

JetBalancer

Selvbalanserende og Autonom Robot

Bacheloroppgave

BO23-G26
Torstein Budberg
Marte Hatlevold
Robin Kaastrup
Sindre Valle Kopperud

Avdeling for Informasjonsteknologi
Høgskolen i Østfold
Halden
27. oktober 2023



Abstrakt

Følgende prosjektoppgave går ut på å ta utgangspunkt i en robot som allerede har kapasitet til å kjøre autonomt, og omforme den til en robot som også kan holde seg oppreist ved selvbalansering. For å løse oppgaven vil gruppen designe et nytt karosseri, samt løfte roboten opp på to hjul. Roboten det ble tatt utgangspunkt i er en JetBot, levert av WaveShare, som består av karosseri, to hjul og to støtter som holder den oppreist. Roboten som gruppen produserer går under navnet JetBalancer, hvis eneste kontaktpunkt med flaten den kjører på er to hjul.

Ideen for løsningsarkitektur kommer av at studentene i gruppen har fullført faget *ITD30019 - Digital Styring og Cyber-Fysiske Systemer*. I faget var en obligatorisk oppgave å få en to-hjulet robot til å balansere. Logikken og teknikken som brukes for å få en robot til å balansere er undervist i det overnevnte faget, i tillegg har studentene erfaring med konsepter innenfor matematikk og fysikk som brukes for å løse oppgaven.

Fra oppdragsgiver har gruppen blitt gitt stort spillerom når det gjelder løsning på oppgaven. De eneste kravene oppdragsgiver har satt til løsningen er at JetBalancer skal bruke Jetson Nano og at den skal stå på to hjul, alle andre avgjørelser som må tas er opp til gruppen.

Gjennom prosjektet vil det bli utført omfattende testing for å kunne avgjøre om komponenter og løsninger fungerer som de skal. Testing gjøres hyppig og resultater analyseres nøye. Prosjektet innehar gode muligheter for videreutvikling, dersom dette er ønskelig for oppdragsgiver og andre studentgrupper.

Takk til

Prosjektgruppen vil takke oppdragsgiver ved Fahad Said for tilliten han har vist gruppen gjennom prosjektet med å gi gruppen tilgang til MakerSpace sine verktøy og komponenter, samt oppfølgingen og tilbakemeldingene han har gitt gruppen. Gruppen vil også takke veileder Erling Petter Strand for hans gode oppfølging og tilbakemeldinger til gruppen og prosjektarbeidet. Videre vil gruppen også takke Maben Rabi og Christian Heide for gode råd og hjelp.

Det rettes også en takk til Kristoffer Kruithof og Sveinung Hatlevold for korrekturlesing og gode tilbakemeldinger på rapporten.

Innhold

Abstrakt	
Takk til	
Innhold	i
Figurer	iii
Tabeller	v
Kodesnutter	vii
Terminologi	ix
1 Introduksjon	1
1.1 Prosjektgruppen	1
1.2 Oppdragsgiver	1
1.3 Oppdrag	2
1.4 Formål	2
1.5 Leveranser	3
1.6 Metode	4
1.7 Prosjektplan	5
1.8 Gjennomføring	8
2 Teori og dens praktiske anvendelse	11
2.1 Oppgaven	11
2.2 Arbeidsmiljø	12
2.3 Maskinlæring	13
2.4 Autonome Kjøretøy	14
2.5 Nettverk	15
2.6 Styrings-/Kontrollsysten	17
2.7 Relevant Teknologi	30
3 Design	33
3.1 Deler fra JetBot	33
3.2 Deler gruppen har designet og produsert	35
3.3 Deler lagt til	41

4 Implementering og Testing	43
4.1 JetBot oppsett	43
4.2 Testing av JetBot	43
4.3 Testing av enkeltkomponenter	44
4.4 Prototype Karosseri	53
4.5 Kodelogikk	54
5 Diskusjon	59
5.1 Prosjektplan	59
5.2 Mål	65
5.3 Leveranser	65
5.4 Evaluering av arbeidet	66
5.5 Læringsutbytte	68
5.6 Hva kunne vært gjort annerledes?	69
5.7 Videre utvikling	70
6 Konklusjon	73
Litteraturliste	75
A Programkode for JetBalancer	77
B Arbeidstegninger for JetBalancer	83
C Bruksanvisning For JetBalancer	87
D Møtereferater	93
E Forprosjektsrapport	103

Figurer

1.1	Gantt-diagrammet til gruppen ved prosjektstart	6
2.1	Overordnet blokkdiagram av komponentene til JetBalancer.	12
2.2	Bilder av området som brukes til testing	12
2.3	Arkitektur på Convolved Neural Network[3]	14
2.4	Blokkdiagram av kommunikasjon mellom komponentene til JetBalancer . .	15
2.5	I ² C-buss til JetBalancer [4]	16
2.6	I ² C-skriv datapakke [4]	16
2.7	I ² C-les datapakke [4]	16
2.8	x-, y- og z-aksene på JetBalancer	17
2.9	Blokkdiagram med detaljer for kontrolleren	18
2.10	Invertet Pendel. Redigert fra [5]	19
2.11	JetBalancer i ustabil likevekt	20
2.12	JetBalancer i ustabil likevekt	21
2.13	Akselerasjon av JetBalancer i forhold til et referansesystem	22
2.14	Dreiemoment til JetBalancer	23
2.15	Eksempel på feedback system	24
2.16	Blokkdiagram av en PID kontroller, illustrerer Ligning 2.6 [7]	25
2.17	Behandling av sensordata	27
3.1	Oversikt av delene JetBot kom med. [12]	34
3.2	Fullstendig sammensatt karosseri og hjul	35
3.3	ETUI_1	36
3.4	ETUI_2	37
3.5	FORLENGER_FESTE	38
3.6	KAMERA_FESTE	38
3.7	Veltetur vektortegning	39
3.8	Hjul	40
3.9	Kontrollbordet med IMU-chip tilkoblet	41
4.1	Bilder hentet fra JetBots treningsdatasett	44
4.2	Bilder av jumper wire modifikasjoner	45
4.3	Kobling mellom NVIDIA Jetson Nano, Waveshare utvidelseskort og IMU .	46
4.4	Koblingsdiagram mellom NVIDIA Jetson Nano, Waveshare utvidelseskort og IMU	46
4.5	Variasjon på avlesningstid fra gyroskopet	48
4.6	Variasjon på avlesningstid fra akselerometeret.	48

4.7	Variasjon på avlesningstid fra akselerometeret og gyroskop.	49
4.8	Variasjon på avlesningstid fra IMU med motor på	49
4.9	Bilder av JetBot med modifikasjoner	50
4.10	Bilder av innsiden og slitasjen av girmotorene	51
4.11	Slark i hjulene på grunn av slitasje på motorene	52
4.12	Motorer som ble testet	52
4.13	Prototype karosseri av JetBalancer, med dekk i rødt	53
4.14	JetBalancer karosseri med støtter	54
4.15	Behandling av sensordata, inndelt i blokker	55
5.1	Oppdatert Ganntdiagram, 9 uker etter oppstart	60
5.2	Oppdatert Ganntdiagram, 14 uker etter oppstart	62
5.3	Oppdatert Ganntdiagram ved prosjektets ende	64

Tabeller

1.1 Risikovurdering for gjennomføring av prosjektet	9
4.1 Motordetaljer	52

Kodesnutter

4.1	Eksempel på bruk av dd	43
4.2	Kode for henting av tiden en avlesning fra sensorene tar	47
4.3	Kode for utregning av snitttid mellom avlesninger	47
4.4	Importerings av biblioteker og instansiering av objekter	54
4.5	Lesing av data fra IMU-enheten	55
4.6	Beregning av gyroskopets bias rundt y-aksen	55
4.7	Initialbetingelser for feedback systemt til programmet	56
4.8	Lesing og behandling av data fra gyroskop	56
4.9	lesing og behandling av data fra akselerometer	56
4.10	Komplementærfilter	57
4.11	PID implementert i Python-kode	57
A.1	Kodecelle 1	77
A.2	Kodecelle 2	78
A.3	Kodecelle 3	79
A.4	Utskrift fra kodecelle 3	80
A.5	Kodecelle 1	80
A.6	Kodecelle 2	80
A.7	Kodecelle 3	81

Terminologi

- AI** Artificial intelligence (kunstig intelligens). Kunstig intelligens er et begrep som blir brukt om datamaskiner som er i stand til å justere egen aktivitet, uten instruksjoner fra mennesker, og dermed fremstå som intelligente. Det finnes flere grener innenfor kunstig intelligens og gruppens løsning vil bruke en gren som heter maskinlæring, der en mengde data brukes til å trenne programmet til å bli mer intelligent.
- CAD** Forkortelse for Computer Aided Design, en samlebetegnelse for programvare som anvendes til å produsere digitale plantegninger.
- CNN** Forkortelse for Convolved Neural Network, en teknikk innenfor maskinlæring og ofte brukt til autonomi. Diskuteres i mer dybde i underseksjon 2.3.1.
- HiØ** Forkortelse på Høgskolen i Østfold. Inngår som oppdragsgiver, samt hovedarenaen som gruppen vil arbeide på.
- I²C** Forkortelse for *Inter-Intergrated Circuit*. Kommunikasjonsprotokoll til å sende data fra NVIDIA Jetson Nano til sensorer og motorkontroller. Beskrevet i mer dybde i underseksjon 2.5.1
- IIK** Forkortelse på Institutt for Informasjonsteknologi og Kommunikasjon. IIK er instituttet som gruppen og lab/test-miljøet har tilhørighet til.
- IMU** Forkortelse for Inertial Measurement Unit. Måler og rapporterer variabler som kraft/akselerasjon, vinkelhastighet, og orientering. De vanligste IMU består av akselerometer, gyroskop, og magnetometer. Diskuteres i mer dybde i underseksjon 2.6.1
- MDF** Forkortelse for Medium Density Fibreboard. Platene er laget av bartre, der en del av råvaren er rester fra annen treproduksjon. MDF passer godt til dypfresing da det ikke har noen bestemt fiberretning.
- MIPI CSI-2** Forkortelse for «Mobile Industry Processor Interface Camera Serial Interface 2». MIPI CSI-2 er en dataoverførings protokoll som brukes til å overføre bilde- og videodata fra en kameraenhet til en prosessor i mobile enheter og innebygde systemer. Protokollen er designet for å gi høyhastighetsdataoverføring med lavt strømforbruk.
- PID-regulator** Forkortelse for Proporsjonal Integral Derivasjon regulator. PID-regulator er et styringssystem hvor input justeres mot refanseverdi avhengig av hvor langt unna det er proporsjonalt; avvik over tid og hastigheten til endringen.
- TPU** Forkortelse for Termoplastisk Polyuretan. Fleksibelt materiale som føles og fungerer som myk gummi.

AI-Modell Programvare som er trent på et sett med data for å gjøre spesifikke oppgaver.
Et eksempel kan være; programvare som tar valg basert på informasjon fra kamera.

Akselerometer Et måleinstrument som måler akselerasjon og krefter indusert av tyngdekraft.
Diskuteres i mer dybde i seksjon 2.6.1

Gyroskop Et måleinstrument brukt for å måle vinkelendring hos objektet gyroskopet er
festet til. Diskuteres i mer dybde i seksjon 2.6.1

JetBalancer Gjennom rapporten vil det refereres til JetBalancer. Dette er en referanse til
hva gruppen produserer selv. JetBalancer vil være en selvbalanserende og autonom
robot som kjører på to hjul. Jetbalancer utarbeides fra *JetBot*, som er roboten
gruppen får utlevert ved begynnelsen av prosjektet.

JetBot Roboten som gruppen får utlevert ved prosjektstart. Roboten er levert av Waveshare,
Jet-delen av det sammensatte ordet refererer kun til at roboten tar i bruk *Jetson*
Nano. Dette betyr ikke at JetBot og JetBalancer nødvendigvis kommer fra samme
leverandør.

Jetson Nano Datamaskinen til JetBot. Levert av NVIDIA og må ikke forveksles med
JetBot som kommer fra en annen leverandør.

Magnetometer Et måleinstrument brukt for å måle det magnetiske feltet ved styrke
og/eller retning. Diskuteres i mer dybde i seksjon 2.6.1

Sensor Fusion Sensorfusjon er en metode som kombinerer avlesningsdata fra to eller flere
sensorer. Ved sensorfusjon er hensikten å redusere usikkerhet sammenlignet med hva
avlesningdata av de individuelle sensorene gir.

Kapittel 1

Introduksjon

I dette kapittelet kommer gruppen med en introduksjon av seg selv, oppdragsgiver, og oppdraget. Her presiseres også formål, leveranser og prosjektstruktur.

1.1 Prosjektgruppen

Gruppen består av fire studenter ved Høgskolen i Østfold, alle gruppemedlemmer går Bachelor i Ingeniørfag, data.

Torstein Budberg

Heltidsstudent fra Gudbrandsdalen. Tidligere utdanning i Fornybar energi fra Høgskolen i Gjøvik. Har erfaring innen Docker, virtualisering, nettverk, med mer.

Marte Hatlevold

Student fra Fredrikstad. Tidligere utdannelse i Samfunnsøkonomi fra USA. Jobber deltid som Junior Utvikler ved siden av studiene. Interesse for matematikk og utvikling.

Robin Kaastrup

Student fra Arendal. Går nå siste år på dataingeniør. Gikk Media og Kommunikasjon på videregående, fokuserte der på grafisk design. Faglige interesser er programmering og fabrikasjon.

Sindre Valle Kopperud

Heltidsstudent fra Fredrikstad som bor på Remmen. Går dataingeniørstudiet, som alle andre i gruppa. Interessert i matematikk og programmering, foretrekker programmeringsspråket Common Lisp.

1.2 Oppdragsgiver

Høgskolen i Østfold (HIØ), institutt for informasjonsteknologi og kommunikasjon (IIK), instituttet er et av tre ved Fakultet for informasjonsteknologi, ingeniørfag, og økonomi. Ved instituttet er det omtrent 55 ansatte, 1000 studenter og 15 stipendiater [1]. Høgskolens avdeling i Halden er lokalisert på Remmen.

Oppdraget er knyttet til høgskolens MakerSpace. MakerSpace er en elektronikk- og fabrikasjonslab ved HIØ og ble startet som et prosjekt ved IIK. Gjennom semesteret er området fritt tilgjengelig for studenter og ansatte ved IIK, og i enkelte perioder vil det være tilgjengelig for alle studenter og ansatte. MakerSpace holder en rekke kurs ved flere anledninger gjennom året og under studentenes sommerferie tilbyr de en sommerskole for barn og ungdom [2]

1.3 Oppdrag

Prosjektet har som formål å designe en selvbalanserende og autonom robot på to hjul. Autonom robot definerer gruppen som en robot som er kapabel til å gjenkjenne sine omgivelser og navigere uten menneskelig påvirkning. For å oppnå autonomi står gruppen fritt til å ta i bruk ferdige moduler levert av NVIDIA og/eller eventuelle åpne kilder.

For at JetBalancer skal kunne være selvbalanserende må gruppen implementere en løsning i både hardware og software som tillater JetBalancer å holde seg oppreist på to hjul uten menneskelig påvirkning.

Prosjektet er lagt opp til at det er stor vekt på det praktiske, men alle avgjørelser bør begrunnes med teori. Ettersom gruppen har fått stort spillerom hva det gjelder metode for løsning på oppgaven er det viktig at gruppen undersøker flere valg og støtter endelige valg med teori og analyse av fordeler og ulemper.

Gjennom prosjektet vil det være behov for at gruppen selv designer og implementerer et karosseri for JetBalancer. Det vil være nødvendig å diskutere alle delene som inngår i en slik løsning. Slike diskusjoner innefatter, men er ikke begrenset til, hjulopppheng, støtfanger, veltebur, og vektfordeling.

1.4 Formål

1.4.1 Prosjektmål

Hovedmål Produsere en robot på to hjul som er både autonom og selvbalanserende.

Delmål 1 Trene modell for autonom løsning, ferdig skrevet og utgitt av JetBot-produsent.

Delmål 2 Designe og implementere et karosseri for JetBalancer.

Delmål 3 Lage en selvbalanserende løsning.

Delmål 4 Se til at JetBalancer håndterer både selvbalansering og autonom kjøring.

Under *Delmål 1* defineres det å teste modell for autonom kjøring med JetBot som den er utlevert og med eksisterende moduler for autonom kjøring. Delmålet regnes som fullført når JetBot har blitt startet, det er oppnådd kontakt mellom datamaskin og JetBot, og har fått lastet opp modul for autonom kjøring, og autonom kjøring har blitt gjennomført.

Under *Delmål 2* defineres design som skisser gruppen har laget, implementering av design regnes som fysiske prototyper for ramme og hjul.

Under *Delmål 3* defineres det å lage en selvbalanserende løsning til prototypen som ble laget under Delmål 2. Delmålet ansees som oppnådd dersom JetBalancer balanserer på to hjul, og kan kjøre fremover, bakover og svinge.

Delmål 4 regnes som fullført når JetBalancer klarer å balansere på egenhånd og kjøre autonomt samtidig. Når Delmål 4 er fullført vil Hovedmålet også ansees som fullført.

1.4.2 Utbytte

JetBot anvendes allerede i fagene *Autonome Kjøretøy* og *Teknologiprosjekt* ved HiØ. Ved å endre på designet, får JetBot ny funksjonalitet og flere bruksområder, som igjen kan øke omfanget av disse fagene.

IIK har tidligere benyttet MakerSpace for å rekruttere nye studenter. Rekrutteringen har typisk blitt gjennomført ved at ansatte på makerspace deltar på messer og viser frem tidligere gjennomførte prosjekter, samt utstyr som HIØ besitter. JetBalancer vil bli et fint tillegg som kan trekke tilskuere og potensielle studenter. IIK har også et samarbeid med Halden kommune hvor det undervises i fabrikasjon og teknologi for grunnskoleelever fra 4. til 7. klasse. JetBalancer kan her benyttes til å engasjere fremtidige utviklere og ingeniører.

1.5 Leveranser

Dokumenter

Én av leveransene er denne prosjektrapporten, hvor det skal komme frem hvordan gruppen har samarbeidet, hvordan løsningen(e) har blitt testet, og hvilken teori gruppen har brukt for å begrunne sine valg. Gjennom prosjektet er det satt opp tidsfrister for rapporten, forprosjektrapporten skal leveres *20/01*, første utkast av prosjektrapporten skal leveres *10/03*, andre utkast *21/04*, og endelig prosjektrapport *22/05*.

I tillegg er det krav om at gruppen skal presentere sitt arbeid ved EXPO samt en presentasjon *12/06*. Gruppemedlemmene skal også levere individuelle refleksjonsnotater der de reflekterer over hvordan prosjektet har blitt gjennomført og deres egen innsats til prosjektet, denne innleveringen faller på samme dag som endelig versjon av rapporten *22/05*.

Software

Ved prosjektets oppstart vil gruppen få tilgang til en JetBot med tilhørende moduler som inneholder algoritmer for autonom kjøring, som bruker maskinlæring til bildegenkjenning. Gjennom prosjektet vil gruppen bli nødt til å lage sin egen programkode for balansering og avhengig av endelig design på hardware, samt strukturen på koden for balansering, kan det være at gruppen må endre på tidligere kode for autonom kjøring.

Gruppen skal levere programkode for JetBalancer, denne programkoden består av tre prinsipper; koden skal bestå av kommandoer og funksjoner som gjør at JetBalancer balanserer uten påvirkning utenifra, koden skal bestå av kommandoer og funksjoner som gjør at den kjører autonomt, til sist skal koden bestå av kommandoer og funksjoner som gjør at JetBalancer balanserer og kjører autonom samtidig.

Hardware

Da roboten som gruppen får utlevert i starten av prosjektet ikke balanserer på to hjul vil det være opp til gruppen å designe og implementere hvordan en slik to-hjulet robot vil se ut. Hardware kan deles opp i to deler; *kontrollbord og kjøretøy*.

Kontrollbordet innebefatter sensorer for posisjonering, bevegelse, hindringer, og lignende. Det finnes mange muligheter for hvilke sensorer som kan tas i bruk og detaljene for hvilke

som skal brukes vil komme frem under testing, men foreløpig har gruppen kun antagelser på hva som er fornuftig. Mulige valg av sensorer er en IMU for å måle orienteringen til JetBalancer, og kameraet som følger med JetBot til diverse bildejenkjenning.

kjøretøy innebefatter hjul, støtfangere, og annen fysisk design. Oppdragsgiver ønsker en robot som er i stand til å balansere på to hjul og som bruker Jetson Nano, annet design og funksjonalitet er opp til gruppen å bestemme.

Instruksjonsbok

Da mye av JetBalancer er designet selv vil det være aktuelt for gruppen å lage en egen instruksjonsbok. Instruksjonsboken skal inneholde forklaringer av oppsett og montering av JetBalancer, samt materiale og verktøy som kreves, dersom en annen ønsker å bygge sin egen JetBalancer ut ifra en Jetbot.

1.6 Metode

For å bygge JetBalancer skal gruppen analysere hva som behøves av en tohjulet selv-balanserende robot.

Analyse

Under analyse vil gruppen undersøke lignende løsninger. Ved en slik analyse håper gruppen på å få en bedre forståelse av hva som behøves for en selvbalanserende og autonom robot. Dette er informasjon som danner grunnlaget for design, og som tas med videre til testing.

Under analyse vil gruppen også undersøke teknologier som trengs for å kunne implementere den funksjonaliteten som trengs for å oppfylle kravene i oppgaven.

Teori

Gjennom teori vil gruppen selv danne en forståelse av hva som trengs for å gjennomføre prosjektet. Teorien vil bli diskutert både som overblikk og detaljert gjennom oppgaven.

Testing

Gjennom prosjektet vil det være behov for å teste ved flere stadier. Under testing vil det være fokus på funksjonaliteten til de forskjellige komponentene, samt oppsett/design av JetBalancer. Før gruppen designer JetBalancer vil det testes for autonom kjøring med JetBot som gruppen får utlevert.

Testing av hardware vil være testing av komponenter som hjul, både oppsett og størrelse, sensorer, motor, og materiale som brukes til kjøretøyet.

Testingen av software er en prosess der mange tester kan bli ført i løpet av en dag, noen ganger vil det være større feilrettinger og funksjonaliteter, andre ganger vil man teste om og om igjen for å kunne rette opp i små feil. Elementer som det vil testes for under testing av software vil være balanseegenskaper, autonomkjøring, kjøreegenskaper på to hjul. Å få JetBalancer til å kjøre autonomt samtidig som den balanserer på to hjul vil også være noe som må testes, da dette er ønsket sluttprodukt.

Produktet regnes som ferdig testet når det oppfyller kravene satt i beskrivelsen av oppgaven og i delmålene.

Samarbeid med oppdragsgiver

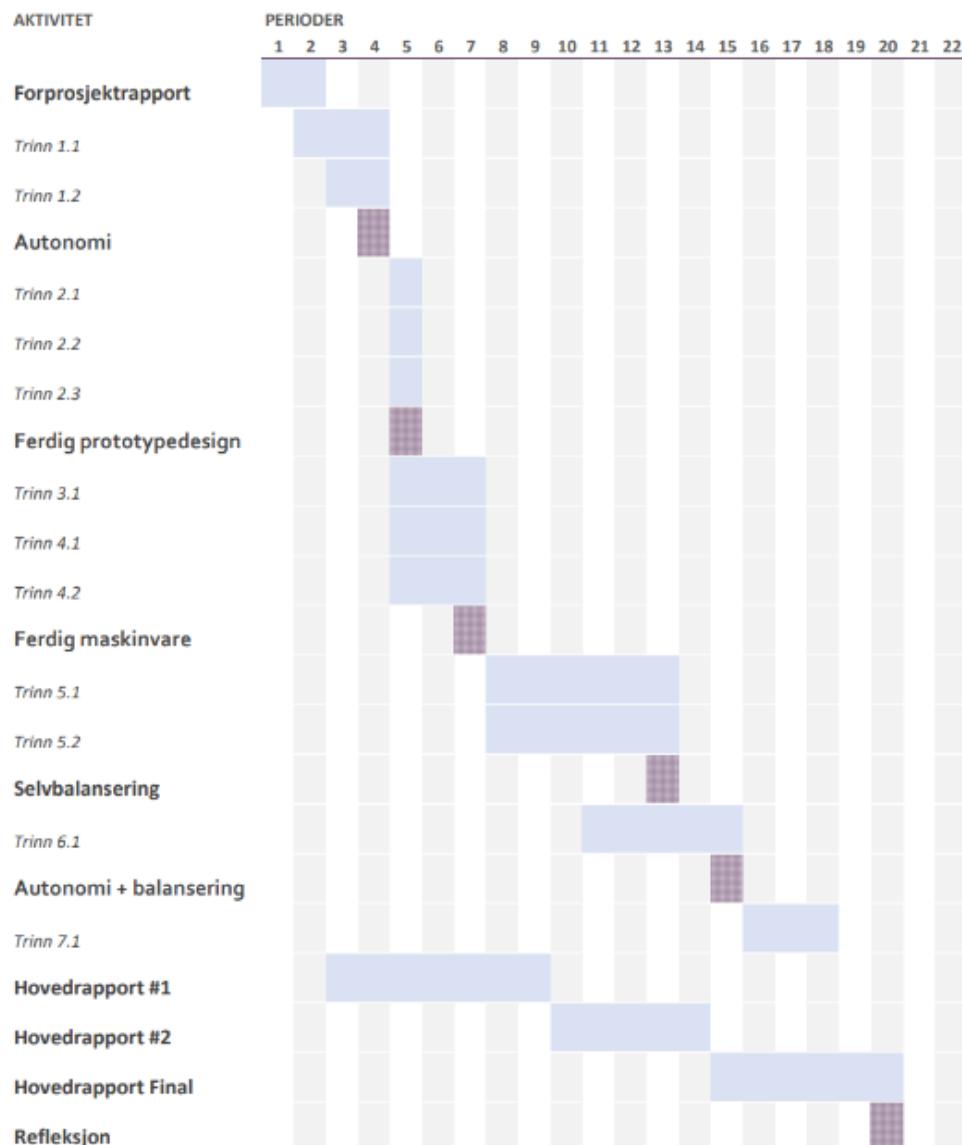
Som en del av prosjektarbeidet vil gruppen ha tett samarbeid med oppdragsgiver hvor de kan få tilbakemeldinger på nødvendige endringer. Det vil også være nødvendig for gruppen å gjøre en befaring ved testområdet. Dersom JetBalancer brukes videre i andre fag vil området som gruppen bruker til testing også brukes ved undervisning, derfor vil det være nødvendig at JetBalancer er tilpasset de tilhørende dimensjonene ved dette området.

1.7 Prosjektplan

Seksjon der gruppen diskuterer hvordan de planlegger tidsbruken og fremgangen på ulike elementer under prosjektet. Her presenterer gruppen et Gantt-diagram samt diskuterer hvilke hendelser som kan forårsake problemer eller forsinkelser i prosjektet.

Aktivitetene og trinn diskutert i denne seksjonen er tett koblet til delmålene diskutert i seksjon 1.4.

1.7.1 Gantt-diagram



Figur 1.1: Gantt-diagrammet til gruppen ved prosjektstart

1.7.2 Beskrivelse av Gantt

Under Gantt-diagrammet tilsvarer en periode én uke, med start i kalenderuke 2.

Aktivitet 1: Autonomi og forskning

Trinn 1.1 Sette opp JetBot med AI-modeller for linjefølging og kollisjonsavvergning

Finne ut hva som kan forventes av modellene med tanke på signaler til motorstyringen.

Trinn 1.2 Forarbeid. Analysere hva som er gjort tidligere av balanserende roboter hvilke sensorer og aktuatorer gir gode resultater.

Ved beskrivelsen gode resultater menes: Kan stå rolig på et punkt, snu på et punkt, kjøre frem og tilbake på kommando, holde balansen dersom den blir utsatt for ytre påkjenninger.

Aktivitet 2: Prototypedesign

Trinn 2.1 Velge hvilke typer eller modeller av sensorer som skal brukes.

Trinn 2.2 Første design av kjøretøyet

Trinn 2.3 Design arkitektur av kjørealgoritme og kommunikasjon mellom motorstyring/balansealgoritme og AI-modeller.

Forslag til hvordan signaler fra AI-modellene skal håndteres av balansealgoritmen.

Aktivitet 3

Trinn 3.1 Koblingsskjema av Jetson Nano og sensorer.

Tørr-test uten montering. Fungerer motorene som de skal? Teller sensorene riktig?

Aktivitet 4: Prototypefabrikering

Trinn 4.1 Montering

Trinn 4.2 Redesign av kjøretøy, hjul etc.

Ved 3D-printing eller CNC-fresing av deler.

Aktivitet 5: Selvbalansering

Trinn 5.1 Koding og testing av balansering

Trinn 5.2 Fullføre implementasjon av bildegjenkjenning

Aktivitet 6: Autonomi & Selvbalansering

Trinn 6.1 Få modellene til å kommunisere med balanseringsalgoritmen

Aktivitet 7: Finpussing

Trinn 7.1 Eventuell forbedring av bildegjenkjenning og balansering

1.8 Gjennomføring

Forklaring av strukturen til gruppen gjennom gjennomføringen av prosjektet.

1.8.1 Møter

I løpet av prosjektarbeidet er det avtalt to faste dager i uken der gruppemedlemmene møtes og jobber med prosjektet i fellesskap. Arbeidet som foregår under møtene vil innebære testing av løsning(er), samt skriving på rapporten. Det legges også opp til at gruppemedlemmene jobber individuelt og deretter diskuteres ideer i plenum ved de ukentlige møtene. Når gruppen ikke jobber sammen fysisk på HiØs lokaler vil kommunikasjon mellom gruppemedlemmene foregå hovedsakelig over gruppechat i tjenesten Discord.

En av de to dagene som gruppen møtes avtales det møter med veileder. Gruppen vil ha møte med veileder en gang i uken hvor det diskuteres hva som har blitt gjort i uken som har vært, og hva gruppen planlegger for uken som kommer. Ved møtene vil gruppen også få innspill til rapportskriving og det vil være mulighet for innspill på design og funksjonalitet på JetBalancer.

Oppdragsgiver har bedt om møter annenhver uke, disse møtene er fastsatt annenhver mandag. Oppdragsgiver stiller seg også tilgjengelig for teknisk hjelp når det trengs, da han ofte er til stede på labområdene som vil bli brukt til testing.

1.8.2 Risikovurdering

Hendelse	Effekt	Konsekvens
Sykdom (lengre periode)	Sykt gruppemedlem vil ikke ha muligheten til å arbeide med prosjektet	Resten av gruppen må gjøre mer arbeid for å få fullført prosjektet i tide
Mangel på deler	Mulig løsning som avhenger av denne delen blir ikke testet.	Prosjektet blir forsinket
Del er bestilt, men ikke levert i tide (før prosjektet avslutes)	Mulig løsning som avhenger av denne delen blir ikke testet	Sluttproduktet blir ufullstendig, da en løsning gruppen mener er god ikke blir implementert
Tap av data (for eksempel kildekode)	Data må gjenopprettes fra sikkerhetskopi eller rekonstrueres	Dersom vi har sikkerhetskopi, ingenting. Ellers, voldsom forsinkelse.
Ødelagte deler	Dersom delen er viktig for funksjonaliteten til JetBalancer vil testing stoppe opp	Prosjektet blir forsinket, mulig at prosjektet ikke blir testet grundig nok
Feilprint	Ved 3D-printing er det fare for at printeren feiler, eller at vi legger inn feil mål i filen som skal printes.	Feilprint vil utsette prosjektet. Hvor forsinket avhenger av størrelsen på printet. Forsinkelsen kan variere fra 30 minutter til en hel dag.

Tabell 1.1: Risikovurdering for gjennomføring av prosjektet

Kapittel 2

Teori og dens praktiske anvendelse

Under dette kapittelet vil gruppen analysere oppgaven og diskutere teori rundt temaer relatert til oppgaven. Dette forarbeidet innebærer forståelse av både selvbalansering og autonomitet, samt hvordan dette ville fungert på en to-hjulet robot.

Relevant teori rundt oppgaven vil diskuteres i dybden, og det vil forklares hvordan teorien anvendes praktisk i løsningen av oppgaven.

2.1 Oppgaven

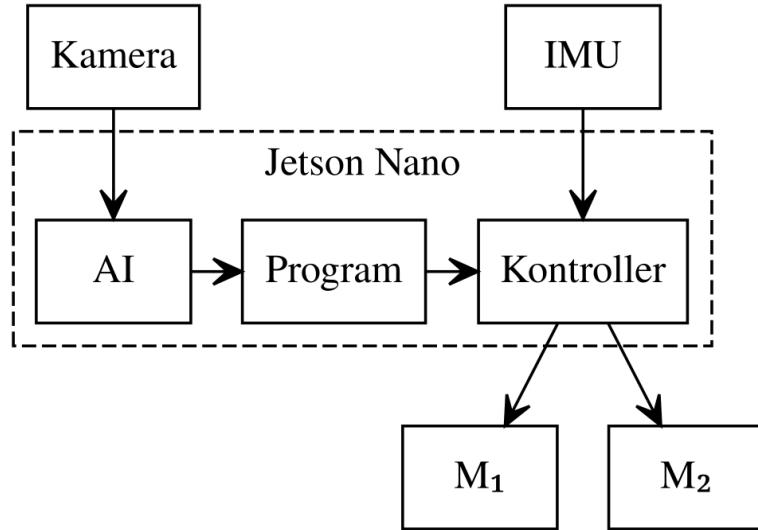
Som beskrevet i seksjon 1.3 skal gruppen designe en to-hjulet robot som er selvbalanserende og autonom.

Gjennom oppgaven må gruppen selv gjøre vurdering for hva som er det beste designet og hvilke komponenter som passer best til jobben som skal gjøres. Komponenter involverer hjul, sensorer, motor, batterier og mer.

Alle fire studentene har tidligere hatt faget *ITD30019 Digital Styring og Cyber Fysiske Systemer*, der studentene måtte fullføre en prosjektoppgave som omhandlet selvbalansering og løsning av problemstillingen *invertert pendel*. Ved gjennomføring av faget og prosjektet fikk studentene erfaring med bruk av gyroskop, akselerometer, og en PID-kontroller, som er kunnskap studentene kan anvende i denne bacheloroppgaven og for å få til en selvbalanserende funksjon.

Oppgaven kan deles opp i to hovedkomponenter; *maskinlæring* og *kontrollsystemer*. Maskinlæring vil bli brukt for autonom kjøring mens kontrollsystemer tas i bruk for å få til en fungerende selvbalanserings-algoritme.

I sekksjonene som følger vil gruppen diskutere teori og komponenter som til sammen danner oppsett og funksjonalitet for JetBalancer. I Figur 2.1 vil **AI** inneholde maskinlæring, diskutert i seksjon 2.3. **IMU** og **Kontroller**, og hvordan IMU kommuniserer med Jetson Nano blir diskutert i henholdsvis underseksjon 2.6.1, seksjon 2.6, og underseksjon 2.5.1. Program er software, M₁ og M₂ er girmotorene.



Figur 2.1: Overordnet blokkskjema av komponentene til JetBalancer.

2.2 Arbeidsmiljø

Arbeidsmiljø for JetBalancer består av både miljøet den testes i under prosjektet, og av miljøet som den skal brukes i etter prosjektet.

2.2.1 Miljøet gruppen tester i

MakerSpace har en egen testbane. Testområdet er svarte gummimatter hvor det er blitt brukt teip for å tegne opp kjørebane med veioppmerkning. Hele veibanen (to kjørefelt) har en bredde på 445[mm], én kjørebane har en bredde på 210[mm]. På grunn av bredden til kjørefeltet er designet på JetBalancer begrenset til en bredde under 210[mm]. Gummimattene som danner testbanen er også funksjonell da hjulene som 3D-printes ikke danner nok friksjon med glatt underlag.



(a) Oversikt



(b) Nærbilde

Figur 2.2: Bilder av området som brukes til testing

2.2.2 Miljøet roboten kan brukes i

JetBalancer skal tas i bruk i arbeidsmiljøer lignende det gruppen tester i. Ved autonom kjøring vil programmet være avhengig av referansepunkter, det vil derfor være best å bruke roboten i miljøer der det finnes naturlige referansepunkter, som veioppmerking.

2.3 Maskinlæring

Maskinlæringsteori brukes i koblingen mellom kamera og AI. Fra blokkdiagrammet i Figur 2.1 kan en se at kameraet sender informasjon til AI. I AI-blokken ligger det program for å analysere bilder som sendes fra kamera. Informasjon som analyseres i AI-blokken er om det ligger en blokkering i veien der JetBalancer har lyst til å kjøre. For bestemmelser av hva som regnes som blokkeringer er AI avhengig av at gruppen trener opp modellen.

For bildegenkjenning og autonom kjøring må studentene ta i bruk maskinlæring. Maskinlæring er en gren innenfor kunstig intelligens hvor en datamaskin ikke blir programmert, men heller opplært ved hjelp av et datasett. Datasettet som dannes består av et treningssett og et testsett. Ved *veiledet trening* vil datasettet inneholde både inngangsverdier og utgangsverdier, dette kan være bildegenkjenning der datamaskinen på en robot lærer og analysere situasjoner. Prosjektet tar i bruk maskinlæring ved autonom kjøring.

For bildegenkjenning og analyse er det allerede implementert et bibliotek som igjen tar i bruk et nevralt nettverk. Det finnes flere typer nevrale nettverk, biblioteket som kommer med Jetson tar i bruk et konvolusjonelt nevralt nettverk, som blir gjennomgått i mer detalj under.

2.3.1 Convolutional Neural Network

Et **nevralt nettverk** er en teknikk innenfor maskinlæring som analyserer data gjennom et nettverk av beslutningslag. Et nevralt nettverk består typisk av tre typer lag; input, hidden, og output. Lag én (*input*) tar i mot data, hvor funksjoner blir oppdaget. Lag to (*hidden*) vil analysere og prosessere egenskapene i input-data. Lag tre (*output*) vil vise det endelige resultatet som output.

I modulene som følger med JetBot tas det i bruk en type nevralt nettverk kalt *Convolutional Neral Network* (CNN). CNN bruker dyplæringsalgoritme og blir brukt spesifikt for bildegenkjenning. Algoritmen vil ta et bilde, lese bildet, og innad i bildet vil forskjellige objekter bli gitt forskjellig vekt, som vil gjøre at algoritmen klarer å skille objekter fra hverandre. Det nevrale nettverket som brukes her er en type *klassifisering*, og forhåndsproseseringen som er nødvendig i CNN er lavere enn ved andre klassifiseringsalgoritmer.

Når en trener opp CNN trenger man referanser. Mulige referanser kan være veioppmerking, objekter i veien, eller skilt.

CNN er ofte brukt til å konstruere algoritmer innenfor deep learning, og er spesielt nyttig for bildegenkjenning og oppgaver som innebærer prosessering av pikseldata. For autonome kjøretøy er CNN optimalt å bruke på grunn av dens funksjonalitet for gjennkjennelse av objekter. For generelle nevrale nettverk er mye av bildegenkjenningen gjort med lineær algebra og matrisemultiplikasjon. For bildegenkjenning i CNN brukes konvolusjon i stedet for matrisemultiplikasjon. Arkitekturen til CNN er delt opp i tre lag; *convolutional*, *pooling*, og *fully connected layer*. En illustrasjon på arkitekturen i CNN er illustrert under i Figur 2.3

Convolutional layer

Dette laget er en viktig byggstein til CNN, samt laget der det meste av beregninger blir gjort. I konvolusjonsprosessen vil en kjerne eller et filter lese de mottakelige feltene for å sjekke om det er tilstede en funksjon i bildet. Konvolusjonsprosessen itereres, og etter hver iterasjon vil det regnes ut et prikkprodukt mellom input piksler og filteret/kjernen.

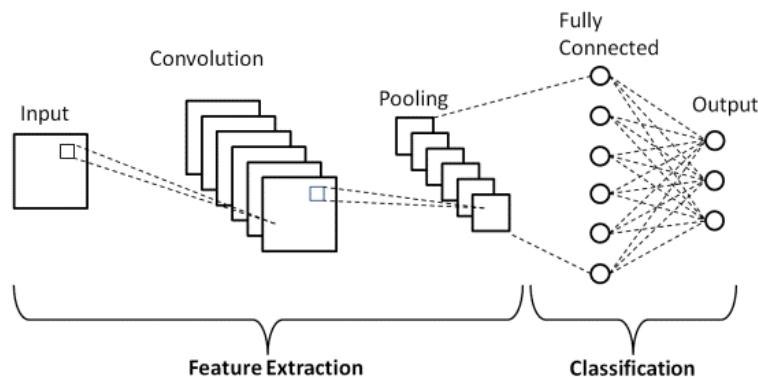
Det endelige output fra konvolusjonslaget kalles et *feature map*, og består av de numeriske verdiene som kommer av alle prikkproduktene som blir regnet ut. Det er de numeriske verdiene som blir regnet ut fra prikkproduktet som gjør at CNN kan tolke bildet og hente ut relevante data.

Pooling layer

I det sammenslående laget brukes det også et filter som leser et input bilde, men i motsetning til konvolusjonslaget blir antall parametere i input redusert. En reduksjon i antall parametere gjør at systemet blir mindre komplekst og øker effektiviteten, samtidig vil en slik reduksjon forårsake noe tap av informasjon.

Fully connected layer

Det er i dette laget at selve klassifiseringen basert på funksjonene som er blitt hentet fra de foregående lagene blir gjort. Fullt tilkoblet viser til at alle input fra et lag er koblet til hver aktivieringsenhet i det neste laget. Det er kun implementert et fullt tilkoblet lag ettersom en struktur der alle lag var fullt tilkoblet ville blitt et umødvendig tett nettverk, og ville vært beregningsmessig dyrt. Et tett nettverk vil også påvirke kvaliteten på output samt økt mengden tapt data.



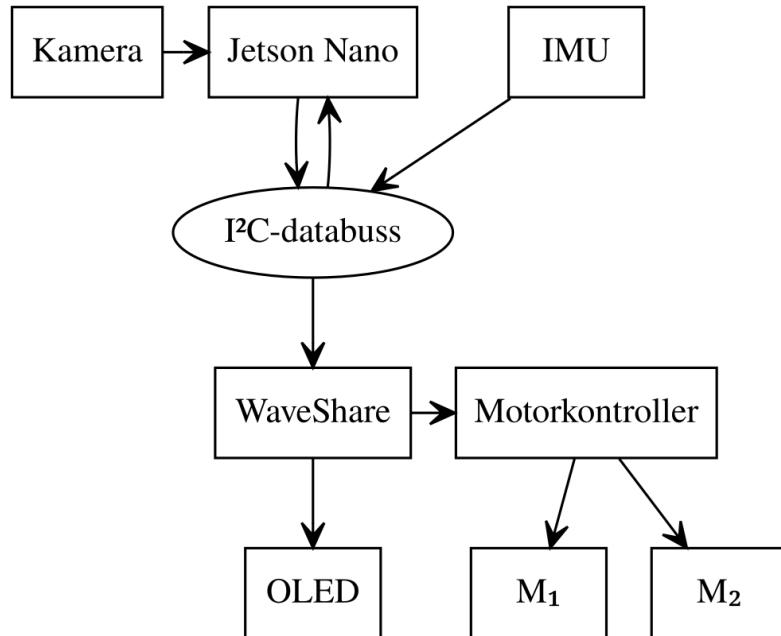
Figur 2.3: Arkitektur på Convolted Neural Network[3]

2.4 Autonome Kjøretøy

JetBot som ble utlevert i begynnelsen av prosjektet kom med ferdige moduler for autonom kjøring. Gruppen testet at modulene fungerte i begynnelsen av prosjektet, før det ble gjort modifikasjoner på karosseriet. Ved å teste funksjonaliteten til modulene som følger med blir det mulig både å kunne teste at delene på JetBot fungerer som de skal—og at det er mulig å bruke disse i prosjektet, samtidig vil det være mulig å ta med mye av denne autonome funksjonaliteten videre i JetBalancer.

2.5 Nettverk

I^2C vil i dette prosjektet bli brukt til nettverket for komponentene til JetBalancer. Jetson Nano vil være master, mens OLED-skjerm, IMU og motorkontrollerene er satt som slaver. Kameraet som følger med JetBot kommuniserer med Jetson nano via MIPI CSI-2 protokoll og er ikke en del av I^2C -nettverket.

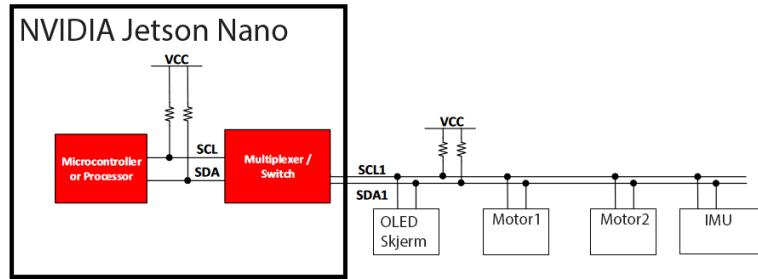


Figur 2.4: Blokkskjema av kommunikasjon mellom komponentene til JetBalancer

For at alle komponentene til JetBalancer skal kunne kommunisere med Jetson Nano planlegger gruppen å benytte I^2C -protokollen som et internt nettverk. Et overordnet diagram over nettverket er illustrert i Figur 2.4.

2.5.1 I^2C – Inter-Integrated Circuit

I^2C er en veletablert synkronisert seriell kommunikasjonsprotokoll som er mye benyttet til kommunikasjon mellom små elektroniske komponenter over korte avstander. I^2C har et bussgrensesnitt med to ledere, hvor en leder er for klokkepuls og en for dataoverføring.

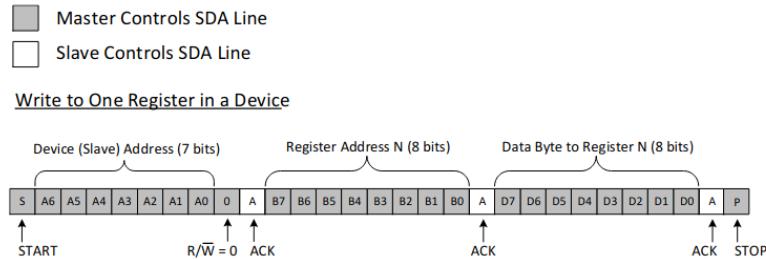
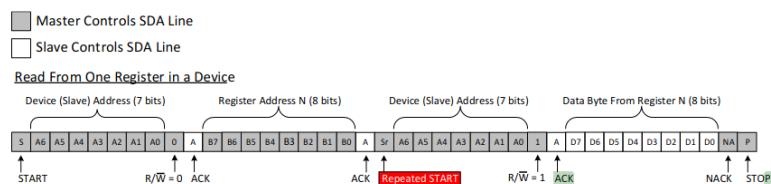
Figur 2.5: I²C-buss til JetBalancer [4]

SCL- og SDA-ledere

Serial Clock (SCL) er en fysisk leder som er dedikert for klokkepulser. Klokkefrekvensen på en I²C-buss bestemmes av en mester-enhet. Serial Data (SDA) er en fysisk leder som benyttes til å sende pakker mellom mester og slave enhetene.

Datapakker

Det er to typer I²C-datapakker. En type er for å skrive til en gitt minneadresse til en slave (Write) Figur 2.6. Den andre pakketypen er for å lese fra en gitt minneadresse til en slave (Read) Figur 2.7.

Figur 2.6: I²C-skriv datapakke [4]Figur 2.7: I²C-les datapakke [4]

Mester- og Slave-enheter

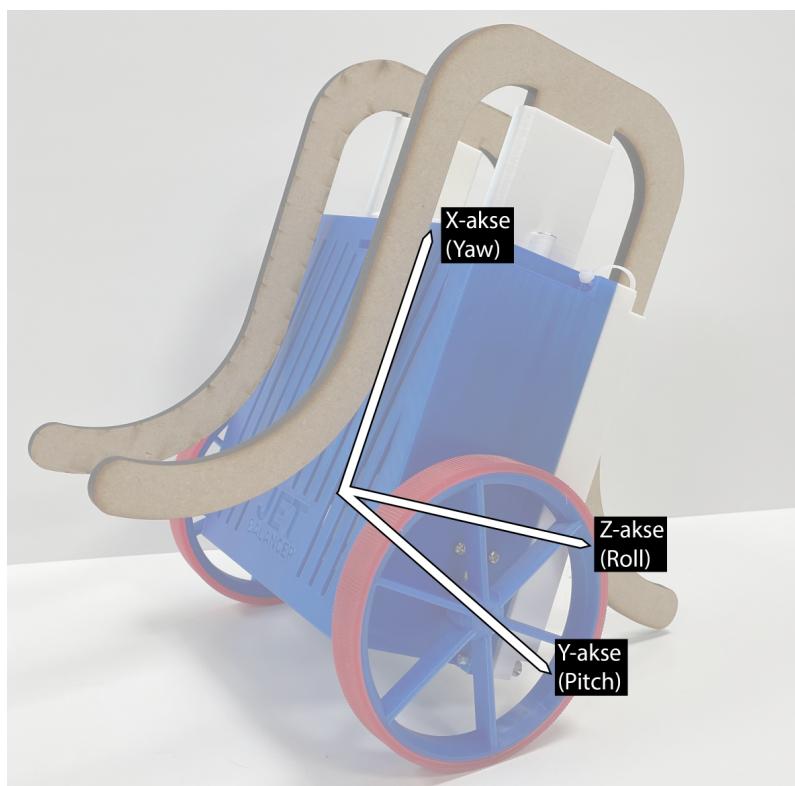
Det finnes to type enheter langs en I²C-buss, mester og slave. En mester bestemmer klokkepulsene, mens slavene har hver sin unike adresse på bussen. Slaveadressen er et sjusifret binærtall, ofte skrevet som tosifret heksadesimal.

2.6 Styrings-/Kontrollsysten

I Figur 2.1 ligger det en blokk *Kontroller*, denne blokken inneholder logikken som brukes til balansering. Logikken får informasjon fra *IMU*, og sender informasjon videre til motorene M_1 og M_2 . Innholdet i *Kontroller* blir diskutert i underseksjon 2.6.5.

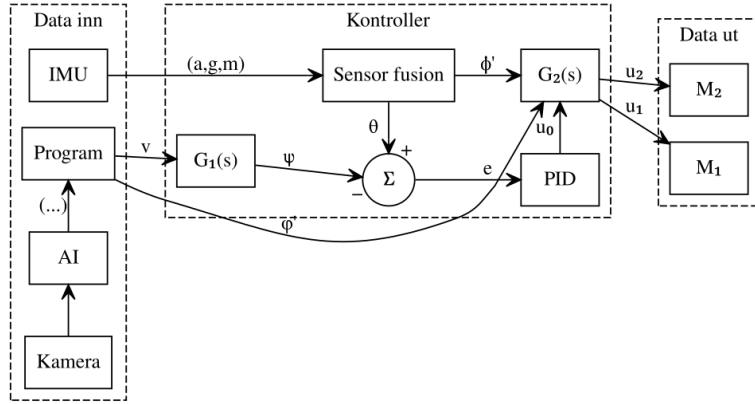
Styringsystem i denne sammenheng er et sett med elektroniske sensorer som samspiller, som igjen styrer/kontrollerer et sett med aktuatorer/motorer for å gjøre et helt system stabilt. Å lage et system for stabilitet er nødvendig i en hver selvbalanserende robot da balansering er i seg selv et ustabilt system. Hovedvekten av løsningen til gruppen ligger i selvbalanseringsfunksjonen til JetBalancer. For å kunne drive nøyaktig selvbalansering trenger man først å diskutere de forskjellige komponentene som utgjør en slik funksjonalitet.

Gjennom rapporten er det referanser til yaw-, pitch-, og roll-rotasjoner. Følgende er en illustrasjon av koordinatsystemet til IMU-enheten, aksene i systemet og hvilken akse som roteres om i de tre tilfellene.



Figur 2.8: x-, y- og z-aksene på JetBalancer

Figur 2.8 viser retningen til aksene i IMU-enhetens koordinatsystem, og kan brukes som referanse i underseksjonene som følger etter denne introduksjonen.



Figur 2.9: Blokkskjema med detaljer for kontrolleren

Figur 2.9 viser et mer detaljert bilde av hvilke komponenter som tar del i funksjonaliteten til JetBalancer. Fra **IMU** kan en se at det sendes data til **Sensor Fusion**, denne dataen består av a , g , og m , som står for akselerometer, gyroskop og magnetometer. Beskrivelsen av IMU og dens komponenter ligger i underseksjon 2.6.1. **PID** blir diskutert med detaljer og utregning i underseksjon 2.6.5. Hvordan **Sensor fusion** blir brukt og hvorfor dette er nødvendig blir diskutert i underseksjon 2.6.8

2.6.1 IMU

Inertial Measurement Unit (IMU) er en samling av sensorer som måler ulike type krefter; centrifugalkraft(gyroeffekt), akselerasjon, og ofte også magnetfelt. Ikke alle IMU-er måler magnetfelt, men IMU-enheten har valgt gjør det. Hver av sensorene måler sin enhet langs tre akser, x, y, z , som tilsammen utgjør ni akser, som også blir kalt ni frihetsgrader. Mellom enhver sensor i IMU-enheten vil tilsvarende akser være parallelle. Parallelle akser vil gjøre arbeidet med å behandle dataene fra sensorene langt lettere, siden x -, y - og z -aksene kan beregnes som samme aksen for alle sensorene. Måten gruppen har montert IMU-chip på karosseriet gjør at y -aksen går i samme retning som hjul-akslen, z -aksen peker i kjøreretning, og x -aksen går parallelt med høyden til JetBalancer, altså ortogonalt på de andre aksene.

Gyroskop

Gyroskop måler vinkelhastigheten rundt tre ortogonale akser. SI-enheten for vinkelhastighet er $[s^{-1}]$ (radianer per sekund), men $[^{\circ}/s]$ (grader per sekund) er også mye brukt. Siden gyroskopet gir vinkelhastighet vil den alltid returnere hastigheten til et gitt tidspunkt. Dersom gyroskopet er i rotasjon vil verdien være høy, og dersom det står rolig vil gyroskopet returnere $0^{\circ}/s$. Ved å ta flere målinger og integrere disse verdiene over et tidsrom finner man orienteringen til gyroskopet på et tidspunkt i forhold til da målingene startet. Gyroskopets signaler har svært lite støy, men en naturlig drift også kalt partiskhet (bias). Den naturlige driften og partiskhet kan variere fra gyroskop til gyroskop, men selv et høyteknologisk gyroskop vil ha en drift som skyldes jordens rotasjon. På grunn av drift er det viktig å kalibrere målingene før en begynner å integrere målingene.

Akselerometer

Akselerometre måler akselerasjonen den selv gjennomgår langs tre ortogonale akser. SI-enheten for et akselerasjon er $[m/s^2]$. Siden jordens tyngdeakselerasjon er konstant $\sim 9,81 [m/s^2]$ vil et akselerometer alltid måle tyngdeakselerasjonen, med mindre akselerometeret er i fritt fall. Dersom en måler tyngdekraften langs hver akse til akselerometeret kan en beregne orienteringen til akselerometeret i forhold til tyngdekraften, som man vil anta å være perpendikulær med bakken. Akselerometeret er ikke godt egnet til å måle vinklingen på egenhånd fordi det vil reagere på annen akselerasjon enn tyngdekraften, og på grunn av høyfrekvens støy. Allikevel er akselerometerdataen nyttig til vinkel måling, fordi den kan brukes til å komplementere gyroskopet, som påvirkes av drift.

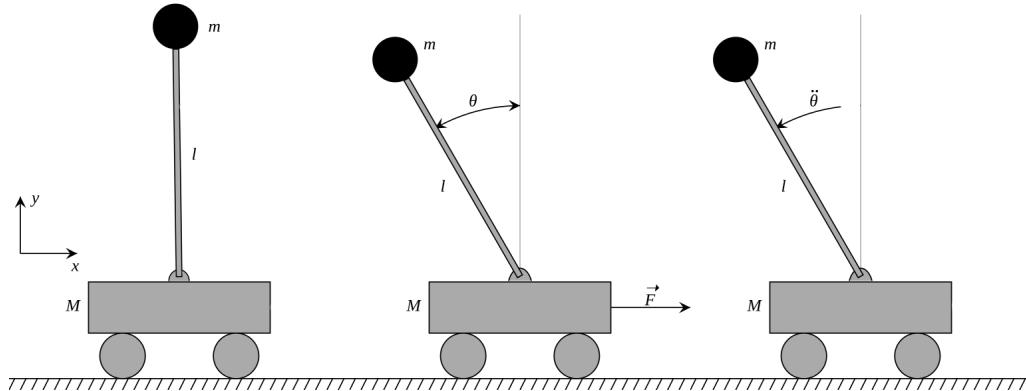
Magnetometer

Et magnetometer måler magnetisk feltstyrke, med SI-enhet $[T = \frac{kg}{s^2 A}]$, relativt til jordens magnetiske nord. For å måle JetBalancers yaw-vinkling, kan en bruke feltstyrken langs y - og x -aksene ettersom magnetometeret vil måle jordas magnetfelt, som man kan anta å være konstant i forhold til bakken.

2.6.2 Invertert Pendel

En invertert pendel er en pendel hvor massen er plassert over et vippepunkt.

En invertert pendel står i ustabil mekanisk likevekt når pendelen står vertikalt over vippepunktet til pendelen, se tegningen til venstre på Figur 2.10.



Figur 2.10: Invertet Pendel. Redigert fra [5]

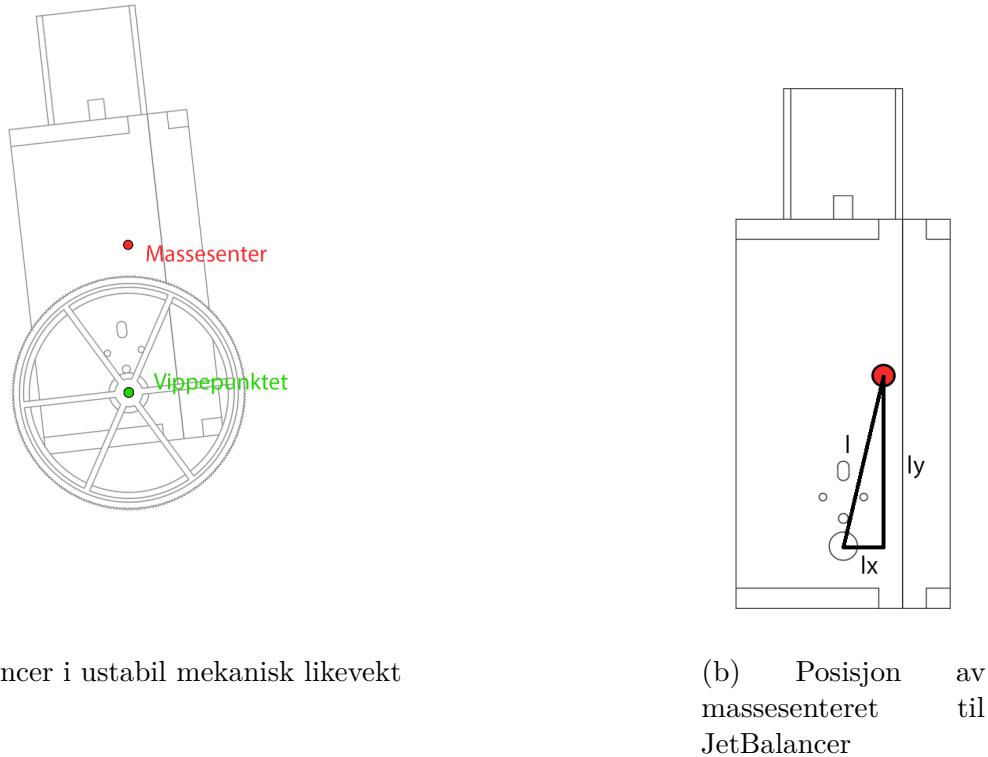
Figur 2.10 illustrerer en invertert pendel plassert på en tralle, staven ansees å være vektløs, mens m er massesenteret til pendelen og M er massen til trallen. Lengden l er distansen mellom vippepunktet og massesenteret til pendelen. θ er vinkelen pendelen har fra likevekt.

Når en kraft \mathbf{F} påføres trallen, vil pendelen endre θ i motsatt retning av vektoren \mathbf{F} . Om trallen så stopper, mens $\theta \neq 0$, vil pendelen fortsette å falle grunnet tyngdekraften.

Det er av de overnevnte grunnene at en invertert pendel krever feedback kontroller for å være stabil.

2.6.3 Invertert Pendel. Del-2

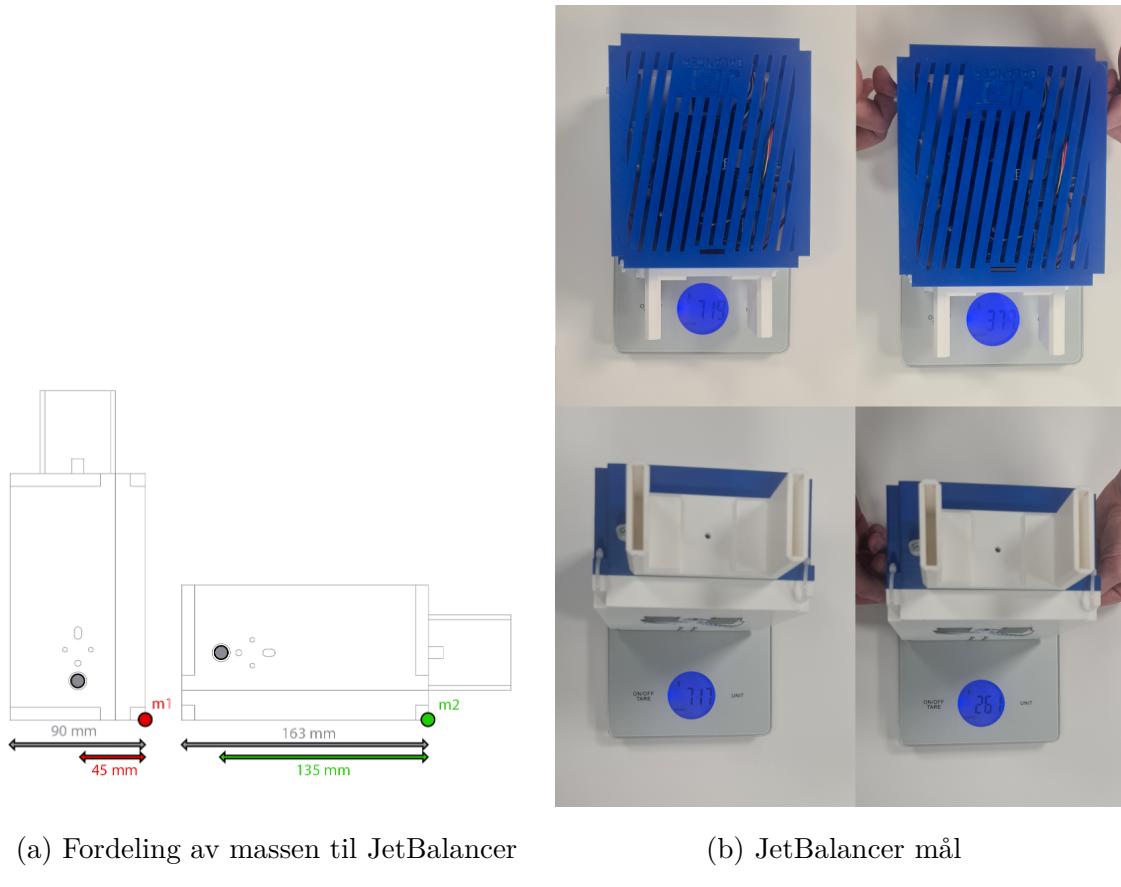
Siden JetBalancer skal balansere sin egen vekt over vippepunktet kan den anses som en invertert pendel. Figur 2.11(a) viser JetBalancer sin posisjon/orientering når den står i ustabil mekanisk likevekt.



Figur 2.11: JetBalancer i ustabil likevekt

For å kunne gjøre beregninger på JetBalancer som en invertert pendel er det viktig å vite avstanden fra massesenteret til vippepunktet, merket med l i Figur 2.11(b). For å beregne lengden lx og ly i Figur 2.11(b), kan man benytte fysikken til en balanserende bjelke, lengden l finner man da enkelt ved hjelp av Pythagoras' læresetning.

For å finne lx og ly måler vi først totalvekten til JetBalancer uten hjul, for så å måle vekten av JetBalancer når vippepunktet er fast om en roterende akse ved å løfte etter vippepunktet.



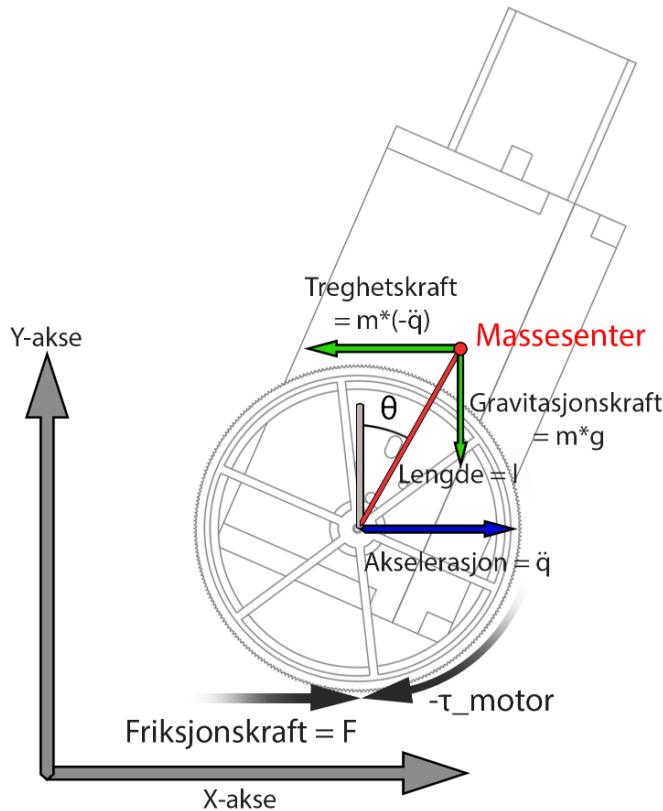
Figur 2.12: JetBalancer i ustabil likevekt

Figur 2.12 viser hvordan fordelingen av massen til JetBalancer ble målt. Bemerk at dette er ikke en presis måling, men vil gi et tilstrekkelig grunnlag til å beregne plasseringen av massesenteret. Fra målingenene i Figur 2.12(b) kan en fastslå $M = 718g$, $m_1 = 261g$, $m_2 = 377g$, hvor M er totalmassen og m_1 og m_2 er massen i punktene fra Figur 2.12(a).

Distansene lx , ly og l i Figur 2.11(b):

$$\begin{aligned} lx &= \frac{377 \cdot 135}{718} \approx 71mm \\ ly &= \frac{261 \cdot 45}{718} \approx 16mm \\ l &= \sqrt{71^2 + 16^2} \approx 73mm \end{aligned} \tag{2.1}$$

Akselerasjon



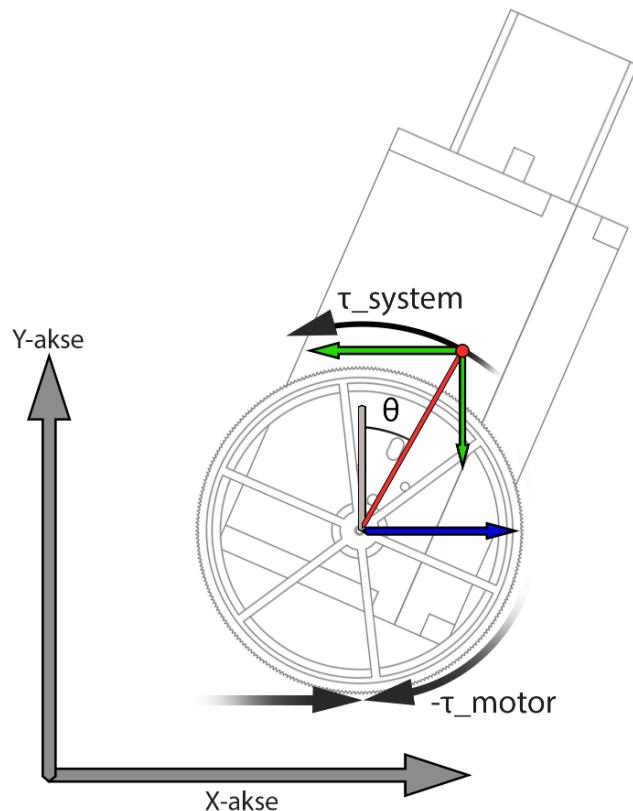
Figur 2.13: Akselerasjon av JetBalancer i forhold til et referancesystem

I Figur 2.13 illustreres kreftene som påføres systemet når motorene til JetBalancer begynner å rotere. Dreiemomentet fra motoren er merket med τ . Så lenge hjulet ikke glipper mot underlaget, vil frikjonskraften F være lik $(\tau/hjulradie)$, dette fører til at hjulnavet begynner å akselerere. \ddot{q} er akselerasjonen til hjulnavet i forhold til et referancesystem.

Som et resultat av akselerasjonen, oppstår det en trehetskraft fra massesenteret vil alltid være motsatt rettet til akselerasjonen \ddot{q} .

$$\begin{aligned} \text{Trehetskraft} &= m_{(\text{massen uten hjul})} \cdot (-\ddot{q}) \\ \text{Gravitasjonskraft} &= m_{(\text{massen uten hjul})} \cdot 9,81[m/s^2] \end{aligned} \quad (2.2)$$

Siden massesenteret roterer rundt et vippepunkt vil summen av trehetskraften og gravitasjonskraften danne et moment.



Figur 2.14: Dreiemoment til JetBalancer

Massesenterets vertikale og horisontale avstand til vippespunktet kan beskrives som en funksjon av θ .

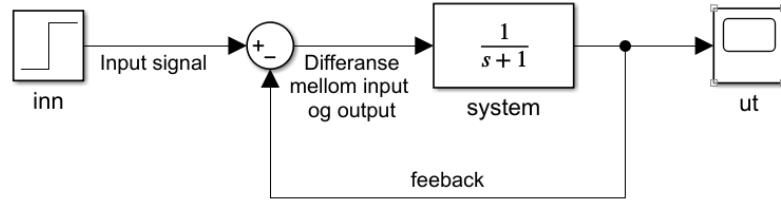
$$\begin{aligned} \text{Vertikal} &= \cos(\theta) \cdot l \\ \text{Horisontal} &= \sin(\theta) \cdot l \end{aligned} \quad (2.3)$$

JetBalancer sin akselerasjon bestemmer da det totale dreiemomentet til systemet slik.

$$\begin{aligned} \tau_{\text{system}} &= (-\ddot{q}) \cdot m \cdot l \cdot \sin(\theta) + g \cdot m \cdot l \cdot \cos(\theta) \\ \tau_{\text{system}} &= m \cdot l \cdot (-\ddot{q} \cdot \sin(\theta) + g \cdot \cos(\theta)) \end{aligned} \quad (2.4)$$

Ut ifra teorien i dette kapittelet kan vi sette opp et feedbacksystem som kontinuerlig overvåker vinkelen θ , og sette spenning over motorene som får JetBalancer til å akselerere i den retningen som minimerer absoluttverdien til θ .

2.6.4 Feedback



Figur 2.15: Eksempel på feedback system

Ettersom teksten vil diskutere feedback systemer vil gruppen komme med en kort og overfladisk definisjon av feedback i denne seksjonen.

Feedback refererer til en situasjon der to eller flere dynamiske systemer er sterkt koblet sammen slik at systemene påvirker hverandre. Oppførselen til et feedback system er ofte kontraintuitiv da det første dynamiske systemet vil påvirke det andre, men det andre vil igjen påvirke det første systemet. Påvirkningen vil danne en løkke.

Et system er i en *lukket løkke* dersom de indre systemene er koblet sammen i en syklus. Ettersom et feedback system er koblet i en syklus er det vanligvis hvilket indre system som blir definert som det første, og hvilket som blir definert som det siste.

2.6.5 PID-Kontroller

En *Proportional Integrated Derivative* (PID) kontroller er en kontroll-loop mekanisme som tar i bruk feedback for å kunne ta i bruk en nøyaktig og responsiv korreksjon til en kontrollfunksjon. For å finne ut hvilke korrekSJoner som må gjøres vil en PID kontroller kontinuerlig regne ut en feilverdi som er differansen mellom ønsket settpunkt og målt verdi. I Figur 2.16 kan en se blokk-diagrammet av feedback som beskriver en PID-kontroller.

Gitt en målt verdi y , referanseverdi ψ og avviket

$$e = y - \psi \quad (2.5)$$

En PID-kontroller er en funksjon av avviket der output-signalen u er en lineærkombinasjon av avviket, den deriverte av avviket, og integralet av avviket. [6] Altså:

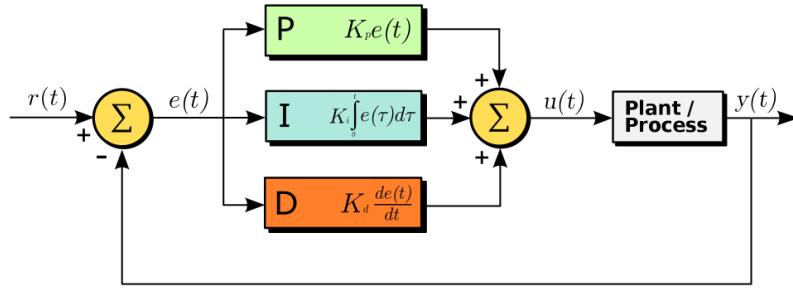
$$u(t) = k_p e(t) + k_i \int_0^t e(t) dt + k_d \frac{de(t)}{dt} \quad (2.6)$$

De tre koeffisientene k_p, k_i, k_d kan justeres for å endre hvor mye utslag hvert ledd tilsvarer.

k_p bestemmer hvor sterkt utslag den proporsjonale delen av PID-kontrolleren gir. Dette leddet er proporsjonalt til avviket.

k_i bestemmer hvor sterkt utslag integrasjonsdelen i kontrolleren gir, som er proporsjonal til *akkumulert* avvik.

k_d bestemmer hvor sterkt utslag derivasjonsdelen gir, som er proporsjonal til hastigheten av endring i avviket.



Figur 2.16: Blokkskjema av en PID kontroller, illustrerer Ligning 2.6 [7]

I prosjektets implementasjon vil den målte verdien være JetBalancer pitch-vinkel, referanseverdien vil være ønsket pitch-vinkel, som er en funksjon av kjørehastigheten som etterspørs, og outputen vil være den lineære hastigheten til JetBalancer. Se gjerne på Figur 2.9 for en mer visuell forklaring på hvilke parametere som er hva.

2.6.6 Emulering

PID-kontrolleren er beskrevet ovenfor i henhold til kontinuerlig tid, men JetBalancers datamaskin jobber i diskret tid. Ettersom datamaskinen kun kan lese diskrete verdier må de kontinuerlige verdiene fra PID-kontrolleren approksimeres slik at de blir lesbare for JetBalancer sin datamaskin.

Gruppen velger å benytte «forward rectangle»-approksimasjoner som er mindre nøyaktige enn en del alternativer, men også en del raskere. Hastigheten gjør at det er verdt å tape litt på nøyaktigheten.

Fremgangsmåten for å finne uttrykk for PID-kontrolleren i diskret tid er at det først approksimeres i Z-domenet, og deretter finner den inverse Z-transformen av det resulterende uttrykket.

Approksimering av integralet

Den integrerte har følgende transferfunksjon.

$$\frac{I(s)}{E(s)} = \frac{1}{s} \quad (2.7)$$

Der I og E er laplacetransformene av funksjonene $i(t) = \int_0^t e(u)du$ og $e(t)$, som har kontinuerlig tid som domene.

Forward rectangle approksimasjonen i Z-domenet er