

$$s \leftarrow \frac{z - 1}{T} \quad (2.8)$$

$$\frac{I(z)}{E(z)} = \frac{T}{z - 1}$$

$$(z - 1)I(z) = TE(z)$$

$$zI(z) - I(z) = TE(z)$$

$$I(z) - z^{-1}I(z) = Tz^{-1}E(z)$$

$$I(z) = Tz^{-1}E(z) + z^{-1}I(z)$$

Approksimasjonen av integralet kan da uttrykkes ved følgende differensligning.

$$\begin{aligned} \mathcal{Z}^{-1}\{I(z)\} &= T\mathcal{Z}^{-1}\{z^{-1}E(z)\} + \mathcal{Z}^{-1}\{z^{-1}I(z)\} \\ i_k &= Te_{k-1} + i_{k-1} \end{aligned} \quad (2.10)$$

Approksimering av den deriverte

Den deriverte med lavpassfilter har følgende transferfunksjon, der f_G er grensefrekvensen til filteret. Lavpassfilteret er inkludert fordi støy i avviket kan gi høye og uønskede utslag i den deriverte.

$$\frac{D(s)}{E(s)} = s \frac{f_G}{f_G + s} \quad (2.11)$$

Forward rectangle approksimasjonen i Z -domenet er, med klokkeperioden T ,

$$s \leftarrow \frac{z - 1}{T} \quad (2.12)$$

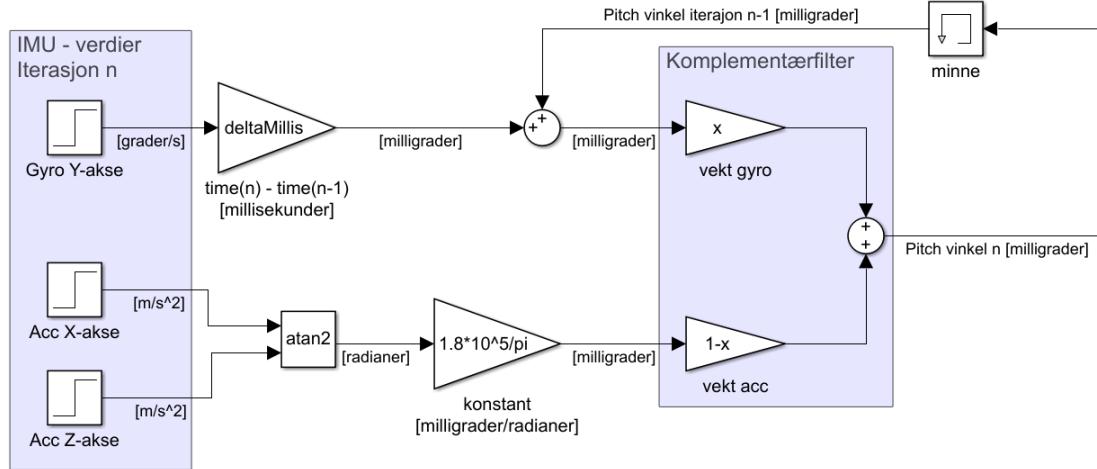
$$\begin{aligned} \frac{D(z)}{E(z)} &= \frac{z - 1}{T} \frac{f_G}{f_G + \frac{z-1}{T}} \\ \left(f_G + \frac{z-1}{T}\right)D(z) &= \frac{(z-1)f_G}{T}E(z) \\ f_GD(z) + \frac{z-1}{T}D(z) &= \frac{(z-1)f_G}{T}E(z) \\ \frac{1}{T}zD(z) - \frac{1}{T}D(z) &= \frac{f_G}{T}zE(z) - \frac{f_G}{T}E(z) - f_GD(z) \\ \frac{1}{T}zD(z) &= \frac{f_G}{T}zE(z) - \frac{f_G}{T}E(z) - f_GD(z) + \frac{1}{T}D(z) \\ D(z) &= f_G(E(z) - z^{-1}E(z)) + (1 - T \cdot f_G)z^{-1}D(z) \end{aligned} \quad (2.13)$$

En rekursiv definisjon av den deriverte med lavpassfilter i diskret tid blir da

$$\begin{aligned} \mathcal{Z}^{-1}\{D(z)\} &= f_G \cdot (\mathcal{Z}^{-1}\{E(z)\} - \mathcal{Z}^{-1}\{z^{-1}E(z)\}) + (1 - T \cdot f_G)\mathcal{Z}^{-1}\{z^{-1}D(z)\} \\ d_k &= f_G \cdot (e_k - e_{k-1}) + (1 - T \cdot f_G)d_{k-1} \end{aligned} \quad (2.14)$$

Det forventes ikke mye høyfrekvens støy, så lavpassfilteret er muligens unødvendig, men det er inkludert for sikkerhets skyld i tilfelle det blir nyttig. Skulle filteret være unødvendig, kan grensefrekvensen økes for å redusere innvirkningen av det.

2.6.7 Behandling av sensordata



Figur 2.17: Behandling av sensordata

Figur 2.17 er en modell som illustrerer en programløkke som kjøres i nåværende iterasjon (n) med forrige iterasjon ($n - 1$). Modellen viser hvordan sensordataene fra gyroskopet og akselerometeret kan benyttes for å pålitelig beregne orienteringen av IMU-enheten. Bemerk at Figur 2.17 benytter y -aksen til gyroen og x - og z -aksene fra akselerometeret—Atan2 trenger målingene langs aksene som er *ortogonale* med rotasjonsaksen.

Modellen i Figur 2.17 vil kun kalkulere orienteringen i forhold til bakken rundt y -aksen. Dette forholdet er tilstrekkelig for prosjektet da JetBalancer skal balansere på to hjul, og det antas at JetBalancer er stabil rundt z -aksen. Orienteringen til bakken målt rundt y -aksen er videre i rapporten navngitt θ .

Vinkelhastighet fra gyro

Gyroskopet mäter vinkelhastigheten rundt y -aksen. Når en måling blir tatt fra gyroen, tilsvarer verdien på målingen hastigheten gyroen har rundt sin egen y -akse, med benevningen [$^\circ$ /sekund]. Siden verdien som skal brukes er endelig vinkel og ikke nødvendigvis vinkelhastigheten, kan vinkelhastigheten integreres med hensyn på tid for å få vinkelverdien. Vinkelhastigheten får navnet $\dot{\theta}_{gyr_y}$.

Nå som den deriverte $\dot{\theta}$ er funnet, kan en integrere med hensyn på tid for å finne θ .

$$\int_0^t \dot{\theta} dt = \theta \quad (2.15)$$

Siden dette er et digitalt system må en ta hensyn til at målingene ikke blir tatt kontinuerlig og av den grunn må hastigheten mellom hvert målepunkt approksimeres. Approksimasjonen kan beregnes på flere måter, men letteste er å ta verdien til $\dot{\theta}$ og multiplisere med tiden som har gått siden sist måling.

$$\Delta\theta_{gyr_y} [\text{milli}^\circ] = \dot{\theta} [^\circ/\text{sekund}] \cdot \Delta t [\text{millisekund}] \quad (2.16)$$

Hvor:

$$\Delta t[\text{millisekund}] = \text{tidspunkt}_n - \text{tidspunkt}_{n-1} \quad (2.17)$$

Fra Ligning 2.16 finner gruppen en approksimasjon på hvor mange grader IMU-enheten har rotert fra forrige måling. Det er viktig å bemerke seg at tiden Δt spiller en viktig rolle i hvor nøyaktig IMUen kan være; jo mindre Δt er desto mer nøyaktig blir approksimasjonen.

Når endringen siden forrige måling er funnet, kan denne summeres med alle tidligere målinger for å få en endelig vinkel, men dette er ikke ønskelig da det har kritiske feil. En feil er at gyroen er utsatt for drift, dette betyr at over tid vil målingene driftet fra start, sånn at nullpunktet vil ikke være det samme som det var da målingene startet. Den andre feilen er at gyroen alene ikke vet hva som er opp eller ned, men registrerer kun retningen den roterer rundt sine akser. På grunn av kritiske feil, som diskutert, vil gruppen benytte verdien som blir funnet etter komplementærfilteret i Figur 2.17, denne verdien får navnet $\theta_{(n-1)}$.

$\theta_{(n-1)}$ er kalkulert endelig vinkel fra forrige måling, av både gyroskopet og akselerometeret, og vil både kansellere driftingen fra gyroskopet og akselerometeret vil gi retningen ned.

$$\theta_{gyr_y}[n] = T\dot{\theta}_{gyr_y} + \theta_{(n-1)} \quad (2.18)$$

$\theta_{gyr_y}[n]$ er da verdien som kan sendes fra gyroskopet til komplementærfilteret, og er den øvre inngangen til komplementærfilteret i Figur 2.17.

Vinkelmåling fra akselerometer

For å bruke akselerometermålingene til å måle pitch-vinkelen θ til JetBalancer, kan Atan2-funksjonen brukes.

$$\theta_{acc} = \text{Atan2}(acc_x, acc_z) \quad (2.19)$$

Her er acc_x og acc_z sensormålinger fra akselerometeret i IMU-ens x - og z -akser.

Atan2

En funksjon som utifra to argumenter, y og x , finner vinkelen til et hjørne i en rettsidet trekant, der y er lengden på *motstående* katet og x er lengden på *hosliggende* katet. [8]

Atan2 ligner på \tan^{-1} , men har en bredere definisjonsmengde, $(-\pi, \pi]$, i motsetning til \tan^{-1} , som er begrenset til $(-\frac{\pi}{2}, \frac{\pi}{2}]$. Atan2 gir også et resultat hvis $x = 0$, som ikke gir mening med \tan^{-1} .

Definisjon av Atan2:

$$\text{Atan2}(y, x) = \begin{cases} \tan^{-1} \frac{y}{x} & x > 0 \\ \tan^{-1} \frac{y}{x} + \pi & x < 0 \wedge y \geq 0 \\ \tan^{-1} \frac{y}{x} - \pi & x < 0 \wedge y > 0 \\ \frac{\pi}{2} & x = 0 \wedge y > 0 \\ -\frac{\pi}{2} & x = 0 \wedge y < 0 \end{cases} \quad (2.20)$$

For x -er og y -er hvor både $\text{Atan2}(y, x)$ og $\tan^{-1}(y/x)$ er definert er

$$\text{Atan2}(y, x) = \tan^{-1}(y/x) \quad (2.21)$$

2.6.8 Sensorfusjon og komplementærfilter

Ved å benytte *sensorfusjon* på dataene fra akselerometeret med den integrerte verdien fra gyroskopet, kan man oppnå en mer nøyaktig måling av orienteringen til IMUen som både har mindre støy enn akselerometeret og ingen drift fra gyroskopet.

Akselerometere har en del høyfrekvens støy, og gyroskoper har lavfrekvens støy i form av drift. For å omgå støy-problemetene, trekkes det beste ut av disse sensorene ved hjelp av et *komplementærfilter*.

Et komplementærfilter er summen av ett signal sendt gjennom et lavpassfilter, og et annet signal sendt gjennom et høypassfilter. Filtrene har samme grensefrekvens.

Komplementærfilteret har følgende laplacetransform, der Θ_x er laplacetransformen av vinkelen målt av sensor x , og f_G er grensefrekvensen

$$\begin{aligned}\Theta(s) &= \frac{f_G}{s + f_G} \Theta_{acc}(s) + \left(1 - \frac{f_G}{s + f_G}\right) \Theta_{gyro}(s) \\ &= \frac{f_G}{s + f_G} \Theta_{acc}(s) + \left(1 - \frac{f_G}{s + f_G}\right) \frac{1}{s} \dot{\Theta}_{gyro}(s) \\ &= \frac{f_G}{s + f_G} \Theta_{acc}(s) + \frac{s}{s + f_G} \cdot \frac{1}{s} \cdot \dot{\Theta}_{gyro}(s) \\ &= \frac{f_G}{s + f_G} \Theta_{acc}(s) + \frac{1}{s + f_G} \dot{\Theta}_{gyro}(s)\end{aligned}\tag{2.22}$$

Backward rectangle approksimasjon i det diskrete Z-domenet gir følgende Z-transform, der T er klokkeperioden:

$$\begin{aligned}s &\rightarrow \frac{z - 1}{Tz} = \frac{1 - z^{-1}}{T} \\ \Theta(z) &= \frac{f_G}{\frac{1-z^{-1}}{T} + f_G} \Theta_{acc}(z) + \frac{1}{\frac{1-z^{-1}}{T} + f_G} \dot{\Theta}_{gyro}(z)\end{aligned}\tag{2.23}$$

Med Ligning 2.23 kan en utlede følgende rekursive ligning for komplementærfilteret:

$$\begin{aligned}\Theta(z) &= \frac{f_G}{\frac{1-z^{-1}}{T} + f_G} \Theta_{acc}(z) + \frac{1}{\frac{1-z^{-1}}{T} + f_G} \dot{\Theta}_{gyro}(z) \\ \left(\frac{1-z^{-1}}{T} + f_G\right) \Theta(z) &= f_G \Theta_{acc}(z) + \dot{\Theta}_{gyro}(z) \\ \left(\frac{1}{T} + f_G\right) \Theta(z) - \frac{1}{T} z^{-1} \Theta(z) &= f_G \Theta_{acc}(z) + \dot{\Theta}_{gyro}(z) \\ \Theta(z) &= \frac{1}{\frac{1}{T} + f_G} f_G \Theta_{acc}(z) + \frac{1}{\frac{1}{T} + f_G} \left(\dot{\Theta}_{gyro}(z) + \frac{1}{T} z^{-1} \Theta(z) \right) \\ &= \left(1 - \frac{1}{1 + f_G T}\right) \Theta_{acc}(z) + \frac{1}{1 + f_G T} \left(T \dot{\Theta}_{gyro}(z) + z^{-1} \Theta(z) \right) \\ &= (1 - \gamma) \Theta_{acc}(z) + \gamma \left(T \dot{\Theta}_{gyro}(z) + z^{-1} \Theta(z) \right) \\ \mathcal{Z}^{-1} \{ \Theta(z) \} &= (1 - \gamma) \mathcal{Z}^{-1} \{ \Theta_{acc}(z) \} \\ &\quad + \gamma \left(T \mathcal{Z}^{-1} \{ \dot{\Theta}_{gyro}(z) \} + \mathcal{Z}^{-1} \{ z^{-1} \Theta(z) \} \right) \\ \theta[k] &= (1 - \gamma) \cdot \theta_{acc}[k] + \gamma \cdot \left(T \dot{\theta}_{gyro}[k] + \theta[k - 1] \right)\end{aligned}\tag{2.24}$$

$$\gamma = \frac{1}{1 + f_G T} \quad (2.25)$$

Passende grensefrekvens vil bli funnet ved eksperimentering.

2.6.9 Basisskifte

Siden IMU-en vil roteres sammen med roboten, vil også koordinatsystemet dets roteres. For å få et justert koordinatsystem som er vannrett med bakken, kan man gjøre et basisskifte, som i dette tilfellet er en rotasjon på vinkel θ om y-aksen.

Transformasjonsmatrisen for skiftet er

$$R_{y,-\theta} = \begin{bmatrix} \cos(-\theta) & 0 & \sin(-\theta) \\ 0 & 1 & 0 \\ -\sin(-\theta) & 0 & \cos(-\theta) \end{bmatrix} \quad (2.26)$$

Bildet v av en vilkårlig vektor v_0 er

$$\begin{aligned} v &= R_{y,-\theta} v_0 \\ &= \begin{bmatrix} \cos(-\theta) & 0 & \sin(-\theta) \\ 0 & 1 & 0 \\ -\sin(-\theta) & 0 & \cos(-\theta) \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \\ &= \begin{bmatrix} (\cos(-\theta))x_0 + (\sin(-\theta))z_0 \\ y_0 \\ -(\sin(-\theta))x_0 + (\cos(-\theta))z_0 \end{bmatrix} \\ &= \begin{bmatrix} (\cos \theta)x_0 - (\sin \theta)z_0 \\ y_0 \\ (\sin \theta)x_0 + (\cos \theta)z_0 \end{bmatrix} \end{aligned} \quad (2.27)$$

I denne justerte basisen kan man bruke gyroskopet til å finne JetBalancers vinkelhastighet i yaw-retning, $\dot{\phi}$, ved å bruke Atan2-funksjonen.

$$\begin{aligned} \dot{\phi} &= \text{Atan2}(gyro_y, gyro_z) \\ &= \text{Atan2}(gyro_{y_0}, (\sin \theta)gyro_{x_0} + (\cos \theta)gyro_{z_0}) \end{aligned} \quad (2.28)$$

$\dot{\phi}$ planlegges å brukes i den delen av styringsalgoritmen som kontrollerer svinging til høyre og venstre, men det har ikke blitt implementert enda.

2.7 Relevant Teknologi

2.7.1 Autodesk Fusion 360

Fusion 360 er et verktøy som samler en hel produktutviklingsprosess i én skybasert programvare [9]. Verktøyet for 3D-modellering brukes i sammenheng med industrielt design for en realistisk modell av arbeidsområder og verktøy. 3D-filene som produseres kan for eksempel eksporteres slik at de kan brukes i programsimulering. Når noe skal 3D-printes kan modellene lages i Fusion 360.

2.7.2 JetBot programvare

JetBot er styrt av NVIDIA Jetson Nano som er en enkeltbords datamaskin (SBC). SBC'en er så videre styrt av operativsystemet Ubuntu, som er en Linux-distribusjon. Installert på operativsystemet er et docker-miljø, med egen container for webserver som gir et web-grensesnitt til JetBot og kjøres når systemet slås på [10].

NVIDIA har et Git-repository som inneholder eksempelkode for JetBot, med eksempler for bildegenkjenning og autonom kjøring. Kodeeksemplene vil antageligvis være et godt grunnlag for vår eventuelle løsning [11].

2.7.3 GitHub

GitHub er en plattform som tilbyr en tjeneste for bedrifter og privatpersoner der man kan lagre programkode og tilhørende dokumentasjon for sine prosjekter.

2.7.4 Jupyter Notebook

Jupyter Notebook er et åpent-kilde IDE, og nettbasert interaktivt beregningsmiljø, som støtter flere programmeringsspråk. I dette prosjektet vil gruppen ta nytte av Jupyter med Python. *kernel* er en *computational engine* som utfører koden som blir skrevet i en notebook.

2.7.5 Docker

Docker er en software plattform som gjør det mulig å bygge, teste, og distribuere applikasjoner kjapt. Docker kan pakke software inn i standardiserte enheter kalt containerere som igjen inneholder alt som programvaren trenger for å kunne kjøre koden inkludert biblioteker, systemverktøy, kode, og runtime. Ved bruk av Docker kan en enkelt skalere og kjøre ønskede applikasjoner.

Kapittel 3

Design

Designet til JetBalancer består av deler tatt med fra JetBot, deler som gruppen har designet og printet selv, og deler som gruppen har lagt til. Seksjonene nedenfor forklarer detaljer om delene tilhørende de nevnte kategoriene.

Design vil innebære utforming og materiale på karosseriet, velteburet, hjulene, og kontrollbordet. Delene som er designet og produsert selv vil være utformet i CAD programvaren Fusion 360 fra Autodesk og enkelte deler vil være utformet i vektorgrafikk-programmet Illustrator fra Adobe.

3.1 Deler fra JetBot

JetBot kom med deler som gjorde den kapabel til å kjøre, og drive bildegenkjenning og maskinlæring. Noen av delene har gruppen valgt å ta med og bruke videre i designet til JetBalancer, mens noen av delene er ikke tatt med videre.

3.1.1 JetBot karosseri og kontrollbord



Figur 3.1: Oversikt av delene JetBot kom med. [12]

Delene gruppen har valgt å ikke ha med videre på JetBalancer er:

- 3** Metalboks
- 4** Kameraholder
- 5** Akrylbiter
- 10** Hjul (2 stk)
- 11** Kuleledd
- 14** Trådløs gamepad

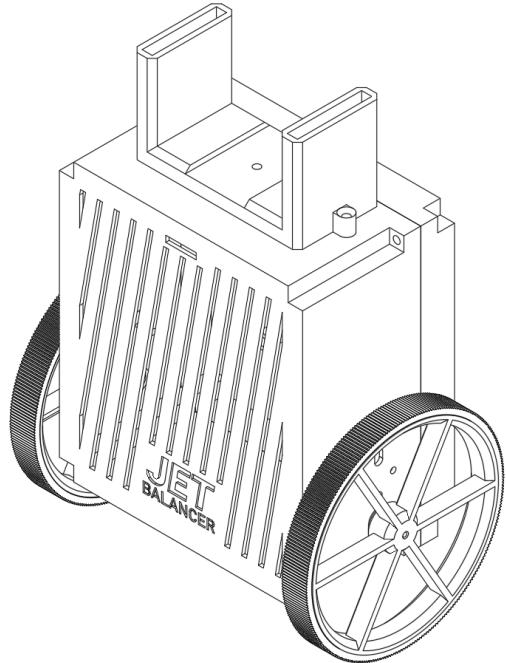
Delene gruppen har valgt å bruke i JetBalancer er:

- 1** Jetson Nano Dev Kit
- 2** microSD-kort
- 6** JetBot utvidelseskort
- 7** IMX219-160 kamera
- 8** Wireless
- 9** Girmotor (2 stk)
- 12** Strømadapter (EU)
- 13** 12.6V batterilader
- 16** 6Pin 9cm kabel
- 17** Skruer
- 19** 4090 kjølevifte

3.2 Deler gruppen har designet og produsert

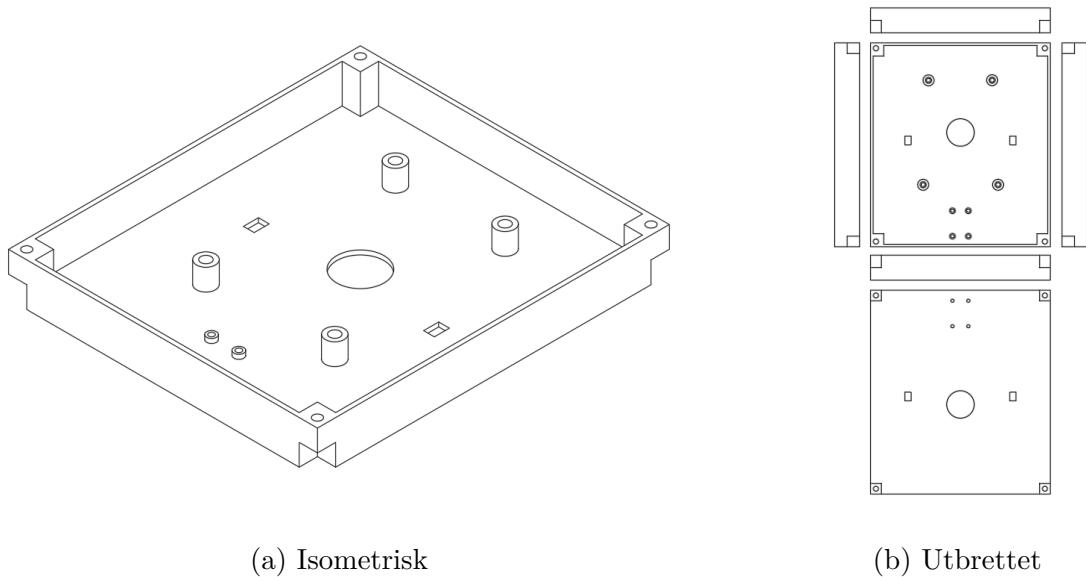
I denne seksjonen beskrives delene som har blitt designet og fabrikkert for JetBalancer.

3.2.1 Karosseri



Figur 3.2: Fullstendig sammensatt karosseri og hjul

Karosseriet vil bestå av fire deler: ETUI_1, ETUI_2, FORLENGER_FESTE og KAMERA_FESTE. ETUI_1 og ETUI_2 omslutter kontrollbord, IMU og motorer. Kontrollbord og IMU er fastmontert i ETUI_1. Motorer og FORLENGER_FESTE er fastmontert i ETUI_2. KAMERA_FESTE er fastmontert i FORLENGER_FESTE. Alle delene av karosseriet er 3D-printet i PLA filament.

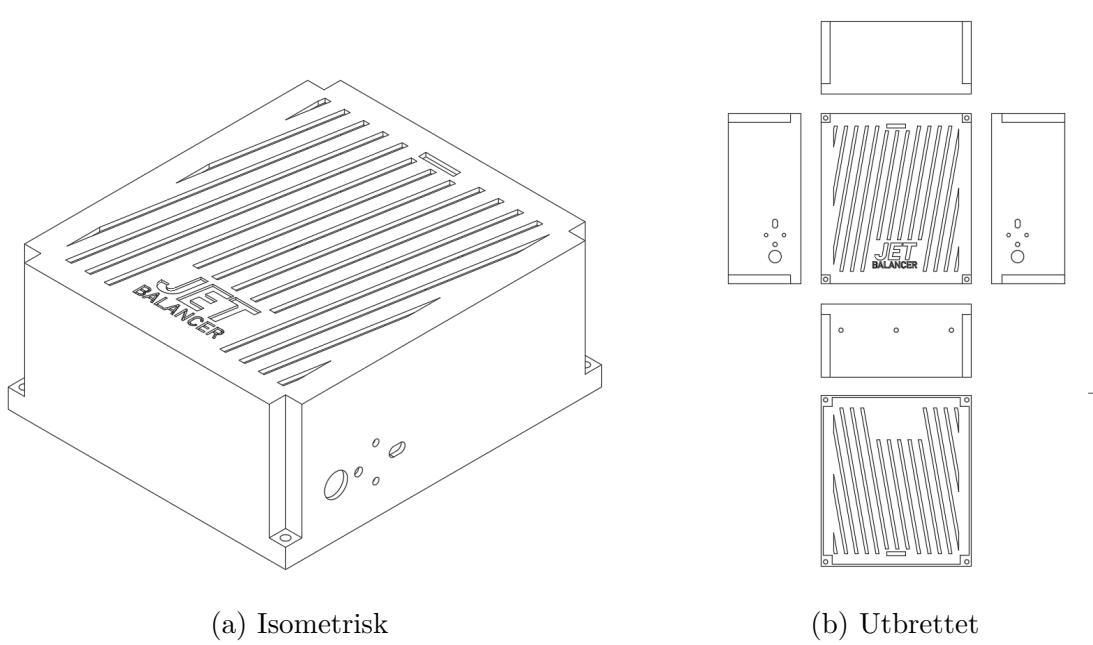


(a) Isometrisk

(b) Utbrettet

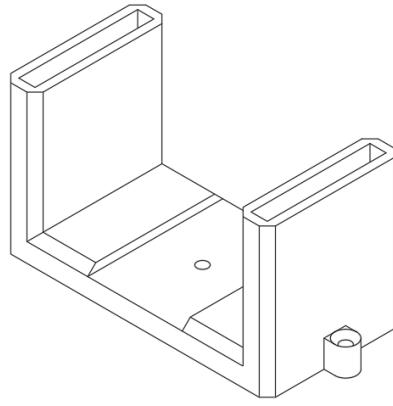
Figur 3.3: ETUI_1

ETUI_1 har på innsiden fire innebygde avstandsstykker hvor det er plass til messing gjenger i M2 størrelse. Messing gjengene er smeltet inn i plastikken med en egnet loddespiss. I messing gjengene monteres Waveshare utvidelseskortet med M2 skruer. Mellom avstandsstykkekene er det hull som kablene til motorene kan passere gjennom. Nedenfor er det avstandsstykker hvor IMUen kan monteres med M2 skruer. På utsiden av ETUI_1 er det fire hull som brukes til å feste ETUI_2

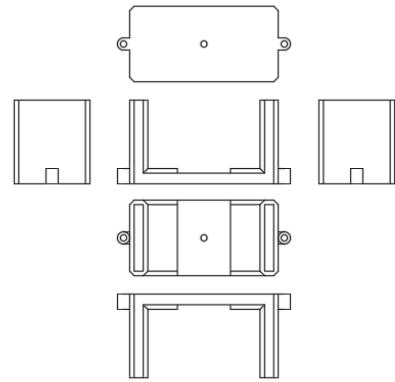


Figur 3.4: ETUI_2

ETUI_2 er festet i ETUI_1 og har et feste på hver side til motorene. Festene består av to skruehull, to hakk som sikrer motoren fra å bevege seg samt et hull til akslingen. På oversiden er det riller som lar luft strømme gjennom og en logo for JetBalancer. På toppen er det et firkantet hull som lar flatkabelen til *IMX219-160*-kameraet passere gjennom.



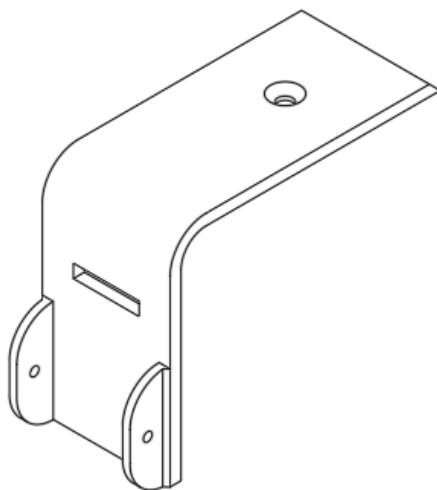
(a) Isometrisk



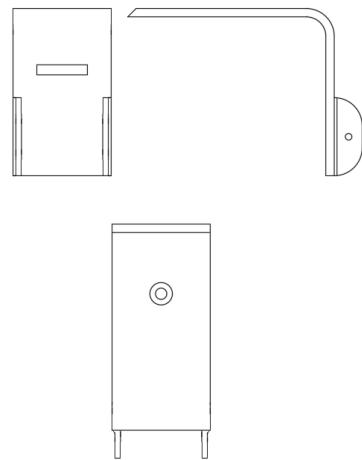
(b) Utbrettet

Figur 3.5: FORLENGER_FESTE

FORLENGER_FESTE monteres på toppen av ETUI_2 med to M4 skruer. En tredje M4 skrue fester KAMERA_FESTE til midten av FORLENGER_FESTE og ETUI_2. FORLENGER_FESTE har som hensikt å tillate hurtig utbytting eller fjerning av velteburet.



(a) Isometrisk

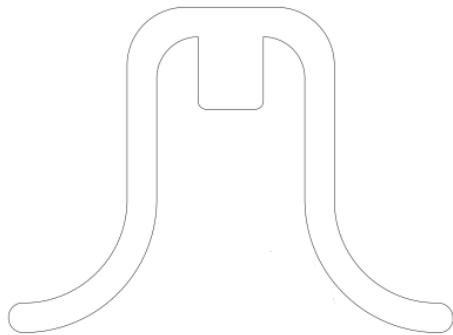


(b) Utbrettet

Figur 3.6: KAMERA_FESTE

KAMERA_FESTE skal gjøre det mulig å montere *IMX219-160*-kameraet på JetBalancer. KAMERA_FESTE monteres i FORLENGER_FESTE. I midten er det et rektangulært hull som lar flatkabelen passere gjennom.

3.2.2 Veltebur



Figur 3.7: Veltebur vektortegning

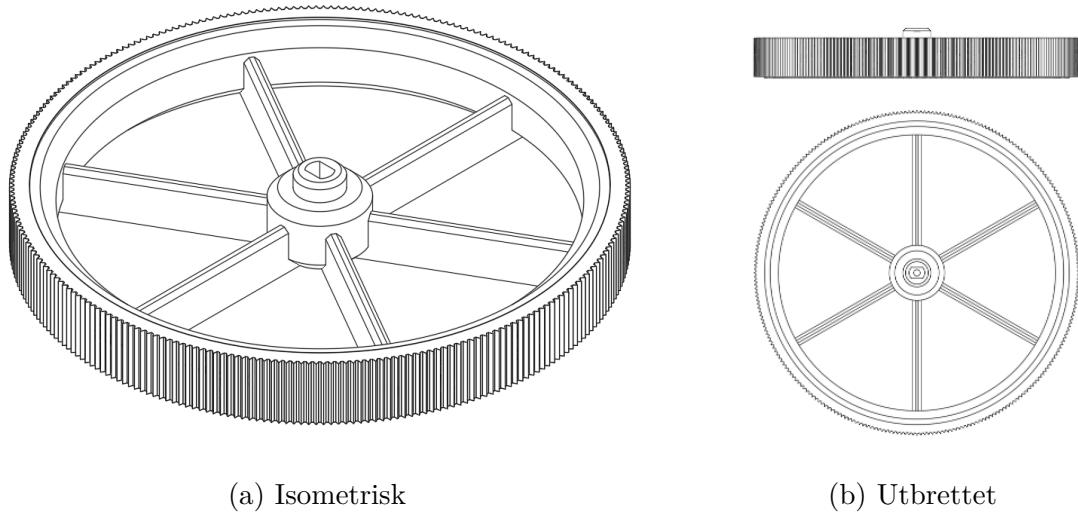
Velteburet har som hensikt å hindre at JetBalancer velter, ettersom dette kan medføre skade på kontrollbordet og øvrige deler. Velteburet er utformet slik at det får bakkekontakt før JetBalancer rekker å få nok moment til at det kan påføre skade.

Velteburet har en sekundær funksjon i at det kan utformes slik at det plasserer JetBalancers tyngdepunkt høyere som gir økt stabilitet på bekostning av økt vekt. Velteburet ble designet i Adobe Illustrator og fabrikkert med Epilog Laserkutter på MakerSpace sin fabrikasjonslab. Velteburet er i 6mm tykkelse MDF treverk.

3.2.3 Hjul

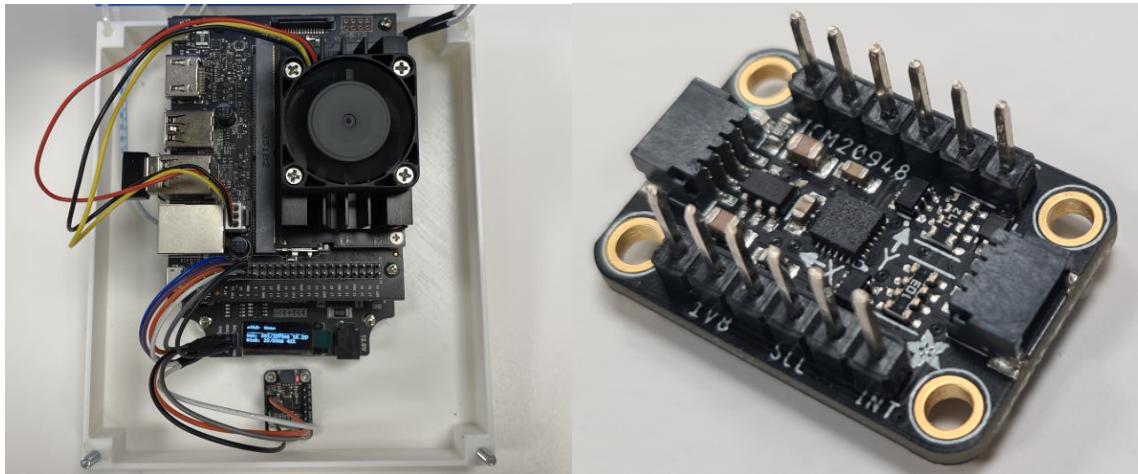
JetBalancer har to hjul. Hvert hjul består av en felg og et dekk som omslutter felgen. Felgen har en diameter på 110mm, en tykkelse på 14,5mm og veier 21g. Felgen er 3D printet i PLA filament og er utformet med seks smale eiker som står ut fra navet. Navet er formet for å passe girmotoren *TT DC Gearbox Motor* med et hull i midten som lar det skrus fast i akselen. Dekket har en diameter på 117,5mm og en tykkelse på 14mm. Dekket er 3D-printet i TPU filament. TPU filamentet er fleksibelt og gir økt friksjon mellom hjulet og bakken.

Størrelsen på hjulene ble bestemt etter mål på hvor stort karosseriet til JetBalancer er. Diameteren til hjulene avhenger av hvor festet til girmotorene står, men også høyden og dybden på karosseriet til JetBalancer. Når JetBalancer skal balansere er det nødvendig at karosseriet ikke hviler på underlaget som den kjører på.



Figur 3.8: Hjul

3.3 Deler lagt til



(a) Kontrollbordet til JetBalancer

(b) ICM-20948 IMU som benyttes i JetBalancer

Figur 3.9: Kontrollbordet med IMU-chip tilkoblet

For å kunne implementere en kontroller for selvbalanseringen la gruppen til en IMU-chip, Figur 3.9(b). IMUen som ble valgt er ICM-20948 levert av Adafruit. Pins som er montert og som kan bli sett på bildet er blitt loddet på av gruppen.

Motorene vil også byttes ut. Hvorfor gruppen bytter motorer og hvilke motorer gruppen bytter til er beskrevet mer i dybden i underseksjon 4.3.4.

Kapittel 4

Implementering og Testing

I dette kapittelet vil gruppen diskutere prosessen for å implementere og teste ideer til JetBalancer. Det vil også være en diskusjon på hvordan det gikk da modulen for autonom kjøring ble testet med JetBot.

4.1 JetBot oppsett

Som nevnt i underseksjon 2.7.2, krever Jetson Nano et operativsystem for å kunne fungere, av den grunn er naturlig å starte med å installere dette. Installeringen gjøres ved å flashe en ISO-fil til en mikro-SD minnebrikke slik at den blir oppstartsbar for NVIDIA Jetson Nano. Det finnes flere programvarer som flasher ISO-filer, eksempelvis kan man benytte Rufus eller Etcher, i dette prosjektet ble det benyttet dd-kommandoen, som følger med de fleste Unix-lignende operativsystemer. ISO-filen som blir benyttet til dette prosjektet er «Jetson Nano 2Gb - JetPack Version 4.5 - JetBot Version 0.4.3» hentet fra guide for oppsett fra JetBot sine hjemmesider. [10].

```
1 sudo dd if='/filtsi/til/systembilde.iso' of='/dev/externt-medium'
```

Kodesnutt 4.1: Eksempel på bruk av dd

Etter installasjonen av ISO-filen har Jetson nå et Ubuntu 18.04.6 LTS operativsystem. Operativsystemet er ferdig konfigurert med web-grensesnitt som kan nås ved å taste inn IP-adressen til Jetson i en nettleser på hvilken som helst datamaskin som er tilkoblet samme nettverk som Jetson.

Sammen med operativsystemet vil også katalogen NVIDIA-AI-IOT/jetbot bli installert, og er programvarene som tillater Jetson å styre JetBot. Programvarene som brukes settes opp i en Dockercontainer og har et nybegynnervennlig web-grensesnitt.

Ved å benytte IP-adressen til Jetson Nano - fra nettleseren på en datamaskin som er tilkoblet samme nettverk som JetBot - kan man logge inn på brukergrensesnittet og få administrasjonstilgang internt i dockercontaineren.

Grensesnittet er utstyrt med Jupyter Notebook som gjør det enkelt å skrive og eksekvere egenprodusert Python-kode.

4.2 Testing av JetBot

Etter oppsettet av JetBot ble det sjekket at JetBot fungerer som forventet ved å gjennomgå de innførende eksemplene i det interaktive kodeeksekveringsmiljøet som følger med JetBot.

Eksemplene *Basic Motion*, *Teleoperation* og *Collision Avoidance* ble valgt til å sjekke JetBots funksjonalitet.

4.2.1 Gjennomgang av *basic motion*

Under gjennomgang av *basic motion* var det observert at begge hjulene kjører i etterspurt hastighet når de riktige funksjonene i JetBot-biblioteket kalles.

4.2.2 Gjennomgang av *Teleoperation*

Under gjennomgang av *Teleoperation* ble det verifisert at kameraet fungerer og at JetBot kan hente bildedata fra kameraet.

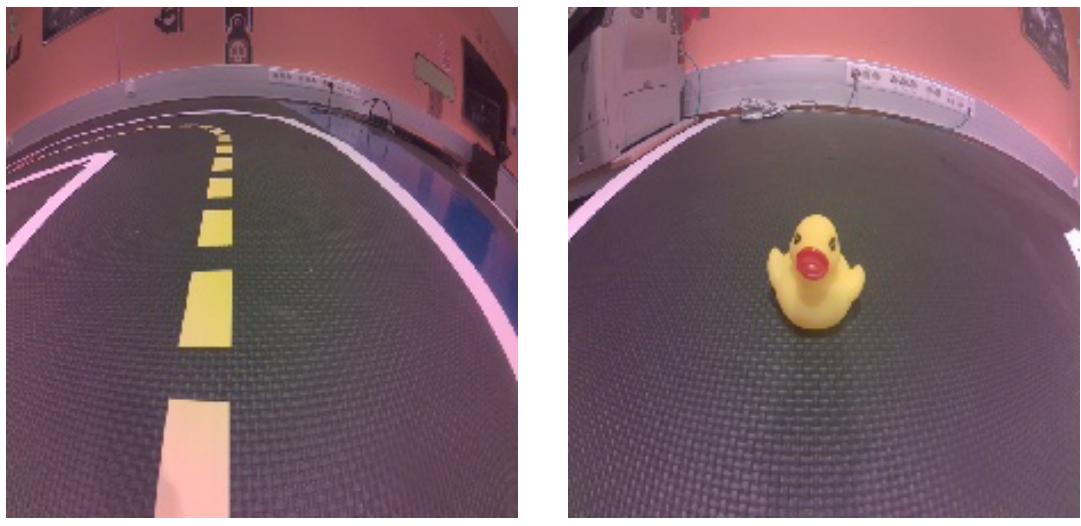
I tillegg var det oppdaget at den trådløse gamepaden er lite responsiv—JetBot reagerer flere sekunder etter at joysticken beveges, og JetBot beveger seg i rykk og napp. Det ble tatt en beslutning på at gamepaden ikke vil bli tatt i bruk.

4.2.3 Gjennomgang av *Collision Avoidance*

Under gjennomføring av *Collision Avoidance* ble det samlet bilder til å trenere maskinlæringsmodellen. Det ble tatt ca. 100 bilder hvor veien til JetBot var blokkert og ca. 100 hvor veien var fri.

Bildene ble brukt til å trenere modellen, og dette ble gjort på selve JetBot-en, og det tok ca. 5-10 minutter.

Etter treningen kunne JetBot fastslå hvorvidt den hadde klar vei eller ikke.



(a) Eksempel på fri vei

(b) Eksempel på blokkert vei

Figur 4.1: Bilder hentet fra JetBots treningsdatasett

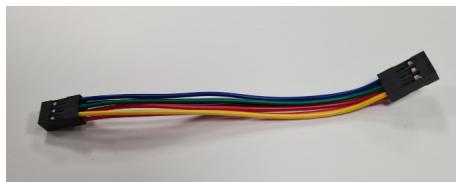
4.3 Testing av enkeltkomponenter

Denne fasen med testing innebærer både modifisering av JetBot, samt en begynnelse på JetBalancer. Modifisering av JetBot innebærer gruppens arbeid med å endre komponenter som er festet til JetBot sitt originale grønne metall-karosseri. Under modifisering av JetBot

var det også et midlertidig feste av IMU. Begynnelsen på JetBalancer innebærer bytting av motor og produsering av nye hjul.

4.3.1 Testing av IMU

IMU-enheten (ICM-20948) kommuniserer via I²C, og siden JetBot originalt benytter I²C-1 bussen til Jetson Nano for styring av motorer og en OLED-skjerm som er fastmontert på Waveshare sitt utvidelseskort, var planen å benytte I²C-0 bussen til IMU-enheten. Å koble IMU-en til I²C-0 byr på utfordringer når det kommer til å benytte ferdige biblioteker til avlesningen av IMU-enhetens sensorer, siden standard innstillingene for bibliotekene som har blitt prøvd benytter I²C-1, og har begrenset dokumentasjon på hvordan man endrer dette.



(a) 3x2 jumper wire, følger originalt med JetBot

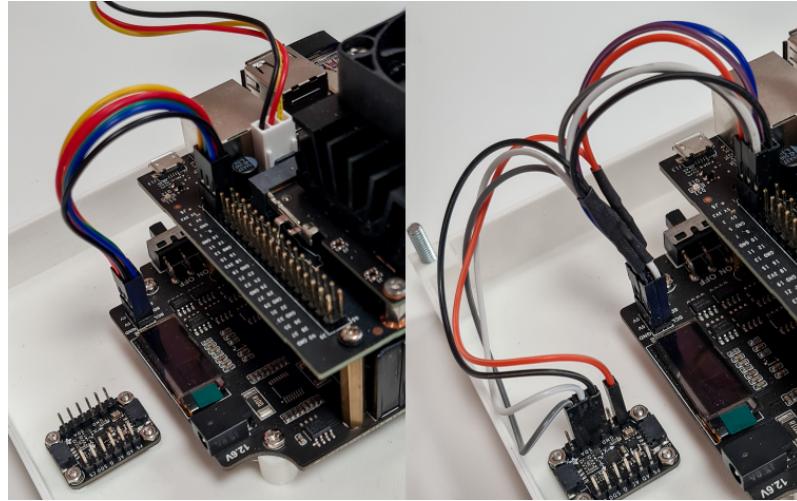


(b) Y-jumper wire (4stk)



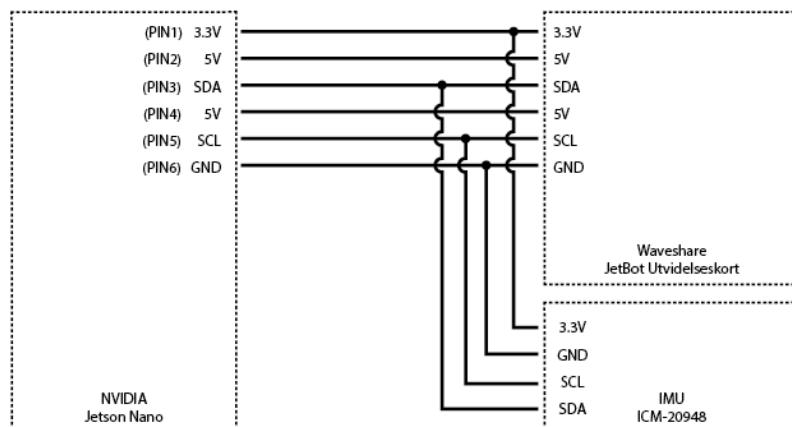
(c) Dobbel jumper wire (1stk)

Figur 4.2: Bilder av jumper wire modifikasjoner



Figur 4.3: Kobling mellom NVIDIA Jetson Nano, Waveshare utvidelseskort og IMU

Gruppen valgte derfor å koble IMU-en inn i I^2C -bussen sammen med komponentene på Waveshare sitt utvidelseskort. For å koble IMU-en på samme I^2C -buss som utvidelseskortet må den originale 3×2 -jumperkablene, Figur 4.2(a) som går mellom utvidelseskortet og Jetson Nano, skiftes ut med fire y-jumperkabler Figur 4.2(b), og en double jumperkabel Figur 4.2(c).



Figur 4.4: Koblingsdiagram mellom NVIDIA Jetson Nano, Waveshare utvidelseskort og IMU

Etter å ha koblet IMU-en til I^2C -bussen kan man lese av sensorverdier ved hjelp av Adafruit sitt Python-bibliotek for IMU-er [13].

Analyse av avlesningstid

Som nevnt tidligere, i seksjon 2.6 har tiden mellom hver måling mye å si for behandlingene av sensorverdiene, dette med tanke på integrasjon av gyroskopet, men også for PID-kontrolleren. For å måle tiden en avlesning fra sensorene tar kan en gjøre dette med følgende Python-kode:

```

1 import board
2 import adafruit_icm20x as ada
3 import time
4
5 i2c = board.I2C()
6 icm = ada.ICM20948(i2c)
7
8 for i in range(10):
9     preMTIME = time.time()*1000
10    gyroValues = icm.gyro
11    #accValues = icm.acceleration
12    #magnetoValues = icm.magnetic
13    postMTIME = time.time()*1000
14    print("Time: ", postMTIME-preMTIME)

```

Kodesnutt 4.2: Kode for henting av tiden en avlesning fra sensorene tar

```

1 import numpy as np
2
3 data = np.array(diffTimeList)
4
5 print('Antall avlesninger:', len(diffTimeList))
6
7 def largest(diffTimeList, n):
8     mx=diffTimeList[0]
9
10    for i in range (1, n):
11        if diffTimeList[i]>mx:
12            mx=diffTimeList[i]
13    return mx
14
15 if __name__=='__main__':
16     diffTimeList.append(diffTime)
17     n=len(diffTimeList)
18     Ans = largest(diffTimeList, n)
19     print('Største verdi: ', round(Ans,2) , 'ms')
20
21 avgTime = np.mean(data)
22 std = np.std(data, ddof=1)
23
24 print('Snitt:', round(avgTime,2), 'ms')
25 print('Standardavvik:', round(std,2), 'ms')

```

Kodesnutt 4.3: Kode for utregning av snitttid mellom avlesninger

Koden i Kodesnutt 4.3 følger Kodesnutt 4.2 og bruker derfor samme biblioteker, i tillegg til `numpy` som her er importert for å kunne gjøre statistisk programmering.

I koden ovenfor refererer `diffTime` til variabelen som dannes av `postMTIME-preMTIME`, listen som inneholder alle disse verdiene blir da `diffTimeList`.

Gjennom testing av varierende iterasjoner oppdager en at når antall iterasjoner øker, så øker størrelsen på ekstremverdien (største verdi), snittverdien, og standardavviket. Figur 4.5

til Figur 4.8 illustrerer utskrift av Kodesnutt 4.3. Økninger av tid mellom avlesninger tyder på at når arbeidsmengden til IMU øker, vil arbeidet den gjør ta lengre tid. Også interessant er det at standardavviket øker, altså blir ventetiden mindre forutsigbar jo høyere arbeidsmengden er. En økning i arbeidstid vil ha en påvirkning på algoritmen for selvbalansering da denne algoritmen vil være avhengig av hyppige og raske avlesninger av IMU-data. Dersom algoritmen blir for treg vil det igjen føre til at JetBalancer ikke klarer å korrigere i tide, og vil lede til at JetBalancer faller.

(a) 200 avlesninger (b) 250 avlesninger (c) 1000 avlesninger

Figur 4.5: Variasjon på avlesningstid fra gyroskopet

Ettersom både gyroskop og akselerometer må brukes for å kunne lage en fungerende selvbalanseringsalgoritme er det viktig å sjekke avlesningstiden til akselerometret i tillegg til gyroskopet. For å hente ut verdier for avlesning av akselerometeret brukes Kodesnutt 4.2, men det er kommentert ut linje 10, og kommentert inn linje 11. Sammenlignes Figur 4.6 med Figur 4.5 kan en se at avlesningstiden er mye lengre for akseleormeteret enn for gyroskopet.

Antall avlesninger: 200 Største verdi: 25.89 ms Snitt: 22.49 ms Standardavvik: 0.75 ms	Antall avlesninger: 250 Største verdi: 27.14 ms Snitt: 22.55 ms Standardavvik: 0.82 ms	Antall avlesninger: 1000 Største verdi: 30.13 ms Snitt: 22.27 ms Standardavvik: 0.71 ms
(a) 200 avlesninger	(b) 250 avlesninger	(c) 1000 avlesninger

Figur 4.6: Variasjon på avlesningstid fra akselerometeret.

Fra Figur 4.6 kan en se at når arbeidmengden til IMU (antall avlesninger) øker, så øker også tiden det tar å få utført arbeidet.

Når JetBalancer kjører og prøver å balansere vil den avlese verdier fra både gyroskop og akselerometer samtidig, det er derfor viktig å vite hvor lang tid den bruker på å lese av fra sensorer samtidig.

I Figur 4.7(c) er største verdi 32,37[ms], men denne kan hoppe opp til 95[ms]. En stor variasjon i tiden det tar å avlese sensorverdier vil ha en negativ påvirkning på balanseringen.

Det neste som ble testet var om arbeidstiden for å hente ut verdier øker når arbeidsmengden øker ved å kjøre motorene samtidig som sensorene avleses. Når JetBalancer faktisk kjører og prøver å holde seg oppe vil den være avhengig av at dens egen maskin og IMU er sterke nok til å klare disse operasjonene samtidig.

Dersom man sammenligner Figur 4.7 og Figur 4.8 ser man at snitt-tiden for avlesninger nesten dobles når motorene skrus på sammenlignet med når motorene er av, i tillegg til at

Antall avlesninger: 200
 Største verdi: 31.5 ms
 Snitt: 23.9 ms
 Standardavvik: 1.19 ms

Antall avlesninger: 250
 Største verdi: 32.76 ms
 Snitt: 23.96 ms
 Standardavvik: 1.29 ms

Antall avlesninger: 1000
 Største verdi: 32.37 ms
 Snitt: 23.98 ms
 Standardavvik: 1.32 ms

(a) 200 avlesninger

(b) 250 avlesninger

(c) 1000 avlesninger

Figur 4.7: Variasjon på avlesningstid fra akselerometeret og gyroskop.

Antall avlesninger: 200
 Største verdi: 102.94 ms
 Snitt: 42.25 ms
 Standardavvik: 9.62 ms

Antall avlesninger: 250
 Største verdi: 101.87 ms
 Snitt: 42.07 ms
 Standardavvik: 9.43 ms

Antall avlesninger: 1000
 Største verdi: 104.4 ms
 Snitt: 42.12 ms
 Standardavvik: 9.64 ms

(a) 200 avlesninger

(b) 250 avlesninger

(c) 1000 avlesninger

Figur 4.8: Variasjon på avlesningstid fra IMU med motor på

standardavviket og ekstremverdien blir større. En økning i alle tidsverdiene tyder på at JetBalancer vil få problemer når balanseringen skal testes.

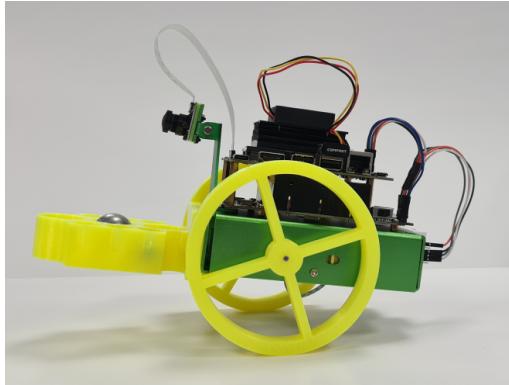
4.3.2 Testing av hjul

Ved design av egne hjul ble det tatt mål av hvor stor diameteren på hjulene måtte være for at ikke karosseri skulle hvile på bordflaten mens det ble testet balanseringsfunksjon. Andre mål som ble tatt var ønsket tykkelse på hjulene, og hva omkretsen på opphenget var. Ved første print viste det seg at det måtte tas hensyn til tykkelsen på filament når målene for printet ble lagt inn. Tykkelsen var ikke en vurdering som programvaren tok hånd om på automatikk, dermed måtte hjulet bli printet på nyt.

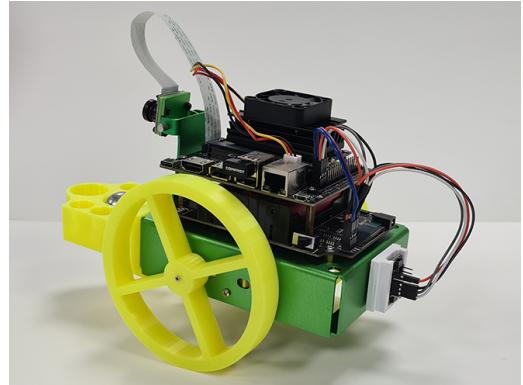
Ettersom tykkelsen på filament i denne omgang kun hadde innvirkning på der hjulet festes til roboten (innerste sirkel) ble kun denne printet delen frem til riktig størrelse ble funnet, dette krevde kun to nye print. Ved å kun printe den mindre delen av hjulet ble det også spart betydelig med tid da total tid på printing gikk fra to timer (per hjul) til 20 minutter (per fest) per print. Da riktig størrelse var funnet valgte gruppen å printe to hjul samtidig med de korrekte målene. Da det ble kjørt parallel print var kun et av hjulene i riktig størrelse. Det er flere variabler som kan ha forårsaket at hjulene ikke ble uniforme, da print ble kjørt på forskjellige printere kan det ha vært forskjell i kalibreringen, det kan også ha vært på grunn av at det ikke ble brukt filament fra samme batch på begge hjulene. Ved bruk av forskjellig filament kan det være variasjon i tykkelse, både ved bruk av to forskjellige typer og ved bruk av to like typer, men fra forskjellig batch.

4.3.3 Vektfordeling

JetBot er baktung når den står på to hjul og kan dermed ikke balansere. For å få JetBot i likevekt ble det produsert en motvekt. Motvekten er justerbar, ved at det kan legges til eller fjernes stålkuler for å endre dens masse.



(a) Bilde av JetBot 2.0, side



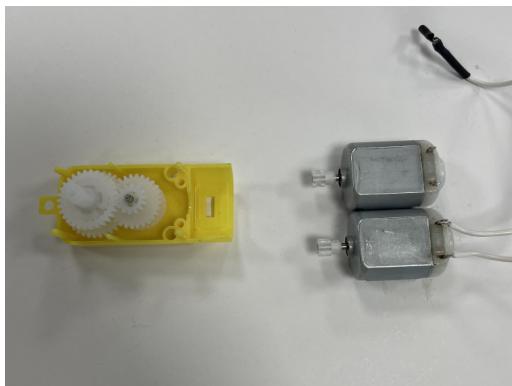
(b) Bilde av JetBot 2.0, bak

Figur 4.9: Bilder av JetBot med modifikasjoner

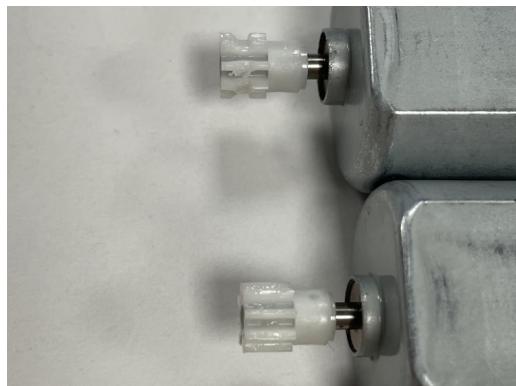
4.3.4 Testing av motor

Med fra JetBot har gruppen forsøkt å bruke to girmotorer av typen *TT DC Gearbox Motor*. Hver av disse motorene har en *gear ratio* på 1:48. Mulig spenning man kan bruke på motorene ligger i intervallet 3 til 6 VDC, men det anbefales å bruke 4,5 VDC. Det står oppført to dreiemoment der motorene vil stanse. Ved 3VDC vil motorene stanse ved 0,4 kg/cm og ved 6VDC vil motoren stanse ved 0,8 kg/cm [14]. Dersom en antar et lineært forhold mellom spenning og motorstans vil motoren stanse ved 0,6 kg/cm ved 4.5VDC.

Ved prosjektstart håpet gruppen på å kunne bruke motorene som fulgte med JetBot. Da motorene ble åpnet og girene ble inspisert var det tydelig at tannhjulene led av slitasje. Slitasjen på girene kommer av at motorene har vært i bruk tidligere, samt at girene er laget av en type plastikk som ikke er egnet for påkjenningen JetBalancer blir utsatt for. Når JetBalancer kjører og forsøker å holde seg oppreist vil motorene måtte utføre små hyppige rykk. Ved hyppige rykk blir det både bråstart og bråstopp for motorene, over tid vil disse rykkene føre til at tannhjulene i girene slites ned. Da gruppen begynte å teste sitt design og begynte å kjøre JetBalancer ble slitasjen verre og girene ble slitt såpass at gruppen besluttet å bytte motor.



(a) Innsiden av motor - oversikt

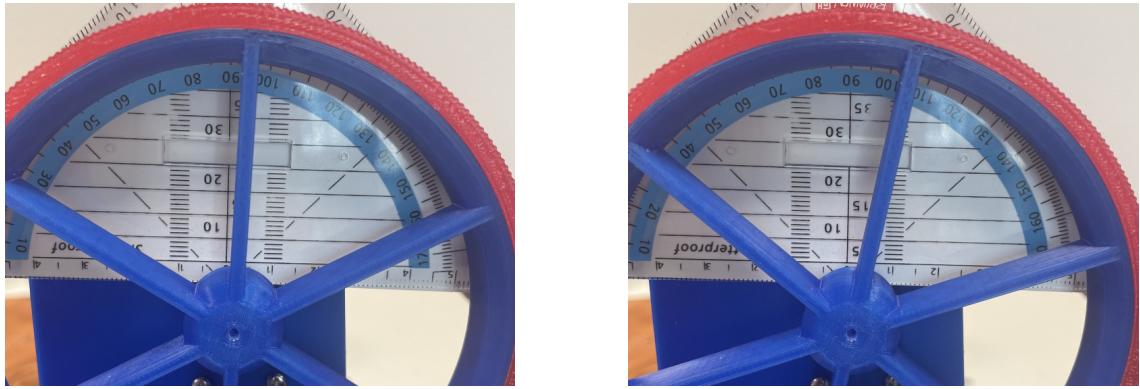


(b) Slitasje på giret, sammeligning

Figur 4.10: Bilder av innsiden og slitasjen av girmotorene

På Figur 4.10(b) er det mulig å se at giret på den øverste motoren er nærmest helt smeltet.

Valget på nye motorer falt på *Encoder TT DC Gearmotor* [15], og det er samme oppheng på de nye motorene og de motorene som JetBot kom med og hjulene passet fortsatt på de nye motorene. Fordelen med nye motorer er at de kom med mindre slitasje enn motorene gruppen tok fra JetBot. Ulempen er at girene er av samme materiale som de gamle motorene og de nye motorene vil med tiden oppleve samme type slitasje som de gamle motorene.



(a) Slark i hjulet, minimum vinkel

(b) Slark i hjulet, maksimum vinkel

Figur 4.11: Slark i hjulene på grunn av slitasje på motorene

Figur 4.11 illustrerer hvilken påvirkning motorenes slitasje har på hjulene. I (a) er hjulene i en vinkel på 95 grader, mens i (b) er hjulene i en vinkel på 105 grader. Bildene i Figur 4.11 viser hvor mye en kan rotere på hjulene før girene i motorene begynner å jobbe, og bildene forsøker å vise at det er en rotasjon på 10 grader før motorene begynner å jobbe.

Gjennom prosjektet ble det testet to sett med motorer til JetBalancer.



Figur 4.12: Motorer som ble testet

Motoren til venstre (gul) er den originale motoren som ble levert med JetBot. Motoren til høyre (blå) er fra en *Encoder TT DC Gearmotor*. Samenligner man disse motorene har begge nominellspenning mellom 3-6V, mens girreduksjonen og dreiemoment er forskjellig.

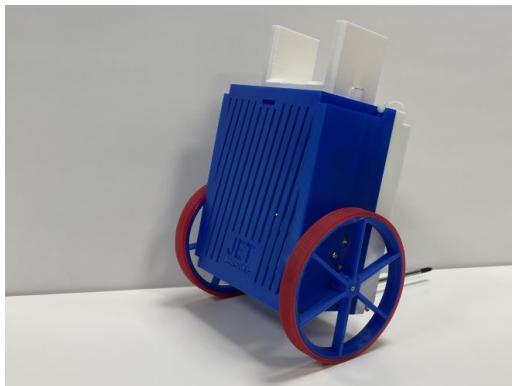
	Gul	Blå
Gir	1:48	1:45
Dreiemoment	0.6 Nm	1.2 Nm
Hastighet	$200 \pm 10\%$ rpm	$155 \pm 5\%$ rpm

Tabell 4.1: Motordetaljer

4.4 Prototype Karosseri

4.4.1 Karosseri

Prototypen til karosseriet er fullstendig 3D-printet med PLA-filament og avbildet under i Figur 4.13. For å få nok friksjon mellom hjulene og underlaget er det printet et dekk til hvert hjul i fleksibel TPU filament. Da hjulene ble testet uten dekk fant gruppen at hjulene i ren plastikk var for glatte til å kunne danne friksjon med underlaget, som førte til at hjulene bare spant på stedet og ikke klarte å kjøre fremover eller bakover. Gruppen valgte dermed å 3D-printe et dekk til hjulene for å kunne øke mengden friksjon mellom hjulene og underlaget.



(a) Bilde av JetBalancer, side



(b) Bilde av JetBalancer, front

Figur 4.13: Prototype karosseri av JetBalancer, med dekk i rødt

4.4.2 Karosseri med støtter

Til karosseriet har det blitt lagt til støtter. Da gruppen begynte med testing av algoritmer for selvbalansering ble det klart at JetBalancer hadde en tendens til å falle i bakken med relativt stor kraft. Et fall med stor kraft kan føre til støtskader på karosseri og støtet kan føre til skader på IMU og datamaskinen. For å unngå støtskader har gruppen valgt å legge til støtter til karosseriet. JetBalancer med støtter er avbildet i Figur 4.14. Støttene er av laserkuttet MDF treverk, ellers er karosseriet det samme som i seksjon 4.4.



Figur 4.14: JetBalancer karosseri med støtter

4.5 Kodelogikk

Programkoden er skrevet i kodespråket Python og kjørt med Python versjon 3.6.9.

4.5.1 Bibliotek og instansiering av objekter

Bibliotekene benyttet i dette prosjektet er; time, numpy, jetbot og adafruit sine biblioteker, board og adafruit_icm20x. `Time` brukes til å hente et tidspunkt slik at koden skal vite hvor lang tid har gått fra en måling til neste. `Numpy` blir brukt til å kalkulere vinkelen θ ved hjelp av metoden «`arctan2(y,x)`». For å sende signaler til motorene er det en egen klasse som følger med JetBot-biblioteket som heter «`Robot`». Klassen `Robot` har to metoder som blir benyttet, «`set_motors(left, right)`» tar to float mellom -1 og 1 og setter en spenning mellom -12V til 12V over motorene. Biblioteket `adafruit_icm20x`, er et samlebibliotek for flere sensorer og har en egen konstruktør for ICM20948 som tilhører IMU-enheten dette prosjektet benytter. Konstruktøren «`adafruit_icm20x.ICM20948(board.I2C())`» tar imot et parameter som er et objekt av I^2C -bussen IMU-enheten er koblet til. Klassen til I^2C -bussen ligger i biblioteket `board` og instansieres med konstruktøren `board.I2C()`. Siden det ikke brukes parametere i `board.I2C()`, vil objektet benytte standard I^2C -buss som er I^2C-1 .

```

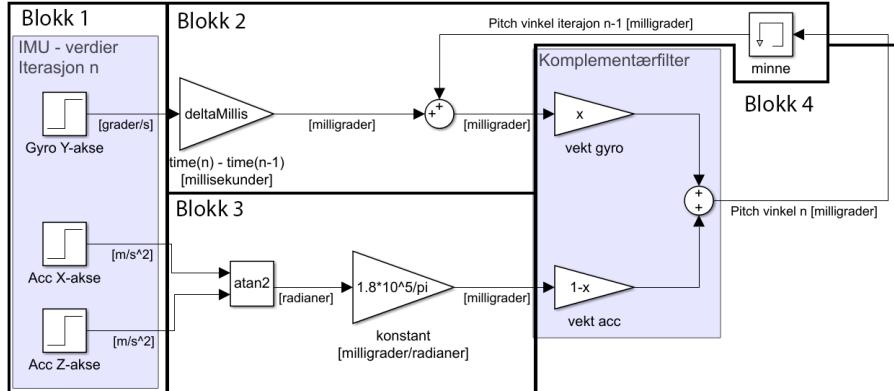
1 import time
2 import board
3 import numpy as np
4 import adafruit_icm20x as ada
5 from jetbot import Robot
6
7 robot = Robot()
8 icm = ada.ICM20948(board.I2C())

```

Kodesnutt 4.4: Importering av biblioteker og instansiering av objekter

4.5.2 Lesing og behandling av sensordata

I underseksjon 2.6.7, blir det beskrevet hvordan sensordataene skal handteres. Figur 2.17 kan deles inn i fire blokker.



Figur 4.15: Behandling av sensordata, inndelt i blokker

Blokk 1: Sensor avlesning

Avlesning av sensordata gjøres med ICM objektet, som har en metode for hver av sensorene; gyroskop, akselerometer og magnetometer. I dette prosjektet er det gyroskopet og akselerometeret som er av interesse, og for å lese av verdiene binder man dette til en variabel.

```

1 gyro = icm.gyro
2 # gyro = (x, y, z)
3 gy = gyro[1]
4
5 acc_data = icm.acceleration
6 # acc = (x, y, z)
7 acc_x, acc_z = (acc_data[0], acc_data[2])

```

Kodesnutt 4.5: Lesing av data fra IMU-enheten

icm.gyro og icm.acceleration er metoder som sender en forespørrelse ut på I^2C -1 bussen til IMU-enheten og spør etter verdiene som er lagret i IMU-enhetens register og returnerer disse som en tuple. På grunn av orienteringen til IMU-enhetens akser er gruppen i dette prosjektet kun interessert i gyroskopets y-akse og akselerometerets x- og z-akse (se Figur 2.8).

Blokk 2: Behandling av gy

Før programmet starter å balansere JetBalancer, må gyroskopets partiskhet beregnes. Ved å kalle på en funksjon med en **for**-løkke som summerer flere målinger fra gyroskopet for å så ta gjennomsnittet av den totale verdien finner man bias til gyroskopet. Alle målinger for gy etter kalibreringen, vil da bli trukket fra bias for å kompansere for gyroskopets partiskhet. Når programmet genererer bias-variablene er det viktig at JetBalancer står i ro, dette er fordi en hver bevegelse av gyroskopet vil ha utslag på bias-variablene.

```

1 def caliGyrY(numLoops):
2     bias = 0

```

```

3     for i in range(numLoops):
4         bias += rad2mdegF(icm.gyro[1])
5     return bias/numLoops
6
7
8 bias = caliGyrY(1000)

```

Kodesnutt 4.6: Beregning av gyroskopets bias rundt y-aksen

Som man kan se i Figur 4.15, er blokk 2 en del av et feedback system, dette betyr at programmet krever initialbetingelser for pitch-vinkelen (prePitch) og tidspunktet fra tidligere iterasjon (preTime). Variablen preTime er tidspunktet før vi bestemmer initialbetingelsen til pitch-vinkelen. Initialbetingelsen til pitch-vinkelen leses fra akselerometeret. Initialbetingelsene settes før programmet går inn i hovedløkken.

```

1 preTime = time.time()
2 prePitch = rad2mdegF(- np.arctan2(acc_z, acc_x))
3
4 while(balancing) # hovedlokke
5     curTime = time.time()
6     # ...

```

Kodesnutt 4.7: Initialbetingelser for feedback systemt til programmet

I hovedløkken holder variablen gy på en verdi for «nåværende» vinkelhastighet rundt gyroskopets y-akse, denne har benevningen [radianer per sekund] og må av den grunn konverteres til milligrader for at programmet skal kunne benytte den i komplementærfilteret. For å konvertere gy til milligrader gjøres ved å multiplisere gy med med $180000/\pi$ og deltaTime. Variablen deltaTime er tiden fra forrige iterasjon av hovedløkken til nåværende iterasjon.

```

1 def rad2mdeg(radians):
2     return radians * 180000 / np.pi
3
4
5 while(balancing)
6     curTime = time.time()
7     deltaTime = curTime - preTime
8     gy = (rad2mdeg(- icm.gyro[1]) - bias) * deltaTime
9     #...
10    preTime = curTime

```

Kodesnutt 4.8: Lesing og behandling av data fra gyroskop

Blokk 3: Behandling av acc

Når dataene fra akselerometeret blir lastet inn, tar vi vare på verdiene til z- og x-aksene for så å kalle på metoden `arctan2()` fra numpy-biblioteket. Metoden `arctan2()` returnerer vinkelen til JetBalancer i radianer, av den grunn må denne verdien konverteres til milligrader før det kan benyttes i komplementærfilteret.

```

1     acc_data = icm.acceleration
2     acc = rad2mdeg(- np.arctan2(acc_data[2], acc_data[0]))

```

Kodesnutt 4.9: lesing og behandling av data fra akselerometer

Merk her at vi bruker negativ verdi fra `arctan2()`, tilsvarende med `icm.gyro[1]` fra Kodesnutt 4.8, dette er så vinkelen 0 milligrader når JetBalancer står oppreist og at en vinkling fremover er positiv retning.

Blokk 4: Komplementærfilter

Først når blokk 2 og 3 er har konvertert benevningene til gyroskopet og akselerometerdataene kan sensordataene fusjoneres sammen, til én verdi. Vektingen er 98% vinkelen fra forrige iterasjon summert med gyroskopets nåværende verdi og 2% fra akselerometerets nåværende verdi.

```
1 curPitch = (acc*0.02 + (gy + prePitch)*0.98)
```

Kodesnutt 4.10: Komplementærfilter

Resultatet etter komplementærfilteret er pitch-vinkelen i milligrader til JetBalancer. Takket være komplementærfilteret vil pitch-vinkel ikke drifte og inneholde relativt lite støy. Pitch-verdien vil da bli sendt til PID-kontrolleren.

4.5.3 PID-kontroll

Implementasjonen av PID-kontrolleren er en funksjon som regner ut PID-verdien basert på nåværende vinkel og tidsendring siden forrige iterasjon.

Funksjonsobjektet har integralet av vinkelen og avviksvinkelen fra forrige iterasjon lagret som instansvariabler, slik at de kan lagres mellom iterasjoner.

En ekstra parameter `c` skalerer hele PID-verdien, men er strengt tatt ikke nødvendig, da denne kan ganges inn i PID-koeffesientene, men den er faktorert ut i håp om å gjøre koden lettere å lese og justere.

```
1 def PID(curPitch, deltaTime):
2     c = 1/40000
3     kP, kI, kD = 1.1, 1, 0.01
4     offsetPitch = 4000
5     reffPitch = 4000
6
7     error = curPitch - reffPitch
8     if abs(error) > 25000:
9         # give up balancing
10        PID.integral = 0
11        return 0
12
13    PID.integral += error * deltaTime
14    derivative = (error - PID.preError)/deltaTime
15
16    p = error * kP
17    i = PID.integral * kI
18    d = derivative * kD
19
20    PID.preError = error
21    return (p + i + d)*c
22
23 PID.integral = 0
24 PID.preError = 0
```

Kodesnutt 4.11: PID implementert i Python-kode

Implementasjonen av integralet og den deriverete her stemmer ikke helt med formelene i underseksjon 2.6.6, men det er fordi implementasjonen er ment til å være foreløpig, og ment til å erstattes med formlene i underseksjon 2.6.6 etterhvert. På grunn av tidspress har arbeid på implementasjon blitt avsluttet før formlene i underseksjon 2.6.6 skulle implementeres.

En refaktorert implementering av PID-kontrolleren ville muligens ha avviksvinkelen som parameter i stedet for den målte vinkelen, fordi referansevinkelen ikke ville vært konstant i en ferdig implementasjon, og fordi den ikke ville vært relevant i seg selv i PID-logikken, kun avviket fra den.

Kapittel 5

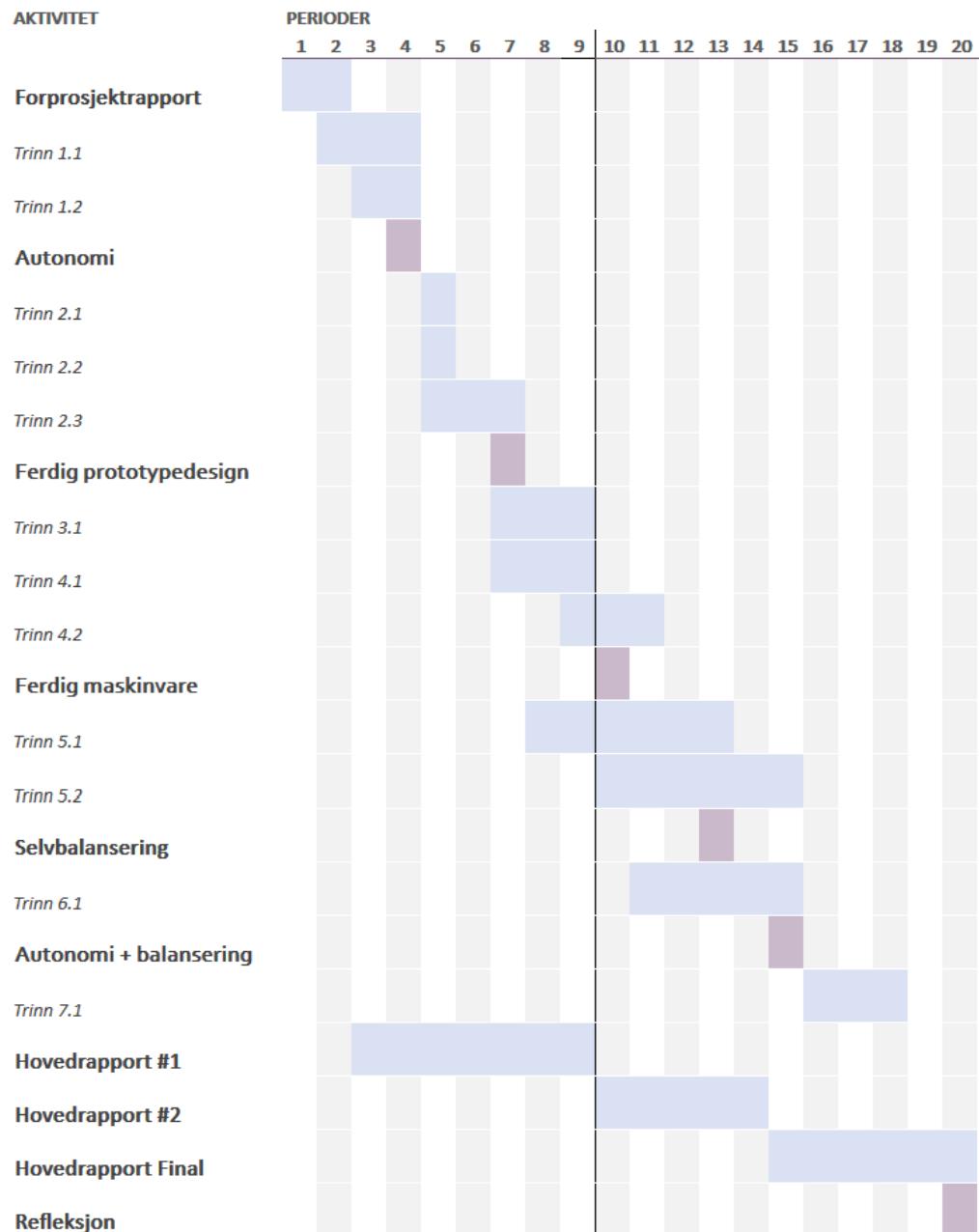
Diskusjon

I dette kapittelet vil gruppen ha en diskusjon om blant annet opplevelsen av prosjektarbeidet, hvilke lærdommer gruppen har fått gjennom prosjektet, hvorvidt prosjektet anses som en suksess, og om gruppen har klart å holde seg til Gantt-diagrammet fra underseksjon 1.7.1.

5.1 Prosjektplan

Under seksjon for prosjektplan sammenligner gruppen Ganntdiagrammet fra begynnelsen av semesteret, som ble introdusert i underseksjon 1.7.1, med Ganntdiagram som har blitt oppdatert etter hvert som prosjektet har gått sin gang.

5.1.1 Første oppdatering av plan



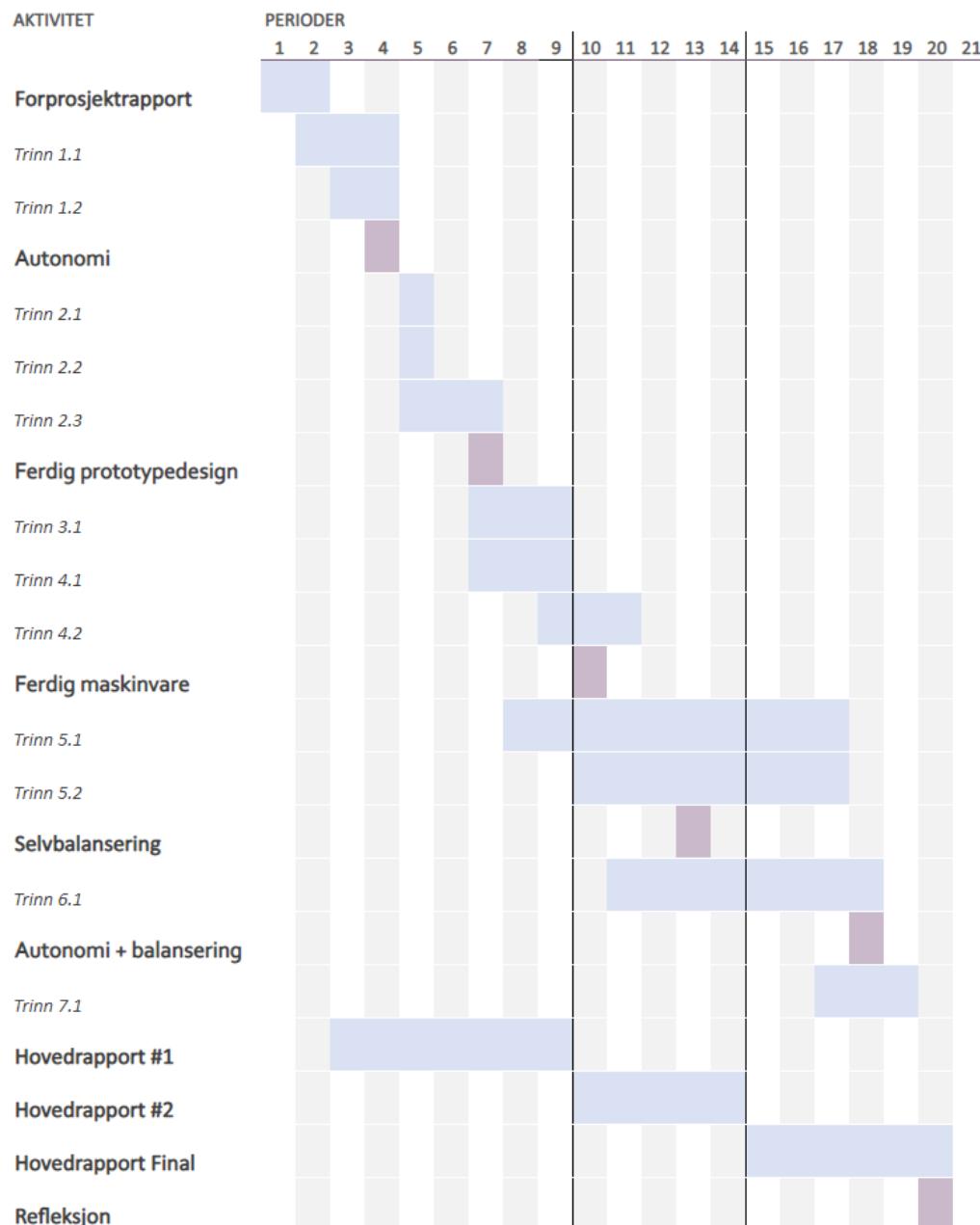
Figur 5.1: Oppdatert Ganndiagram, 9 uker etter oppstart

Figur 5.1 viser hvordan fremgangen har vært til og med prosjektuke 9. Sammenlignes Figur 5.1 med Ganttdiagrammet fra underseksjon 1.7.1 ser en at gruppen ligger en til to uker bak planen. Forsinkelsen skyldes utfordringer med kommunikasjonen mellom IMU enheten og NVIDIA Jetson Nano.

Siden I²C-buss-1 blir benyttet av Jetbot til å sende data mellom Nano og utvidelseskortet fra Waveshare, var planen i utgangspunktet å benytte to I²C-busser, I²C-buss-1 for utvidelseskortet, og I²C-buss-0 for IMU. Gruppen forsøkte spesifisere I²C-buss-0 i Python programkode, men etter noen dager med forsøk endte gruppen opp med å lage fire y-adapttere slik at en kunne koble IMU på samme I²C-buss som utvidelseskortet. Etter at gruppen koblet IMU enheten til I²C-buss-1 var implementasjonen langt lettere siden dette er standard I²C-bussen i bibliotekene til IMU enheten.

Den andre utfordringen kom av at Adafruit har to pythonbiblioteker for IMU-enheten (ICM20948). Et av disse bibliotekene er dedikert til ICM20948 [16], mens det andre er et samlet bibliotek for flere enheter [13]. Først benyttet gruppen det dedikerte biblioteket, målingen fra gyroskopet virket å ha mistenkelig mye støy. Etter å ha prøvd med to forskjellige IMU enheter med en rekke forskjellige innstillinger på både Jetson Nano og en Raspberry pi, ble samlebiblioteket [13] funnet.

5.1.2 Andre oppdatering av plan



Figur 5.2: Oppdatert Ganndiagram, 14 uker etter oppstart

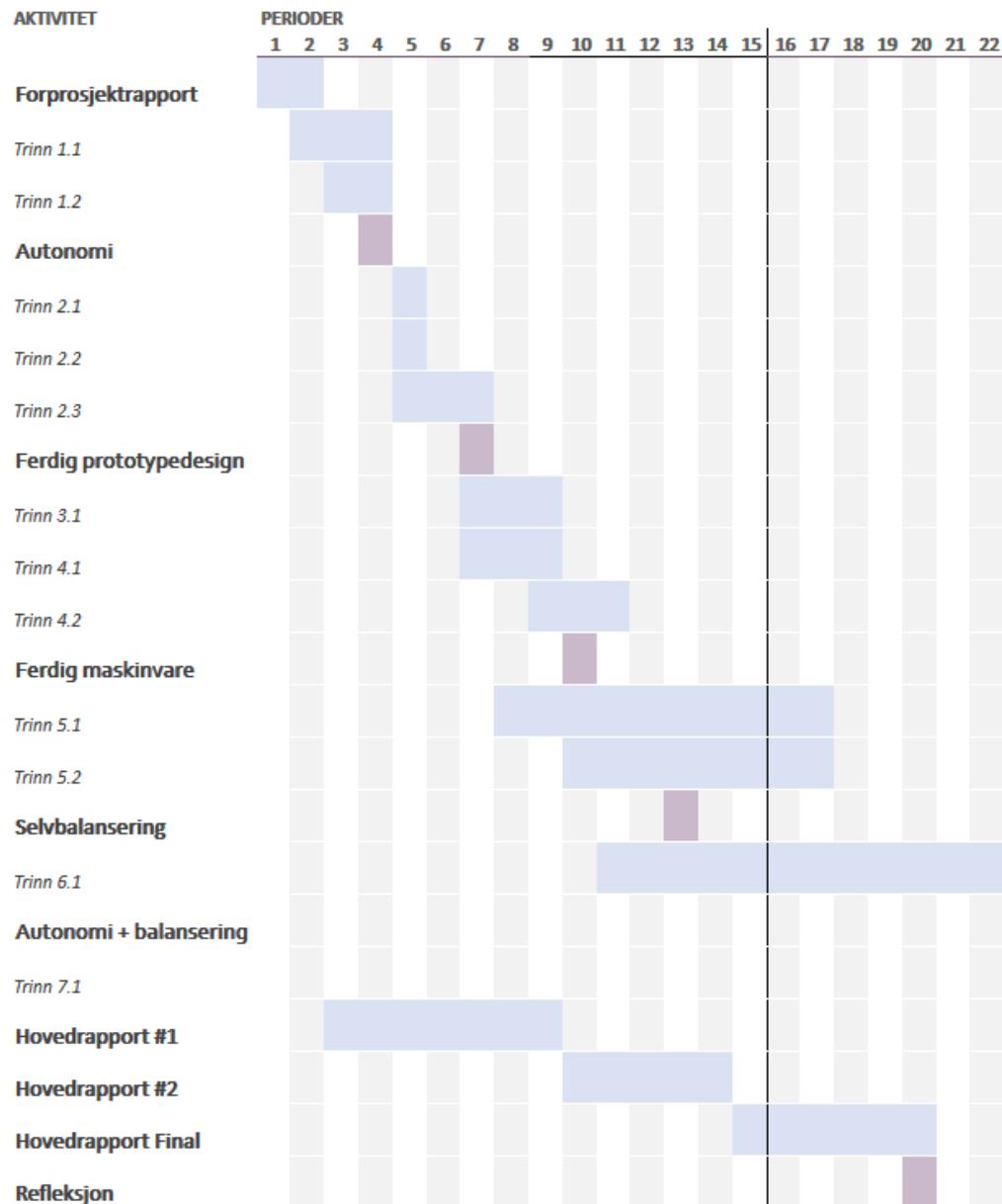
Figur 5.2 viser fremgangen til og med prosjektuke 14. I følge tidligere Gantt-diagram i underseksjon 1.7.1 skulle gruppen vært nærmest ferdig med selvbalanseringen innen uke 14. Forsinkelsen er sammensatt av flere problemer som gruppen har støtt på gjennom prosjektets gang. Det har vist seg at tannhjulene i girmotoren som får hjulene til å snurre gir for mye, og dette gjør det vanskelig å implementere en fungerende selvbalanseringsalgoritme.

Ved forrige oppdatering av Gantt i underseksjon 5.1.1 ble det antatt at gruppen ville kunne begynne testing med endelig prototype av karosseriet uken etter. I mellomtiden

har gruppen hatt noen feilprint av deler til karosseriet, som har ført til noen forsinkelser. Hver feilprint gjennom prosjektet har kostet omtrent en dags forsinkelse, til sammen vil forsinkelser på grunn av feilprint utgjøre ca én uke.

Fremover ser gruppen for seg at vi kommer til å følge den reviderte planen. Gruppen kommer til å fokusere på selvbalanseringen og resten av koden fremover. Gruppen ser for seg at vi må stoppe arbeidet to uker før innleveringsfristen for å ha god nok tid til å skrive en god rapport. Om gruppen ikke blir ferdig med selvbalansering og all kode før den tid antar de at de har kommet langt nok på vei til å fortså detaljene som mangler for at det hele skal fungere.

5.1.3 Endelig oppdatering av plan



Figur 5.3: Oppdatert Ganndiagram ved prosjektets ende

Figur 5.3 viser fremgangen gjennom hele prosjektperioden. Som man kan se av Figur 5.3 har ikke gruppen kommet i mål med prosjektet, men gruppen mener selv at selbalanserende løsning kan oppnåes ved tuning av PID-kontroller.

5.2 Mål

Målene ble introdusert i seksjon 1.4. I seksjonen som følger vil gruppen diskutere hvorvidt delmålene og hovedmålet er fullført.

5.2.1 Delmål 1

Til dette delmålet satte gruppen mål om å trenere modellen for autonom kjøring som kommer med JetBot. Da JetBot ble utlevert hadde studentene også tilgang til moduler for autonom kjøring som var ferdig skrevet av NVIDIA. Delmålet gikk da ut på å implementere disse modulene for å forsikre at Jetson Nano fungerte som den skulle. Testingen av modulene ble diskutert i seksjon 4.2. I seksjonen for testing av modulene kommer gruppen frem til at modulene for autonom kjøring fungerer. Delmål 1 regnes som fullført.

5.2.2 Delmål 2

Til dette delmålet har gruppen definert at det skulle designes, produseres, og implementeres et karosseri for JetBalancer. Designet, produksjonen og implementasjonen av karosseriet til JetBalancer er beskrevet i detalj i seksjon 3.2 og seksjon 4.4. Delmål 2 regnes som fullført.

5.2.3 Delmål 3

Til delmål 3 har gruppen definert at det skulle lages programkode som gjorde at JetBalancer kunne balansere på to hjul, kjøre fremover og bakover, og svinge. Gruppen har ikke lykkes med å fullføre en programkode for selvbalansering, men slik funksjonalitet er blitt påbegynt. Forklaring på teori og en forklaring på kodelogikken kan en finne henholdsvis i seksjon 2.6 og seksjon 4.5. Delmål 3 regnes derfor som påbegynt, men ufullstendig.

5.2.4 Delmål 4

Til delmål 4 har gruppen definert at JetBalancer skulle håndtere både selvbalansering og autonom kjøring samtidig. Da delmål 3 ikke er fullført har gruppen heller ikke klart å fullføre delmål 4.

5.2.5 Hovedmål

Som hovedmål har gruppen definert at målet var å produsere en robot på to hjul som er både autonom og selvbalanserede. Delmål 3 og 4 er ufullstendige, og hovedmålet må derfor også regnes som ufullstendig.

5.3 Leveranser

Leveranser ble først introdusert av gruppen i seksjon 1.5. Dersom en leveranse ikke blir levert vil gruppen komme med en forklaring på hvorfor.

5.3.1 Dokumenter

Gjennom semesteret er tidligere versjoner av rapporten levert til satte innleveringsfrister. Denne endelige versjonen av rapporten vil også leveres til innleveringsfristen. I tillegg vil