.howstuffworks.com/gyroscope.htm

Dejan. (u.d.). *Ultrasonic Sensor HC-SR04 and Arduino – Complete Guide*. Hentet mars 2025 fra How to Mechatronics: https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/#google_vignette

ELECROW. (u.d.). *5A 180KHz 36V Buck DC to DC Converter.* Hentet mars 2025 fra Elecrow: https://www.elecrow.com/download/XL4015_datasheet.pdf?srsltid=AfmBOoru2I6C3fRgkvBRIJRs15C_KlB2JrNk4BQkQr6fzQmrZ-LSxBC2

Handson Technology. (u.d.). *HC-SR04 Ultrasonic Sensor Module User Guide.* Hentet fra www.handsontec.com: https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf

Honeywell. (u.d.). *Three-Axis Digital Compass IC.* Hentet mars 2025 fra Farnell: https://www.farnell.com/datasheets/1683374.pdf

Irazabal, J.-M., & Blozis, S. (2003, Mars 24). *I2C MANUAL*. Hentet fra https://www.nxp.com/docs/en/application-note/AN10216.pdf

Kistler. (u.d.). *MEMS accelerometer*. Hentet fra kistler.com: https://www.kistler.com/INT/en/mems-accelerometer/C00000759

Larsen, B. B. (2025, februar 22.). *mikrokontroller*. Hentet mai 3., 2025 fra Store norske leksikon: https://snl.no/mikrokontroller

Lenzi, F., & Giorgio, C. D. (2022). *MEMS and Microsensors*. Hentet fra https://valerionew.github.io/triennale-elettronica-polimi/4/MEMS/MEMS_and_Microsensors.pdf

MPS. (2025, mars 12). *Battery Balancing: A Crucial Function of Battery Management Systems*. Hentet mars 2025 fra MPS: https://www.monolithicpower.com/en/learning/resources/battery-balancing-a-crucial-function-of-battery-management-systems

Nabto. (u.d.). *ESP32 for IoT: A Complete Guide*. Hentet mars 2025 fra Nabto: https://www.nabto.com/guide-to-iot-esp-32/

Raspberry Pi Foundation. (u.d.). *Raspberry Pi 5*. Hentet 2025 fra Raspberrypi.com.

Siebeneicher, H. (2024, 9. 4.). *Universal Asynchronous Receiver-Transmitter (UART)*. Hentet fra Arduino.cc: https://docs.arduino.cc/learn/communication/uart/

SinotechMixic. (2022). *MX1616H dual H-bridge motor driver IC [datablad]*. Hentet fra https://xonstorage.blob.core.windows.net/pdf/mixic_mx1616h_apr22_xonlink.pdf

Slamtec. (2020, 10 15). *RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet*. Hentet fra Slamtec.com: https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf

Store norske leksikon. (2005-2007). *corioliskraften i Store norske leksikon*. Hentet mars 2025 fra snl.no: https://snl.no/corioliskraften

Veishida. (2023, mai). *What Is A DC Motor With Encoder, And How Does It Work?* Hentet fra Veishida: https://www.vsdmotor.com/news/what-is-a-dc-motor-with-encoder-and-how-does-67246435.html

Watson, J. (2016, Mars 1.). *MEMS Gyroscope Provides Precision Inertial Sensing in Harsh, High Temperature Environments*. Hentet Mars 2025 fra analog.com: https://www.analog.com/en/resources/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html

Wikipedia. (2025, Februar). *DC motor*. Hentet 2025 fra Wikipedia: https://en.wikipedia.org/wiki/DC_motor

Wikipedia. (2025). *Omni Weel*. Hentet fra https://en.wikipedia.org/wiki/Omni_wheel#:~:text=Omni%20wheels%20or%20poly%20wheels,slide%20laterally%20with%20great%20ease

Wikipedia contributors. (u.d.). *Gyroscope*. Hentet fra Wikipedia: https://en.wikipedia.org/wiki/Gyroscope

Wikipedia. (u.d.). *ESP32*. Hentet mars 2025 fra Wikipedia: https://en.wikipedia.org/wiki/ESP32

Wikipedia. (u.d.). *Holonomic constraints*. Hentet 2025 fra Wikipedia: https://en.wikipedia.org/wiki/Holonomic_constraints

Wikipedia. (u.d.). *Lidar*. Hentet fra Wikipedia: https://en.wikipedia.org/wiki/Lidar

Wikipedia. (u.d.). *Proportional–integral–derivative controller*. Hentet Mars 2025 fra Wikipedia: https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller

Wikipedia. (u.d.). *Pulsbreddemodulasjon*. Hentet Mars 2025 fra Wikipedia: https://no.wikipedia.org/wiki/Pulsbreddemodulasjon#:~:text=Pulsbreddemodulasjon%20eller%20pulsbreddemodulering%20(PBM%20eller,og%20derfor%20grunnfrekvensen)%20er%20konstant

8.1. Definisjoner og forkortelser

Forkortelse / Begrep	**Forklaring**
ESP32 / ESP32-WROOM-32	En mikrokontroller med innebygd Wi-Fi og Bluetooth, brukt til å styre elektroniske komponenter i sanntid.
GPIO (General Purpose Input/Output)	Universelle innganger og utganger på en mikrokontroller som kan programmeres til ulike formål.
PWM (Pulse Width Modulation)	En metode for å regulere spenning ved å sende raske pulser med varierende bredde. Brukes til å styre motorhastighet.
PID-regulator (Proportional, Integral, Derivative)	Et kontrollsystem som finjusterer motorbevegelse for å oppnå jevn og presis drift.
SLAM (Simultaneous Localization and Mapping)	En metode for å bygge et kart av omgivelsene samtidig som roboten lokaliserer seg selv i det.
I²C (Inter-Integrated Circuit)	En kommunikasjonsprotokoll som brukes mellom mikrokontrollere og sensorer over to ledninger.

UART (Universal Asynchronous Receiver-Transmitter)	En annen type kommunikasjonsprotokoll for dataoverføring via serielle porter.
DC-motor med enkoder	En motor som roterer ved likestrøm og har en innebygd sensor som måler rotasjonshastighet og posisjon.
BMS (Battery Management System)	Et system som overvåker og beskytter batteriet mot overlading, overutlading og kortslutning.
HAT (Hardware Attached on Top)	Ekstrautstyr eller moduler som kan monteres direkte på en Raspberry Pi for å utvide funksjonaliteten.
IR (Infrarød)	Usynlig lys brukt i fjernkontroller for å sende signaler til mottakere.
Omni Drive / Kiwi Drive	Et hjulsystem som gjør at en robot kan bevege seg i alle retninger uten å rotere først.
MEMS (Micro-Electro-Mechanical Systems)	Mikroskopiske sensorer brukt iblant annet gyroskop og akselerometre.
SoC (System-on-Chip)	En brikke som samler prosessor, minne og I/O på én fysisk enhet. Brukes i f.eks. ESP32 og Raspberry Pi.

Lidar (Light Detection and Ranging)	En sensor som bruker laser for å måle avstander og lage et kart av omgivelsene.
Gyroskop	En sensor som måler rotasjon og orientering.
Akselerometer	En sensor som måler akselerasjon (bevegelse) i én eller flere retninger.
Step-down omformer (Buck Converter)	En elektronisk krets som konverterer høyere spenning ned til et lavere nivå.
Kiwi Drive	En spesifikk type Omni Drive med tre hjul plassert med 120 graders mellomrom. Gir høy manøvrerbarhet.

8.2. Figurliste

Figur 1 Prinsippskisse over roboten 8

Figur 2 Viser prinsippet av et omni-hjul. Illustrasjon laget av prosjektgruppen. 9

Figur 3 Bilde av ESP 32 Foto: Prosjektgruppen 11

Figur 4 Oversikt GPIO hentet fra espressif.com 12

Figur 5 Raspberry Pi 5 Foto: Prosjektgruppen 13

Figur 6 Raspberry med GPIO, hentet fra pinout.ai/raspberry-pi-5 14

Figur 7 Gyroskopet Foto: Prosjektgruppen 15

Figur 8 Gyroskop med kardansk oppheng. Hentet fra snl.no 15

Figur 9 Innsiden av MEMS-gyroskopet MPU- 6050, her representert av en 3D printbar model fra printables.com 15

Figur 10 Bilde av lidaren. Foto: Prosjektgruppen 18

Figur 11 Lidarens virkemåte hentet fra Slamtec.com 18

Figur 12 Motordiver MX16161H Foto: Prosjektgruppen 19

Figur 13 Prinsippskisse av en DC-motor. Hentet fra Wikipedia. 22

Figur 14 Prinsippskisse av enkoder. Hentet fra encoder.com 22

Figur 15 Motoren med enkoder og omni-hjul.. Foto: Prosjektgruppen 23

Figur 16 Prinsippskisse av ultralydsensoren. Hentet fra howtomechatronics.com 25

Figur 17 Kobling av ultralydsensoren. Hentet fra howtomechatronics.com 25

Figur 18 Oppkobling av ultralydsensoren mot Arduino Uno. Hanson Technology 26

Figur 19 Kompasmodul HMC5883L Foto: Prosjektgrupen 27

Figur 20 Viser roboten under bygging. Foto: Prosjektgruppen 29

Figur 21 Viser enkoderene som er koblet til motorene. Foto: Proskjetgruppen 32

Figur 23 Bildet viser fjernkontrollen som ble brukt. Foto: Prosjektgruppen 38

Figur 22 Bilde viser IR mottaker for mottak av fjernkontollkommandoer. Foto: Prosjektgruppen 38

Figur 24 Prinsippskisse av PID prosessen. Illustrasjon latet av prosjektgruppen. 41

Figur 25 Viser roboten med batteriene, DC-DC konverter og Raspberry Pi. Foto: Proskjetgruppen 53

Figur 26 Batterisystemet tegnet ved hjelp av Cirkit Designer 54

Figur 27 Viser prinsippet for balansering av batterier (MPS, 2025). 54

Figur 28 Bildet over viser en overlapp av miljøet og Lidar avlesning med blå strek, og med ultralydsensorer med røde streker. Foto: Prosjektgruppen 57

Figur 29 Graff av PID loop kjøring på 1 hjul 58

Figur 30 testbilde fra Lidar i robostudio: 59

Figur 31 Scan gjort med python i rom på skolen. 59

Figur 32 Skjermdump av Influx db under testing av ultralydsensorene. 60

9. Vedlegg 1 Prosjektets GitHub

https://github.com/LidarinoAuto/SelfDriving

10. Vedlegg 2 Prosjektets Google drive

https://drive.google.com/drive/folders/1z9EGviIEr1ERyJ5F5aWWMVZ5W3uMAqVC?usp=drive_link

11. Vedlegg 3 koblingsskjema til Raspberry og ESP 32