Universitetet i Sørøst-Norge
Kongsberg, Vår 2023

University of South-Eastern Norway

Lone Wolf 2023

Dokumentasjon til presentasjon 3

Leif Arne Ulvestad Bastesen

Vegard Skårdal Brenna

Joachim Jamtvedt Børresen

Martin Jørgensen

Sigurd Sætherø Spangelo

Vebjørn Aleksander Østlie

Veileder: Roger Werner Laug

Veileder: Kristian Sandaa

Veileder: Sindre Bøe

Veileder: Chris Andre Brombach

Bacheloroppgave i ingeniørfag

Universitetet i Sørøst-Norge

Dette gruppearbeidet gjennomføres som et ledd i Bachelorstudiet i ingeniørfag ved Universitetet i Sørøst-Norge. Gruppen består av studenter fra følgende linjer med studieretning; dataingeniør - Cyber Physical Systems, maskiningeniør - Produktutvikling og elektroingeniør - Kybernetikk

Forord

Bachelorprosjektet LoneWolf 2023 består av 6 studenter ved Universitetet i Sørøst-Norge nærmere bestemt 3 dataingeniørstudenter (Cyber Physical Systems), 2 maskiningeniørstudenter (Produktutvikling) og 1 elektroingeniørstudent (Kybernetikk) og prosjektet har som formål å undersøke, analysere og utvikle et eller flere «delprodukter» til Can-AM Outlander 650 ATV (heretter kalt ATV) for utvidet fjernstyrt funksjonalitet av ATV. Vi vil takke Roger Werner laug (Ekstern veileder, KDA, generell), Sindre Bøe (Ekstern veileder, KDA, system design), Kristian Sandaa (Ekstern veileder, KDA, hardware), Chris André Brombach (Ekstern veileder, KDA, Software) og Joakim Bjørk (Intern veileder, USN) for innspill, tilbakemeldinger og teknisk veiledning hittil i prosjektet. Oppgaven er organisert i kapitler der hvert kapittel tar for seg viktige og kronologiske aspekter ved problemstillingen. Vi har møtt på utfordringer underveis, inkludert [Sykdom, mangel på dokumentasjon og ingen godkjenning fra USN til å parkere ATV på skolens område], og vi har gjort vårt ytterste for å overkomme dette på best mulig måte for å kunne levere resultater innenfor avtalt tidsramme. Vi håper at vår oppgave kan gi leseren en god oversikt over LoneWolf kjøretøyet og en inngående og brukervennlig forståelse av våre utvidelsesimplementasjoner på kjøretøyet.

Universitetet i Sørøst-Norge

Kongsberg, Mai 2023

Leif Arne Bastesen	Vegard Brenna
Joachim Børresen	Martin Jørgensen
Sigurd Sætherø Spangelo	Vebjørn Østlie

Sammendrag

Rapporten beskriver prosessen og resultatene av et prosjekt for å designe et nytt automatisk girskiftsystem for en all-terrain vehicle (ATV). Rapporten begynner med en introduksjon til prosjektet, inkludert verktøy og prosjektledelsesmetodikk. Rapporten går deretter videre til å detaljere bruken av Scrum-modellen, og rollene og verktøyene som ble brukt i prosjektet. Rapporten tar for seg bachelorgruppen sitt tekniske arbeid innen mekanikk, elektronikk, og programvare utvikling, og gruppen beskriver gjennom dokumentet sin designprosess og utfordringer med løsninger.

Forkortelser

ADC: Analog-to-digital converter

ACM: Abstract control model

AP: Autonom Programvare

ATV: All-Terrain Vehicle

ATV: Can-Am Outlander 650 også kalt «kjøretøyet»

BJT: Bipolar junction-transistor

C: Capacitor (Norsk: kondensator) (kretstegningsymbol)

cd: Change directory (Linux kommando)

DAC: Digital-to-analog converter

DC: Direct-current (Norsk: likestrøm)

DIP: Dual-in-line (socket)

EMS/EMF: Elektromotorisk spenning/electromotive force

enum: Enumeration (datatype)

EOS: End-of-service

EXT: Extend (aktuatorbevegelse)

FDB: Feedback (Norsk: tilbakekobling)

FEM: Finite Element Method

float/Float: Floating-point number

GUI: Graphical User Interface (Norsk: grafisk brukergrensesnitt)

H: Høy (gir)

IC: Integrated Circuit (Norsk: integrert krets)

IDE: Integrated development environment (Norsk: integrert utviklermiljø)

int/Int: Integer (Norsk: heltall) (datatype)

IO: Input-output

IPxx: Ingress protection (f.eks. IP44, IP66 ...)

IR: Impulsrespons

KDA: Kongsberg Defence & Aerospace

L: Lav (gir)

LA: Lineær aktuator (f.eks. LA14)

LED: Light-emitting diode (Norsk: lysdiode)

LTSpice: Linear Technologies Spice (kretsdesignprogram)

LW: Lone Wolf

LW-GCA: Lone Wolf Gear Controller, Arduino

M: Motor (kretstegningsymbol)

MCU: Microcontroller Unit (Norsk: mikrokontrollerenhet)

N: Nøytral (gir)

NCS: Natural color system

Op-amp: Operational Amplifier (Norsk: operasjonsforsterker)

P: Park (gir)

PCB: Printed circuit board

PID: proportional, integral, derivative (-controller)

Pot: Potmeter (variabel resistor-komponent)

PPTC: Polymeric Positive Temperature Coefficient Device (selvgjenopprettende sikring)

PU: Processing Unit (Norsk: prosesseringsenhet) - Navn for datamaskinen montert på ATV'en

PWM: Pulse-width modulation (Norsk: pulsbreddemodulasjon)

R: Revers (gir)

R: Resistor (kretstegningsymbol)

RAM: Random-access memory

RC: Resistor-capacitor (analogt filter)

rcl: ROS Client Library

rclc: ROS Client Library for C

rclcpp: ROS Client Library for C++

rclc_cppb: rclc C++ bindings (bibliotek)

RET: Retract (aktuatorbevegelse)

ROM: Read-only memory

ROS: Robot Operating System

ROS2: Robot Operator System 2

RPC: Remote Procedure Call(s) (Maskinvaresystem for tverr-kjernede funksjonskall)

SPICE/Spice: Simulation Program with Integrated Circuit Emphasis

TF/tf: Transferfunksjon

tfest: Transferfunksjonestimator (Matlab-verktøy)

TP: Trim-point (målepunkt for finjustering på kretskort)

Trimpot: Trim-potmeter (miniatyr-potmeter montert på kretskort, som finjusteres med skrujern)

TUI: Text-Based User Interface (Norsk: tekstbasert brukergrensesnitt)

tty: Teletypewriter

UI: User Interface (Norsk: brukergrensesnitt)

uint/UInt: Unsigned integer (Norsk: positivt heltall) (datatype)

USB: Universal Serial Bus

USN: Universitetet i Sørøst-Norge

VCC/Vcc: Voltage-common-collector

VDC: DC-Voltage (likestrømsspenning)

Vin/VIN: Voltage-in(put) (spenningsinngang)

VSCode: Visual Studio Code (IDE)

WSL: Windows Subsystem for Linux

Innhold

1 Innledning

Dette kapitellet tar for seg en kort introduksjon av gruppen, oppgaven og prosjektverktøy som er blitt benyttet.

1.1 Forberedelser

Diskusjoner om å danne en bachelorgruppe startet ved semesterstart høsten 2022, og gruppesammensetningen ble bestemt før KDA besøkte Campus Kongsberg i forbindelse med "Your Extreme Tourden 19. september. Gruppelederen fremmet gruppens interesse i å skrive en bacheloroppgave for KDA under en samtale med Roger Werner Laug i turbussen. Det ble lagt vekt på den brede kompetansen i gruppen, som inkluderte tre disipliner av ingeniørstudenter, og vårt ønske om å vise frem vår kompetanse, kreativitet og beslutningsevne. Etter flere digitale møter over telefon, korrespondanse via e-post og til slutt et fysisk møte i teknologiparken, ble gruppen tildelt en bacheloroppgave for KDA.

1.2 Oppgavebeskrivelse

Lone Wolf er et tidligere bachelor- og sommerprosjekt som ble startet av Kongsberg Defence & Aerospace (KDA) Division Land Systems (DLS). Prosjektet har som mål å utvikle systemer som kan fjernstyre en All-terrain vehicle (ATV). Så langt har det blitt utviklet systemer for gasspådragskontroll, brems og styring som kan styres via enten datamaskin eller fjernkontroll. I tillegg benyttes forskjellige navigasjonssystemer som gjør det mulig for ATV-en å navigere gjennom terreng og oppdage hindringer. Blant de navigasjonssystemene som er benyttet er VectorNAV, Lidar og kamera. Målet er å utvikle en autonom ATV som kan benytte sensordata fra disse systemene, men dette er foreløpig kun testet ut i Gazebo-simulatoren.

Hovedoppgaven i dette prosjektet er å utvikle et system for automatisk giring av ATV-en, noe som er essensielt for å kunne styre den fullstendig via fjernkontroll eller benytte den autonomt. Dette skal kunne gjøres ved hjelp av programvare gruppen skal utvikle og koble sammen med det eksisterende systemet som ligger på en datamaskin navngitt Processing Unit (PU) som er montert på siden av ATV-en. En lineær aktuator er allerede kjøpt inn

for å implementere og manipulere selve girstaget til ATV-en. En utfordring med giringen på ATV-en er at det til tider kan være vanskelig å bytte gir uten å gynge frem og tilbake på ATV-en. Det kan dermed være nødvendig å utvikle et system som kan sørge for noe bevegelse i girkassen for å løse dette problemet. Systemet bør også være robust og værfast, og burde kunne implementeres uten at det gjøres irreversible endringer på de originale ATV-deler.

I tillegg til hovedoppgaven er det satt opp flere tilleggsoppgaver, som kan utføres om det skulle bli tid til overs. Disse inkluderer et system for fjernstyrt start av ATV-en, tuning av bremseaktuatoren, forbedring av kraftoverføringen til styrstammen og montering av et kamera for objektdeteksjon.

1.3 Veiledning for prosjektet

I forbindelse med bachelorprosjektet er det flere veiledere og sensorer. Fra universitetet er det en intern sensor og en veileder. Den interne veilederens rolle går ut på å gi generell veiledning og ha en oversikt over statusen på prosjektet. Gruppen skal selv ha kontroll over prosjektstyringen og internveileders rolle skal derfor kun være å støtte under prosessen.

Som eksterne veiledere stiller KDA med 3 tekniske veiledere med kompetanse innenfor forskjellige fagfelt. Disse vil være til gruppens disposisjon når gruppen jobber ATV'en på KDA sitt lokale og bistå dersom det skulle dukke opp spørsmål relatert til ATV'en underveis i arbeidet. Til slutt er det intern sensor fra universitetet og en ekstern sensor som sammen med internveileder vil bli enige om en individuell karakter for hver prosjektdeltaker.

1.4 Prosjektverktøy

Gruppen benytter en rekke verktøy i form av kommunikasjon, fildeling og teknisk utvikling for å oppnå effektivt samarbeid for prosjektet. En oversikt over hvilke verktøy som brukes beskrives i underkapitlene.

1.4.1 Teamup

Ved oppstart av bacheloroppgaven ble gruppen enige om å koordinere kjernetid for arbeidet, samt håndtere eventuelle hindringer ved hjelp av en felles kalender. Teamup

ble fort adoptert av gruppen, og er et nyttig verktøy som lar gruppen dele en felles kalender. Verktøyet brukes som en oversikt for start og stopp av sprinter, forelesninger, eksamensdatoer, jobbintervjuer mm.

Iht. gruppekontrakten skal fravær meldes fra til gruppen ved første anledning, se vedlegg A1. Teamup gir en oversikt over gruppemedlemmenes personlige ansvarsområder som jobb og familie som hindrer arbeid i kjernetid. Denne oversikten forenkler gruppens prosess for planlegging av møter utenfor de bestemte tidsrammene for prosjektet.

1.4.2 Clockify

Ved oppstart av bacheloroppgaven oppsto et behov for timelister. Timer ble manuelt innskrevet i en mal Microsoft Excel, men det var lite effektivt. I sammenheng med første veiledermøte med intern veileder, Joakim Bjørk, ble Clockify anbefalt. Clockify har mengder av funksjoner og metoder for sortering av prosjekter, som ledet til diskusjoner blant gruppemedlemmene. Som et tidsbesparende tiltak ble et medlem utnevnt til å bestemme oppsettet i Clockify.

Clockify er et verktøy som lar gruppen føre timer manuelt og i form av tidtakere for et prosjekt. Ved bruk av beskrivelsesfelt, prosjekt, oppgaver, og tagger kan gruppen effektivt produsere mengder med relevante rapporter som oppsummerer arbeidet for prosjektet. Clockify har spart gruppen mye tid, og økt effektiviteten etter mer bruk.

1.4.3 Microsoft Excel

Microsoft Excel er et verktøy med mange innebygde funksjoner som gir gode og oversiktlige representasjoner og illustrasjoner av data. Verktøyet brukes til enkle utregninger i form av timer, opptelling av stemmer ved valg, utforming av grafer mm.

1.4.4 Facebook Messenger

Ved oppstart av bacheloroppgaven var effektiv kommunikasjon nødvendig for kommunikasjon med alle gruppemedlemmer, gruppe-chat i Facebook Messenger ble opprettet for dette formålet. Et krav om å være aktive på gruppe-chat på Facebook Messenger ble en del av første versjon av gruppekontrakten.

1.4.5 Microsoft Teams

Gruppen består av studenter med erfaring fra Covid-19 pandemien, mange studenter har derfor erkjent behovet for en digital plattform. Ved etablering av bachelorgruppen, ble kommunikasjon og fildeling opprettet i Microsoft Teams. Microsoft Teams synkroniserer delte filer med Microsoft OneDrive, som tillater for effektiv fildeling integrert i Windows Filutforsker, samt tillater det for digitale møter og meldingsfunksjonalitet. Fildeling er ikke uten feil, forsinkelser i synkronisering kan redusere effektivitet i arbeid, men gruppen har fått god nytte av verktøyet.

1.4.6 Draw.io

Ved utvikling av systemer kan det være fordelaktig å benytte verktøy som muliggjør felles redigering. Et enkelt verktøy som tillater gruppen å visualisere arbeidsoppgavene enkelt og oversiktlig. Verktøyet benyttes for å tegne tankekart, produsere illustrasjoner og diagrammer og gir gruppen en visuell forståelse under diskusjoner om spesifikke undersystemer i prosjektet.

1.4.7 Azure DevOps

Arbeidsgiver benytter Azure DevOps for prosjekter i samarbeid med studenter. Gruppen ble tildelt brukere knyttet til Azure DevOps prosjektet Lone Wolf. Azure DevOps brukes til git repository, wiki, sprint oversikt, og lagring av tidligere dokumentasjon. Dette er elementer som blir nyttige for overlevering av sluttprodukt til arbeidsgiver, kommende studenter vil få tilgang til alle prosjektfiler relevante for utviklingen, og arbeidsgiver kan undersøke endelig sluttrapport for konklusjoner av utført arbeid.

1.4.8 Qt Creator

Qt Creator er et program for utvikling av grafiske brukergrensesnitt på ulike plattformer, inkludert Windows, Linux, MacOS, Android og iOS. Det tilbyr en intuitiv plattform med verktøy for å lage moderne og interaktive grafiske brukergrensesnitt. Qt Creator har en visuell designer for rask oppretting av brukergrensesnitt, og en kraftig kodeeditor med funksjoner som syntaksutheving og automatisk utfylling. Den støtter utvikling på tvers

av plattformer, slik at utviklere kan skrive kode én gang og distribuere den på forskjellige operativsystemer.

1.4.9 SolidWorks

SolidWorks er 3D-modelleringsverktøyet gruppen bruker for å designe de fysiske delene som skal produseres. SolidWorks brukes også som et simuleringsverktøy, i form av FEM[1] analyse, for å foreta forskjellige styrkeberegninger på delene som blir tegnet i programmet.

1.4.10 Git

Git er et åpen kildekode versjonskontrollsystem[1] som er utbredt i utviklingsbransjen. Dette systemet benyttes for samarbeid på kodebasen mellom flere personer, og gir en komplett logg over endringer som er gjort på et prosjekt fra start. Git muliggjør også deling av kode, splitting av kodebaser i grener og andre viktige funksjoner. Ved å benytte Git, er det mulig å registrere endringer, trekke tilbake endringer, synkronisere flere lokale instanser av kodebasen, og ha en backup av koden dersom prosjektet skulle gå tapt på din lokale maskin. Betegnelsen for en kodebase som er satt inn i Git-systemet, er ”repository”. Gruppen benytter både Azure DevOps og GitHub som vert for våre kodebaser, og har dermed sikret en god oppbevaring og styring av vårt kodearbeid.

1.4.11 Github

Github er en skybasert tjeneste som tilbyr Git-verter for opprettelse og opplasting av Git-repositorier. Github muliggjør enkel og effektiv bruk av Git, uten å måtte opprette en lokal vert. Denne tjenesten gir dermed en enkel tilgang til kollaborativt arbeid på kodebasene. Betegnelsen for en kodebase inne i Git-systemet er ”repository”. Gruppen benytter Github som vår Git-vert for deler av prosjektet som ikke er flettet inn i hovedrepositoryet på Azure DevOps. Dette gir oss en fleksibel og praktisk måte å håndtere og dele kode på, samtidig som det sikrer en god styring og oppbevaring av våre kodebasene.

[1] Finite Element Method

1.4.12 PlatformIO

PlatformIO er et gratis arbeidsområdemiljø for embedded programmering, som tilbyr støtte for et bredt spekter av programmeringsspråk, inkludert C++, og plattformer, inkludert Arduino. PlatformIO har også en innebygd terminal for seriell kommunikasjon. Gruppen benytter PlatformIO for programmering og debugging av Arduino i C++, i kombinasjon med Visual Studio Code. Dette valget er basert på vår vurdering av at PlatformIO og Visual Studio Code gir et mer effektivt og funksjonsrikt arbeidsmiljø enn ArduinoIDE.

1.4.13 Visual Studio Code

Visual Studio Code (også kjent som VSCode) er en gratis, delvis åpen-kildekode, web-app-basert IDE[2] utviklet av Microsoft. Denne IDE'en er svært populær fordi den tillater installasjon av gratis, modulære utvidelser som gjør det mulig å arbeide med en enorm rekke programmeringsspråk og utviklermiljø, og kan personaliseres etter brukerens ønsker. Utvidelsene er laget både av brukere, selskaper og Microsoft selv.

Det er ikke blitt etablert grupperegler for bruk av IDE, men VSCode er en IDE flere av oss bruker for å skrive, kompilere og debugge kode. Gruppens medlemmer for lov til å velge IDE etter personlig preferanse.

1.4.14 LTSpice

LTSpice er et gratis CAD-verktøy for design, simulering og anaylse av elektriske kretser, utviklet av Analog Devices (tidligere kjent som Linear Technology), og er en videreutvikling/variant av SPICE[3]. LTSpice, som andre SPICE-type verktøy, lar deg koble opp kretser digitalt, og simulere og analysere de. LTSpice lar deg også eksportere kretsene dine som moduler (kalt "subcircuits") som kan brukes i andre kretser og er lagret i et standard SPICE råtekstformat. LTSpice har et enormt utvalg av ferdiglagde moduler og komponenter, både laget av brukere og av produsenter, og moduler fra andre SPICE-verktøy er lagret med en identisk eller nært-beslektet syntaks som ofte kan gjøres

[2] "Integrated Development Environment", et program som er laget for å redigere tekst, med en innebygd kompilator, debugger og andre funksjoner som er nyttige for å utvikle programvare.

[3] "Simulation Program with Integrated Circuit Emphasis", et kretssimulatorprogram

kompatibel med LTSpice med noen enkle modifikasjoner. Bruker- og produsertlagde moduler og komponenter kan lett lastes ned og importeres som råtekstfiler. Selv om LTSpice er gratis har den ingen funksjoner som er låst bak en betalingsmur, og er derfor det mest utbredte og brukte SPICE-verktøyet i industrien per dags dato.

Gruppen tar i bruk LTSpice som vårt primære hjelpemiddel for utvikling, tegning, simulering og analyse av våre elektronikksystemer.

1.4.15 EasyEDA

EasyEDA er et gratis CAD-verktøy for design av kretser og kretskortlayout, laget primært for brukervennlighet. Kretskort kan eksporteres som Gerber-filer som er kompatible med de fleste kretskortprintere. Programmet har også mulighet for kretssimulasjon og "design-rule checking" (DRC). Det støtter multi-layer PCB design, og har et bibliotek med produsent- og brukerlagde komponenter.

Gruppen bruker EasyEDA for å designe kretskort, kontrollere våre design med DRC og deretter eksportere de som Gerber-filer. EasyEDA brukes ikke for kretssimulasjon, ettersom LTSpice er bedre egnet.

1.4.16 Matlab

Matlab er et godseid programmeringsspråk, IDE og utviklermiljø, utviklet av MathWorks. Matlab har en rekke verktøy i sitt bibliotek som kan lastes ned som utvidelser, som blant annet System Identification Toolbox, Signal Processing Toolbox og Simulink. Matlab og mange av dens utvidelser krever en betalt lisens for å kunne brukes.

Gruppen kan ta i bruk Matlab med studentlisens utdelt av USN, for å gjøre beregninger og analyser på kontrollsystemet vårt, og for graftegning. Bruk av System Identification Toolbox sin funksjon "tfest" for estimasjon av transferfunksjon, og Control System Toolbox sin "pidtune"-funksjon for instilling av PID-parametere er verdt å nevne.

1.4.17 WSL II Ubuntu

WSL (Windows Subsystem for Linux)

WSL II Ubuntu er en plattform som brukes av mange utviklere for å utvikle kode som

skal kjøres på Windows-maskiner, uten bruk av virtuell maskin eller dual-booting. Denne plattformen muliggjør enkel testing og validering av programvare, ved å sikre kompatibilitet mellom operativsystemet på maskinen og utviklingsmiljøet på Windows.

WSL II Ubuntu gir en effektiv måte å simulere og teste programvare i et lignende miljø som på PU. Ved å benytte seg av denne plattformen, kan gruppens utviklere unngå vanlige problemer med manglende kompatibilitet mellom software og PU, og dermed sikre en mer effektiv utviklingsprosess.

1.4.18 VMware Workstation 17 Player

VMware Workstation 17 Player er et verktøy som gjør det mulig å opprette og kjøre virtuelle maskiner på lokal datamaskin. Ved å laste ned en .iso fil fra en operativsystem distributør er det mulig å opprette virtuelle maskiner med ulikt operativsystem ulikt det som er installert på lokal datamaskin. Ressursene som CPU, RAM, nettverkskort, diskplass osv. blir delt fra den lokale maskinen med den virtuelle maskinen.

Ettersom Ubuntu 20.04 LTS er installert på Processing Unit (PU), datamaskinen montert på Lone Wolf ATV, kan utviklerne programmere, kompilere og teste kode i et operativsystem tilsvarende det som er installert på PU.

1.4.19 7-Zip File Manager

Verktøyet 7-Zip File Manager er et program som kan komprimere filer og mapper til .zip filer. Verktøyet har blitt benyttet i sammenheng med sikkerhetskopier av prosjektfilene gjennom prosjektet.

Alle gruppemedlemmer har blitt anbefalt og har hatt et felles ansvar for å ta jevne sikkerhetskopier, slik at ikke all dokumentasjonen forsvinner ved en uforventet feil.

1.4.20 Doxygen

Doxygen er et verktøy som går gjennom kildekoden som er utviklet gjennom et prosjekt, og genererer en rapport som beskriver programmet. Den tar for seg alt fra klasser, funksjoner, macros, inkluderte pakker, osv. Ved bruk av programmet Doxywizard har gruppen produsert doxygen PDF rapporter som vedlegges til dokumentasjonen.

2 Scrum

Kapittelet beskriver gruppens valg, bruk og implementasjon av Scrum.

2.1 Valg av prosjektmodell

Valg av prosjektmodell ble holdt etter individuell informasjonsinnhenting for de ulike prosjektmodellene. Gruppen gikk deretter i en felles diskusjon hvor det ble tydeliggjort at en Agil-prosjektmodell var foretrukket, og at modellen måtte ha en iterativ utviklingsprosess. Gruppen gjennomførte en avstemning hvor hvert medlem kunne prioriterte ønsket prosjektmodell med en verdi fra én til fem, uten gjentagende verdier.

Prosjektstyringsmodeller	Joachim	Leif Arne	Vegard	Martin	Sigurd	Vebjørn	Sum
Scrum	4	4	3	3	0	3	17
Kanban	5	0	1	0	4	0	10
Scrumban	0	5	2	5	5	0	17
Spiral	0	3	4	2	1	5	15
Unified Process	2	0	5	4	3	2	16
Agile	1	2	0	0	2	4	9
RUP (uni.proc.)	0	0	0	1	0	0	1
V model	3	1	0	0	0	1	5
T model	0	0	0	0	0	0	0
CAFCR / CAFCR +	0	0	0	0	0	0	0

Figur 2.1: Resultat av stemming på prosjektmodell

Etter grundige vurderinger og en avstemming, besluttet gruppen å implementere Scrum-prosjektmodellen for prosjektet, se figur 2.1. To av medlemmene uttrykte mistillit til Scrumban-modellen, mens kun et medlem viste mistillit til Scrum. Videre støttet anbefalingene fra KDA Division Land Systems bruken av Scrum-modellen, da denne har vist seg å være en god modell for nyskapning. Det er også verdt å merke seg at Scrumban-modellen er en blanding av Scrum og Kanban, der Kanban er bedre egnet for vedlikehold av eksisterende systemer, mens Scrum er bedre egnet for nyskapning. Gruppen kom til slutt til enighet om at Scrum som er en agil og iterativ prosjektmodell med fokus på nyutvikling var velegnet for prosjektet.

2.2 Scrum Events

Scrum består av fem Events: Sprint Planning, Daily Scrums, Sprint Review, Sprint Retrospective, og Sprintene, sistnevnt diskuteres i kapittel 9.1.

2.2.1 Sprint Planning

Ved oppstart av ny sprint kommer gruppen til enighet om hva som skal oppnås med sprinten. The Scrum Guide beskriver tre spørsmål [2, s. 8] som er med på å definere sprinten:

1. Hvorfor er sprinten verdifull?
2. Hva kan gjennomføres denne sprinten?
3. Hvordan skal gruppen oppnå målet?

Sprint Planning gjøres i samarbeid mellom utviklerne og Product Owner, der det skal diskuteres hvilke oppgaver fra Product Backlog som skal være med i Sprint Backlog. Product Owner har ansvar for å prioritere oppgavene. Resultatet av Sprint Planning er definisjonen av Sprint Goal.

2.2.2 Daily Scrum

Daily Scrum gjennomføres iht. beskrivelsen i The Scrum Guide [2, s. 9] og holdes hver morgen kl. 08:10-08:25. Daily Scrum holdes med hensikten å ta i bruk de fem verdiene til Scrum: forpliktelse, fokus, åpenhet, respekt og mot [2, s. 4].

Daily Scrum ble først utført som kortvarige stående morgenmøter, dette gjør det lettere for utviklerne å holde fokus, samt er det med på å øke produktiviteten fra starten av dagen. Møtene holdes til samme tid hver dag, som gir en forpliktelse til å møte på jobb for å være med i teamet. Møtet har som formål å informere om arbeidsoppgavene til de enkelte, samt å avdekke hva som hindrer dem fra å fullføre oppgavene. Dette er med på å forbedre åpenheten i gruppen, der det er forventet å ta hensyn til hindringene og dermed respektere hverandre ved å prioritere oppgaver som blokkerer andres progresjon. Det kreves også mot å informere om hvordan andre utviklere er til hinder for dem, som betyr at alle utviklere må respektere det som blir tatt opp.

Gruppen har tatt hensyn til forsentkomminger i form av utsetting av Daily Scrum når medlemmer ikke er tilstede, eller ved å gi en oppsummering på Daily Scrum til medlemmer som er forsinket.

2.2.3 Sprint Review

Hver sprint avsluttes ved hjelp av to Scrum Events: Sprint Review og Sprint Retrospective, hvor sistnevnt konkluderer sprinten. Scrum Review utføres iht. The Scrum Guide [2], og som navnet gir en indikasjon på har Sprint Review som hensikt å gi en gjennomgang for resultatet av sprinten. Sprint Review skal hjelpe å presentere resultater til arbeidsgiveren, dette har ført til en endring slik at slutten på hver sprint sammenfaller med progresjonsmøte med arbeidsgiver.

The Scrum Guide anbefaler 4 timers Timebox for Sprint Review for en 4-ukers sprint [2, s. 9], gruppen begrenser seg dermed til en Timebox på 2 timer for 2-ukers sprinter.

Sprint Review gjennomføres i to deler, hvor første del har en Timebox på 1 time som brukes til gjennomgang av oppnådde mål, og blir samtidig formulert til en presentasjon til arbeidsgiver i del 2. Del 2 gjennomføres med arbeidsgiver i form av presentasjon og diskusjon under progresjonsmøtene.

2.2.4 Sprint Retrospective

Første Sprint Retrospective ble gjennomført siste dag i Sprint 3, les nærmere om implementasjonen av Sprint Retrospective siste paragraf i kapittel 9.1.4. Slutten av Sprint 4 var med på å definere implementasjonen av Scrum Retrospective. Hvert medlem får en skriftlig oppgave å skrive et paragraf hvor det skal komme frem hva som ble fullført, hvilke hindringer som oppstod, hvordan hindringene ble løst, og hva medlemmene lærte gjennom sprinten.

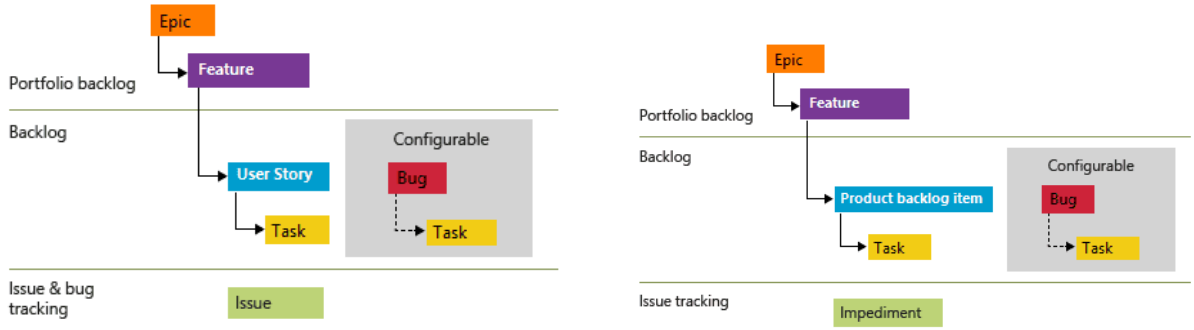
Som nevnt i kapittel 2.2.3, konkluderer Sprint Retrospective sprinten, og er et hjelpemiddel med hensikt å øke effektiviteten og kvaliteten av gjennomføringen av arbeidsoppgavene, samt å gi innsikt til gruppemedlemmene for progresjon, hindringer og lærdom.

2.3 Prosjektstyringsverktøy - Azure DevOps

Under tilbakemeldingene ved 1. presentasjon ble det klart at oppdragsgiver ønsker at det legges til rette for et transparent progresjonssystem som lett kan observeres eksternt (digitalt). KDA bruker stort sett Jira og Confluense som Scrum verktøy, men har ved tidligere Lone Wolf prosjekter benyttet det digitale verkøyet Azure DevOps. Det er derfor

ønskelig at bachelorgruppen benytter samme verktøy. Grunnet generell høy IT-sikkerhet i KDA, tok det noe tid etter 1. presentasjon før gruppen fikk utlevert innloggingsinformasjon til DevOps og fra og med 10. februar var prosjektet LoneWolf i organisasjonen kdastudents klar til bruk og utfylling for bachelorgruppen.

Azure DevOps har mulighet for å velge forskjellige oppsett relatert til ulike prosjektstyringsmodeller. Alle de tidligere Lone Wolf prosjektene har benyttet oppsettet for Agile som fint kan brukes til Scrum. DevOps har også alternativet som heter Scrum, men Agile ble likevel benyttet siden det var i bruk fra før. Det er veldig liten forskjell på bruken av det i Devops og i alle praktiske henseender er hovedforskjellen at **Product backlog item** istedet kalles **User Story** [figur: 2.2].



(a) Agile in Azure DevOps **(b)** Scrum in Azure DevOps

Figur 2.2: Agile vs Scrum in Azure DevOps

Epic: I Scrum sammenheng er Epic en stor oppgave som umulig kan løses i løpet av én sprint og ønskes derfor delt opp i mindre håndterbare deler. Underliggende funksjonalitet og brukerhistorie det skal itereres over vises i Sprint Backlog når det legges til relaterte oppgaver (Tasks) som skal arbeides med i den aktuelle sprinten.

Feature: En Epic har én eller flere funsjonaliteter

User Story: Også kalt **Product backlog item** benyttes for å dele opp systemets funksjonalitet i mindre og mer konkrete bruksområder.

Task: For å kunne utvikle et produkt som dekker behovet i en **User Story** kreves det gjerne at en rekke oppgaver er fullført. Oppgaver bør derfor være utformet slik at de er spesifikke og håndterbare nok til å fullføres i løpet av en sprint. En oppgave anses som ferdig når den er testet og godjent sett fra status på systemet ved endt sprint. Ved senere iterasjoner av systemet vil ny oppgave med ny ID opprettes, da

med andre premisser enn tilsvarende fullført oppgave i en tidligere sprint.

Issue/Bug: Utfordringer og feil som dukker opp under utvikling eller testing som må utbedres legges inn som Issues eller Bugs.

2.4 Scrum Roller

Prosjektmodellen Scrum beskriver et Scrum Team bestående av Product Owner, Scrum Master og utviklere. Rollene beskriver ansvarsområder innenfor styring av prosjektet og beskrives i underkapitlene. I dette prosjektet fungerer Product Owner og Scrum Master som utviklere der det er hensiktsmessig.

Etter tilbakemelding fra intern sensor, Karoline Moholth, kom gruppen til enighet om at det mest effektive er å implementere Scrum med en fast Scrum Master. Gruppen holdt avstemning og utnevnte Joachim Jamtvedt Børresen som Product Owner og Martin Jørgensen som Scrum Master.

2.4.1 Product Owner og Backlog

Ansvarlig Product Owner: Joachim Jamtvedt Børresen

Product Owner er ansvarlig for å maksimere verdien av produktet som et resultat av det samlede arbeidet i Scrum gruppen og gjennomføres iht. beskrivelsen i The Scrum Guide [2, s. 5]. Hovedoppgaven er å opprette og vedlikeholde en transparent og forståelig Product Backlog, samt legge til og fjerne oppgaver i denne. Det er svært viktig at Product Owner forstår behovene til oppdragsgiver (stakeholders) for å kunne utforme og prioritere oppgaver og kommunikasjon er viktig i så henseendet.

Som et ledd i anvarsområdet ble det lagt inn Epics, Features, User Stories og tasks i Product Backlog. Disse samt nye elementer kan endres fortløpende av Product Owner ved behov. Med utgangspunkt i Sprint Retrospektive analyse på sprintens avsluttede dag, bestemmes det hvilke oppgaver som må endres til senere sprinter dersom de ikke blir ferdigstilt iht. plan.

Utdrag av Product og Sprint Backlog i Azure Devops kan sees i [figur: 2.3 og 2.4] nedenfor. Oppgaver relatert til User Story; **Som fjernstyrt ATV operatør bør jeg kunne skifte gir** (ID 1662), er her vist i Product Backlog [figur: 2.3]. Se at oppgavene som er planlagt

utført for sprint 4 [figur: 2.4] kun er et utvalg av oppgavene som må utføres totalt sett for ID 1662. Samme figur viser alle oppgaver planlagt for Sprint 4 på tvers av alle User Stories og Features.

Order	Work Item Type	Title	State	Effort	Busin...	Value Area	Tags
1	Epic	Dokumentasjon og presentasjon	Active			Business	
2	Epic	Sikkerhet relatert til ATV og personskade	Active			Business	
3	Epic	Fjernstyrt girmekanisme på ATV	Active			Business	
	Feature	Del 1: Skifte gir ved bruk av UI gjennom PU	Active			Business	
	User Story	Jeg som bruker ønsker at batterier på ATV skal vare så len...	New			Architectural	KongsbergHW
	User Story	Jeg som bruker ønsker å kunne bruke ATV utendørs og i ...	New			Architectural	KongsbergHW
	User Story	Som fjernstyrt ATV operatør bør jeg kunne skifte gir	Active			Business	
	Task	Måle gir-punkter	Closed				KongsbergHW KongsbergSW
	Task	Måle step respons aktuator	Closed				KongsbergHW
	Task	Implementere seriellkommunikasjon på Arduino	Active				KongsbergSW
	Task	Utarbeide seriellprotokoll for respons fra arduino til PU	Closed				KongsbergSW
	Task	Undersøke om det er tilstrekkelig rom for aktuator bev...	Active				
	Task	Måle størrelse og dybde på låseklipp-bolt	Active				
	Bug	Mulig defekt motordriver levert av KDA	Resolved			Business	KongsbergHW
	Task	Koble opp og montere motordriver på ATV	New				
	Task	Programmere mikrontroller for styring av motordriver	Closed				KongsbergSW
	Task	Sette opp kommunikasjon med mikrokontroller gjenn...	Active				KongsbergSW
	Task	Koble opp og montere mikrokontroller til ATV	New				KongsbergHW
	Task	Utvikle kontrollsløyfe for styring av aktuator	Closed				KongsbergHW KongsbergSW
	Task	Måling av aktuatorens transferfunksjon	Closed				
	Task	Teste og feilsøke aktuatorstyring uten last	Closed				
	Task	Få kontroll på aktuatorens bevegelse	Active				KongsbergHW KongsbergSW
	Task	Beherske ROS2 funksjonalitet og tilhørende koding	Active				KongsbergSW
	Bug	Aktuator kan ikke styres med nøyaktighet som er ønsk...	Active			Business	KongsbergHW KongsbergSW
	Task	Bestemme plassering av brakett på girstag	Active				
	User Story	Som fjernstyrt ATV operatør bør jeg kunne styre girskift vi...	New			Business	
	Feature	Del 2: Skifte gir selv om girkasse ikke er i posisjon til å flytte g...	Active			Business	
	User Story	Påføre rotasjon på girkasse dersom girskift ikke er mulig	Active			Business	
	Feature	Hindre girskift dersom det kan oppstå potensiell skade på AT...	Active			Business	
4	Epic	ATV systemdata	Active			Business	
5	Epic	Simuleringsmiljø	New			Business	
6	Epic	Forberede autonomi	New			Business	

Figur 2.3: Product Backlog pr 15. mars 2023

Order	Title	State	Assigned To
1	Som fjernstyrt ATV operatør bør jeg kunne skifte gir	Active	Joachim.Jamtv...
	Implementere seriellkommunikasjon på Arduino	Active	Sigurd Sæther...
	Undersøke om det er tilstrekkelig rom for aktuator bevegelse	Active	Vebjorn.Aleksa...
	Måle størrelse og dybde på låseklipp-bolt	Active	Vegard Skårdal...
	Beherske ROS2 funksjonalitet og tilhørende koding	Active	
	Bestemme plassering av brakett på girstag	Active	Vegard Skårdal...
2	Påføre rotasjon på girkasse dersom girskift ikke er mulig	Active	Joachim.Jamtv...
	Til stadighet er det ikke mulig å flytte girspak i ønsket posisjon	Active	Vegard Skårdal...
3	Forhindre skade på girmekanisme	Active	Joachim.Jamtv...
	Tilpasse korrekt kraft fra aktuator til girstag	Active	Sigurd Sæther...
	Beregne og simulere tillatt kraft påført fra aktuator på girstag	Active	Vebjorn.Aleksa...
4	Som bruker er det viktig å vite om farer som kan oppstå og hv...	Active	Martin Jørgens...
	Utarbeide risikovurdering for prosjekt Rev. 1	Closed	Martin Jørgens...
	Utarbeide risikovurdering for personskade, arbeid og bruk av...	Active	Martin Jørgens...
5	Vi som bachelorgruppe ønsker å dokumentere utført arbeid	Active	Leif.Arne.Bastes...
	Sette opp Latex dokument via overleaf	Closed	Leif.Arne.Bastes...
	Oppsett og 1. utkast til nettside	Closed	Leif.Arne.Bastes...
	Tilstandsdiagram girskift	Active	Martin Jørgens...
	Sette opp Sprint 4 backlog	Closed	Joachim.Jamtv...
	Sette opp Sprint 5 backlog	Closed	Joachim.Jamtv...
	Initielt oppsett av Azure DevOps SCUM prosjekstyring	Closed	Joachim.Jamtv...
6	Vi som bachelorgruppe ønsker å forstå behovet til oppdragsgiv...	Active	Joachim.Jamtv...
	Kravspesifikasjon Rev 3	Active	Vegard Skårdal...

Figur 2.4: Sprint 4 Backlog pr 15. mars 2023

2.4.2 Scrum Master

Scrum Master har som funksjon å hjelpe gruppemedlemmer med implementasjonen av Scrum i henhold til The Scrum Guide [2]. Scrum Master har også som oppgave å øke effektiviteten til gruppen, noen tiltak verdt å nevne er:

- Som et supplement til ordinære Scrum Event timeboxes, ble daglig timeboxing introdusert som et tiltak for å ha skape flere korte tidsfrister. Arbeid mot tidsfrister

er med på å øke effektiviteten, hvor disse tidsrammene kan også bidra til en iterativ arbeidsprosess gjennom arbeidsdagen.

- Gruppen startet Sprint 1 med 15 minutters pauser hver time, inkludert en 45 minutters lunsj pause. Pausene forårsaket støy og førte til tap av fokus én gang i timen, men etter et forslag om å forlenge lunsj pausen til 60 minutter. Pauser på eget initiativ har ført til flere aktive arbeidstimer samt en økning i effektivitet ettersom forstyrrelser forekommer sjeldnere.
- Som nevnt i kapittel 2.3 brukes Azure DevOps. Et forslag om en ny Definition of Done ble lagt frem og godkjent av gruppen, hvor en oppgave på Sprint Backlog ikke skal regnes som fullført før resultatet av oppgaven er dokumentert i dokumentasjonen. Tiltaket er til for å unngå en langvarig prosess til sluttdokumentasjonen.
- Ettersom Daily Scrum er et kort morgenmøte for å få start på dagen ble det lagt sterkt fokus på å begrense møtene til 15 minutters Timebox.

Scrum Master skal hjelpe gruppen med bruk, forståelse og implementasjon av Scrum, og bidrar til forslag for tiltak for økning av effektivitet og kvalitet på arbeid. Scrum Master skal også sørge for gjennomføring av Scrum Events.

2.4.3 Utviklere

Utviklerne består av resterende gruppemedlemmer som bidrar til teknisk utvikling av prosjektet. Hver utvikler styrer og planlegger selv for hva de skal gjøre basert på prioriteringer fra Product Owner og sprintens Sprint Goal. Progresjonen på arbeidet videreformidles under Daily Scrum. Product Owner og Scrum Master opptrer som utviklere under Daily Scrums.

3 Sikkerhetsdokument

Fysisk arbeid på ATV krever gode sikkerhetsrutiner ettersom ATV består av eksponerte bevegelige deler i form av tannhjul, kjede og aktuatorer. Som første delleveranse til arbeidsgiver ble gruppen instruert om å produsere et dokument som beskriver sikkerhetrutiner for håndtering av ATV og dens komponenter. Dokumentet er skrevet, lest og forstått av gruppemedlemmene. Første gjennomgang av oppstartsrutinene ble gjennomført med tilsyn fra ekstern veileder, og en risikoanalyse for arbeid på ATV ble senere inkludert som et tiltak fra risikoansvarlig for å redusere sannsynligheten for skade på personell, materiell, og tapt arbeidstid. Risikoanalysen beskrives av en 3x3 matrise for antatt sannsynlighet og risiko, se Appendiks A11 for oppstartsprosedyre, risikoanalyse og tiltak. Sikkerhetsdokumentet ble godkjent av arbeidsgiver 8. mars 2023.

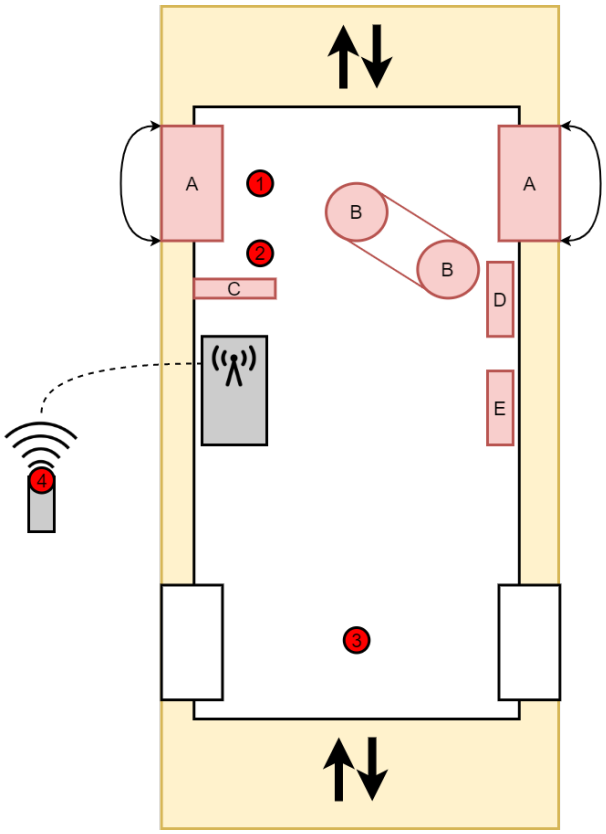
3.1 Nødstoppbrytere og faresoner på ATV

Figur 3.1 beskriver posisjon av nødstopp og faresoner på ATV. Punkt 1 til 3 kartlegger omtrentlig posisjon på nødstoppbryterne montert på ATV. Punkt 4 illustrerer den fjernstyrte nødstoppbryteren som oppbevares i bagasjeluken, bak på ATV.

1. Dødmannssnor
2. Motorstopp
3. Nødstopp bak på hekk
4. Fjernstyrt nødstopp

Punktene A til D i listen under beskriver monterte komponenter på ATV. Disse bevegelige delene er ikke er innkapslet, som gjør det svært viktig å være observante under strømstart, motorstart og arbeid på ATV med system i drift. Det er også viktig at alle i nærheten av ATV er til enhver tid bevisst over hvorvidt strøm eller motor startes før prosedyren gjennomføres.

A. Framhjul
B. Tannhjul for styring av ATV
C. Aktuator for brems



Figur 3.1: Faresoner på ATV

D. Aktuator for akselerasjon

E. Aktuator for giring

Ved oppstart av ATV kalibreres styringsmekanismen. Framhjul svinger fra maksimalt utslag til venstre, deretter til senter. Bremsemekanismen aktiveres uforutsigbart på grunn av støy som fører til en økt risiko for klemfare. Ettersom det er flere områder på ATV som kan påføre personskade, er det viktig at alle tilstedeværende er oppmerksomme og forsikte under det kommer til oppstartsprosedyrene.

Før motor starter skal ATV ha klar bane foran og bak, det gule feltet på figur 3.1 illustrerer rulleretning for ATV. Dersom motor startes i gir, eller ATV settes i gir etter oppstart med inaktive bremser, kan ATV bevege seg, og påkjøring kan forekomme.

3.2 Risikoanalyse for prosjekt

Som et tiltak for å redusere sannsynlighet og konsekvens for risiko relatert til prosjektarbeid, utfører risikoansvarlig en risikoanalyse for prosjektet, se vedlegg A12. Dokumentet

dekker punkter i form av tap av dokumentasjon til livstruende skader. En antagelse for sannsynlighet blir utført ettersom en nøyaktig sannsynlighetsberegning anses som unødvendig for prosjektet.

	Mindre sannsynlig Antatt [0%,1%) sannsynlighet	**Sannsynlig** Antatt [1%,20%) sannsynlighet	**Meget sannsynlig** Antatt [20%,100%] sannsynlighet
Fatal konsekvens Alvorlig skader på person/komponent/prosjekt	Medium	Medium	Høy
Middels konsekvens Mindre skade på person/komponent/prosjekt	Lav	Medium	Medium
Ubetydelig konsekvens Ingen skader på person/komponent/prosjekt	Lav	Lav	Lav

Tabell 3.1: Risikomatrise for risikovurdering

Tiltak **bør vurderes** for lav risiko, **må vurderes** for medium risiko, og **må gjøres** for høy risiko, se tabell 3.1.

Tap av dokumentasjon rammer gruppen, og har en fatal konsekvens for prosjektet, hvor det antas en sannsynlighet i intervallet $[1\%, 20\%)$. Kombinasjonen av *sannsynlig* og *fatal konsekvens* resulterer i medium risiko på risikomatrisen. Hvilket innebærer at tiltak må vurderes. Et forslag om regelmessige sikkerhetskopier på forskjellige medier beskrives, hvor resultatet av tiltaket reduserer konsekvensen av hendelsen fra fatal konsekvens til middels konsekvens. Dersom sikkerhetskopier tas regelmessig og hendelsen forekommer, vil det kun gå tapt progresjon for timer eller dager med arbeid, fremfor totalt tap av all dokumentasjon.

4 System

Kapittelet beskriver systemetvalg, tverrfaglig idemyldring og valg tatt i plenum.

4.1 Valg av løsninger for automatisk giring

Under første arbeidsøkt med ATV ble det først fokusert på hva som var gjort av arbeid på girsystemet av tidligere bachelorgrupper. En lineær aktuator (Linak LA14) var allerede kjøpt inn for å brukes til systemet. Det var også sveiset fast ett festpunkt på det ettermonterte rammeverket som var tiltenkt aktuatoren. I tillegg, var det laget en innfestningskloss som var festet til girstaget for kunne koble aktuatoren på denne. Man kunne enkelt se for seg hvordan systemet var ment å fungere, men det var ikke mulig å finne dokumentasjon på det som var utviklet for girsystemet. En av de første oppgavene for maskiningeniørstudentene ble derfor å sette seg inn i denne mulige løsningen for å verifisere om dette kunne være en løsning som gruppen ønsket å jobbe videre med. Arbeidet med dette blir nærmere beskrevet i avsnitt 5.2.

Videre ble det prøvd å gire manuelt med ATV. Det tok ikke lang tid før problemet med å gire viste seg. Det ble først forsøkt å gire med ATV med bensinmotoren av. Dette viste seg å være problematisk, og girspaken var vanskelig å få i ønsket gir gjentatte ganger. Det ble så diskutert innad i gruppen om det kunne være enklere å gire med bensinmotoren på. Dette viste seg umiddelbart å redusere problemet betraktelig. Problemet var fortsatt tilstede, men opptredde nå sjeldnere og krevde mindre dytting på ATV når det først oppsto. Det ble til slutt konkludert med at problemet fortsatt vil by på utfordringer, men ikke i like stor grad som først antatt.

Parallelt med arbeidet om å få aktuatoren til å kunne velge gir var gruppen gjennom flere idemyldringsøkter der flere løsninger for girproblemet ble vurdert. Gruppen kom i all hovedsak frem til to løsninger på problemet. Den ene løsning innebærer å bruke en mekanisk innretning som fysisk får satt i gang bevegelse i akslingen på ATV. Her har gruppen kommet med flere forskjellige konsepter. Den andre hovedløsningen går ut på å benytte gasspådrag på motoren til ATV for forflytningen.

4.1.1 Mekanisk bein

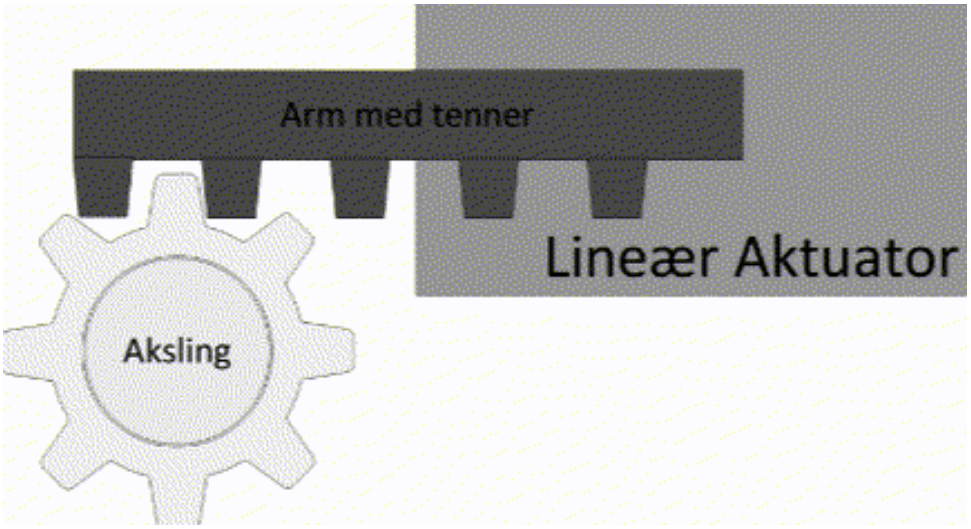
En av de mekaniske løsningene var å ha et komponent som fysisk dytter ned i bakken, for å få bevegelse på ATV. Dette kan gjennomføres ved å bruke en lineær aktuator som dytter ned i bakken på en måte som skaper bevegelse. En annen løsning er å benytte en steppermotor med et bein som dytter ned i bakken og skaper bevegelse ved rotasjon ifra steppermotoren.

4.1.2 Innretning direkte på aksling

Denne løsningen går ut på å dytte direkte på akslingen fremmenfor å dytte ned i bakken. Det er flere måter dette kunne ha blitt løst på.

En løsning er å benytte et tilsvarende komponent som brukes til start av biler. Dette innebærer et stempel med tannhjul som skyves frem til et tannhjul på akslingen som deretter overfører rotasjon til akslingen, for å så trekke seg tilbake.

En annen løsning er å benytte en lineæraktuator med tenner, som skyves inn på et tannhjul på akslingen og skaper rotasjon. Dette er illustrert i Figur 4.1. Ulempen med denne løsningen er at den må bli værende i sin posisjon helt utstrakt etter bevegelsen er skapt, for å ikke ende opp i samme posisjon som ved start. Dette medfører et vist faremoment for å komme i uønsket kontakt med akslingen mens ATV kjører.



Figur 4.1: Lineæraktuator med tenner

4.1.3 Motorstyrt forflytning

Denne løsningen går ut på å benytte gasspådrag på bensinmotoren for å skape forflytning. Hvis brems holdes inne ved girskifte, vil ATV alltid være i stand til å skifte tilbake i giret den skiftet ifra ettersom akslingens posisjon ikke endres. Hvis girskifte ikke er mulig kan dermed ATV kjøre litt fremover for å endre akslingens posisjon, for å deretter prøve girskifte på nytt.

4.1.4 Tester

Det har blitt gjennomført en del tester for å kartlegge hvorvidt det er realistisk å bruke ATV sin bensinmotor for forflytning av ATV. Dette ble gjennomført ved fysisk kjøretest av ATV. Testen gikk ut på å skifte gir med lineæraktuator montert, og gire tilbake og kjøre litt frem hvis girskifte ikke var mulig, for å deretter prøve å gire på nytt.

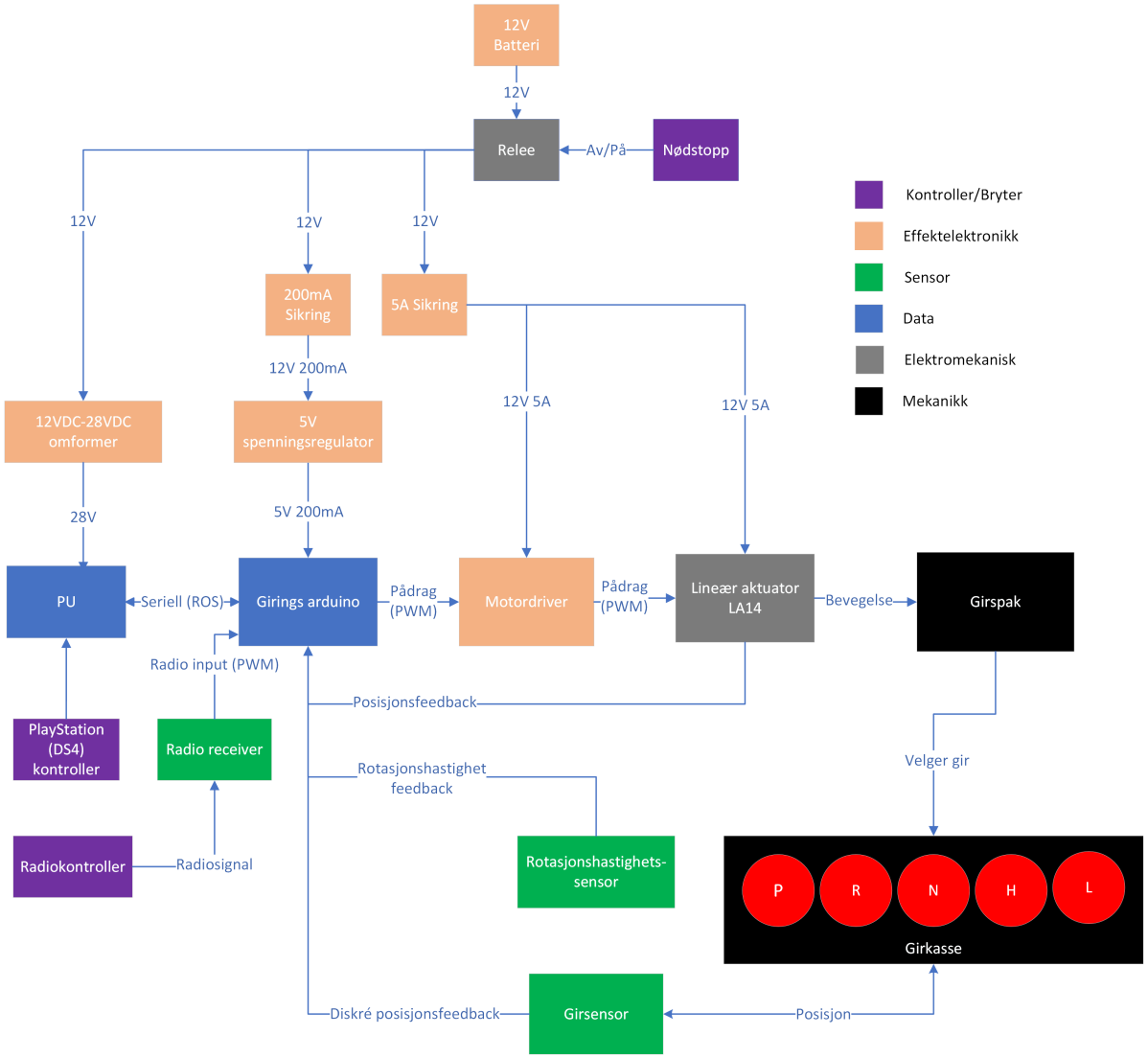
Det ble testet hvorvidt motorstyring kunne benyttes til å manuelt girskift. Testen ga en god indikasjon på at det er mulig å få dette til ved hjelp av programmering.

4.1.5 Konklusjon

Etter å gjennomført testene med motorstyring virket dette til å være en god løsning. Fordelene med en slik løsning er at det ikke er behov for å utvikle en kompleks innretning som må monteres på ATV. Dette ville vært tidskrevende, kostbart og økt ATV'ens kompleksitet ytterligere. Ved motor styring løsningen, trengs det kun å implementere nødvendig programvare på ATV'ens datasystem for å få den til å flytte seg til ønsket posisjon slik at girskifte kan gjennomføres.

4.2 Systemarkitektur

For å få en bedre forståelse av hvordan de forskjellige komponentene i et tenkt girsystem vil samhandle med hverandre, har det blitt laget ett Physical Block Diagram. Dette gir en god oversikt over hvordan de forskjellige komponentene til ATV kan brukes for å oppnå ønsket funksjonalitet. Diagrammet gjorde det også enklere å kartlegge hva som mangler for å få systemet til å fungere som ønsket. Et eksempel på dette er girsensoren. ATV har en egen sensor for å lese av hvilket gir den befinner seg i. Ved å koble seg på dette systemet kan man bekrefte/avkrefte om aktuator har klart å flytte girspak til korrekt gir. Hvordan denne sensoren kan brukes i girsystemet vises i Figur 4.2.



Figur 4.2: Physical Block Diagram av girsystemet

4.3 Utvikling av Krav- og testspesifikasjon

Kravene som settes til systemet er særdeles viktig for å ha klare mål på hva man ønsker av systemet. Det sørger også for at gruppen sammen med arbeidsgiver har en god felles forståelse av hva som er ønskelig av funksjoner i de forskjellige systemene. Oppgaven fra KDA er i utgangspunktet svært åpen med muligheter for egen tolkning. Det har derfor blitt gjennomført et grundig arbeid, med støtte fra KDA sine veiledere, for å lage en krav- og testspesifikajson som dekker ønsket funksjonalitet som skal være mulig å oppnå innefor gitt tidsramme. Se vedlegg A33 for den komplette krav og testspesifikasjonen.

Etter kravspesifikasjonen ble satt opp, ble det satt opp tester som kan verifisere om kravene er oppfylt. Det ble tatt utgangspunkt i at hvert krav kan verifiseres med en eller flere tester, som knyttes opp til kravet ved bruk av en test ID. Første versjon av testspesifikasjonen ble satt opp på dette viset, og er vist i Figur 4.3.

Prosjekt:	1651 Del 1		
Test ID:	T_MEK1	Dato:	03.03.2023
Krav:	MEK1	Resultat:	I.U
Kravklasse:	A		
Kriterium for bestått test:	Lineær aktuator kan bytte gir mellom R, H og L når motor står på.		
Utførsel av testprosedyre:	Testprosedyre: 1. Start opp motor på ATV etter rutiner fra sikkerhetsdokumentet 2. Lineær aktuator styres deretter til de forhåndsinnstilte posisjonene for skifte av de aktuelle girene (R, H, L) frem og tilbake.		
Hvis kriteriet feiler:	Prøv å identifiser hva som hindrer aktuatoren i å flytte girspak til ønsket posisjon.		

Figur 4.3: Eksempel fra testspesifikasjon V1

Etter gode tilbakemeldinger i fra KDA ble det bestemt å gjøre litt endringer på oppsettet. Ulempen med oppsettet slik det er satt opp til nå, er at tillhørende krav ikke er tilgjengelig i testspesifikasjonen, slik at det må letes frem for å få knyttet testen opp til kravet. For å unngå dette, ble det derfor bestemt å samle krav- og testspesifikasjoene til en samlet spesifikasjon. Det ble også bestemt å legge til en egen kolonne der resultatet av testen kan kommenteres etter testen er gjennomført. Resultatet av denne revisjonen kan ses i Figur 4.4.

Prosjekt:	1651		
Krav ID:	MEK1	Dato:	18.04.2023
Kravklasse:	A	Status, krav:	V
Test ID:	T_MEK1	Status, test:	G

Kravspesifikasjon	ATV skal kunne bytte gir mellom R, N, H og L ved bruk av ett mekatronikk system.
Beskrivelse av krav	For at ATV skal kunne kjøre autonomt må den kunne skifte automatisk mellom de forskjellige girene. ATV har girene R (Reverse), L (Low) og H (High) der L brukes for kjøring i røft terreng i lav fart, og H brukes der det er ønskelig med høyere hastigheter.
Kriterium for bestått test:	Lineær aktuator kan bytte gir mellom R, H og L når motor står på.
Utførsel av testprosedyre:	1. Start opp motor på ATV etter rutiner fra sikkerhetsdokumentet 2. Lineær aktuator styres deretter til de forhåndsinnstilte posisjonene for skifte av de aktuelle girene (R, H, L) frem og tilbake.
Hvis kriteriet feiler:	Prøv å identifiser hva som hindrer aktuatoren i å flytte girspak til ønsket posisjon.
Resultat av test:	Denne testen er godkjent, som vil si at lineær aktuatoren greide å skifte imellom de forskjellige girene uten problemer.

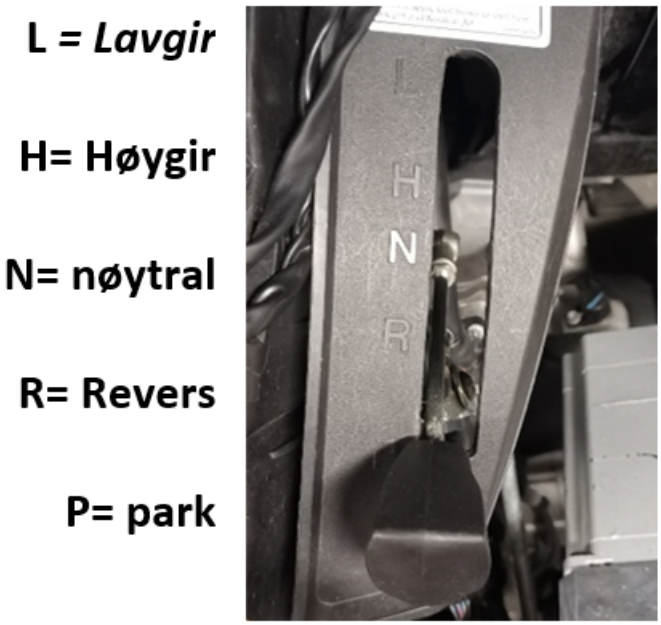
Figur 4.4: Samlet krav- og testspesifikasjon

5 Mekanikk

Hovedoppgaven som er å utvikle et automatisk girsystem er delt inn i 2 deler. Disse blir referert til som «Girsystem del 1» og Girsystem del 2». Del 1 består av å lage ett mekatronisk system som kan skifte gir automatisk ved hjelp av en lineær aktuator, og del 2 omhandler systemet som må utvikles for å sikre at girskift alltid lar seg gjennomføre.

5.1 Tekniske spesifikasjoner ATV

ATV er utstyrt med en CVT girkasse, som er en form for beltedrevet automatgir. Denne er laget slik at girskifte kun kan forekomme uten gasspådrag. Dette er fordi girkassen har en clutch som begynner å gripe når sentrifugalkraften på en roterende disk blir stor nok. Det vil si at clutchen begynner å gripe ved et bestemt turtall motoren. Etter clutch har begynt å gripe burde ikke girskifte gjennomføres, ettersom dette kan medføre skade på interne komponenter i girkasse. Det er 5 forskjellige gir moduser, vist i Figur 5.1.



Figur 5.1: Girspakens posisjoner

5.2 Mekanisk analyse av girsystem del 1

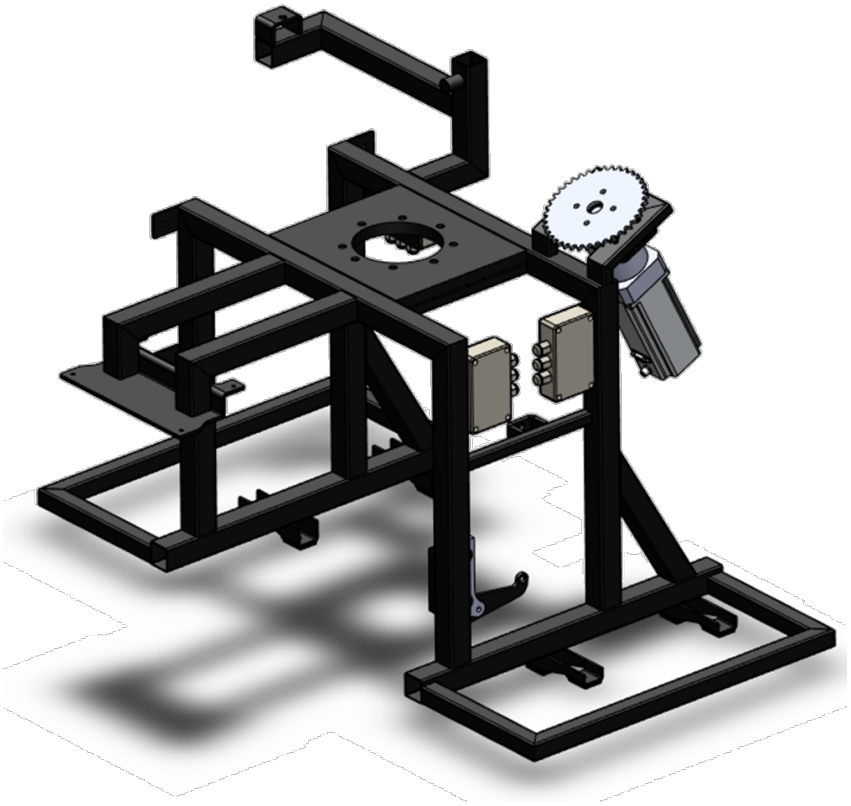
Del 1 består i hovedsak av mekaniske og mekatroniske oppgaver. Når det kommer til det mekaniske så har arbeidsoppgavene bestått av montere aktuator på ATV for så å teste og verifisere at aktuatoren får bevege seg fritt og kan følge girspakens bevegelse effektivt. Siden girspaken nå vil bli brukt på en måte som den ikke er designet for har det også vært et stort fokus på å estimere den maksimale kraften som kan tillates uten å sette de mekaniske komponentene til ATV i fare.

5.2.1 Testing av festepunkter for aktuator

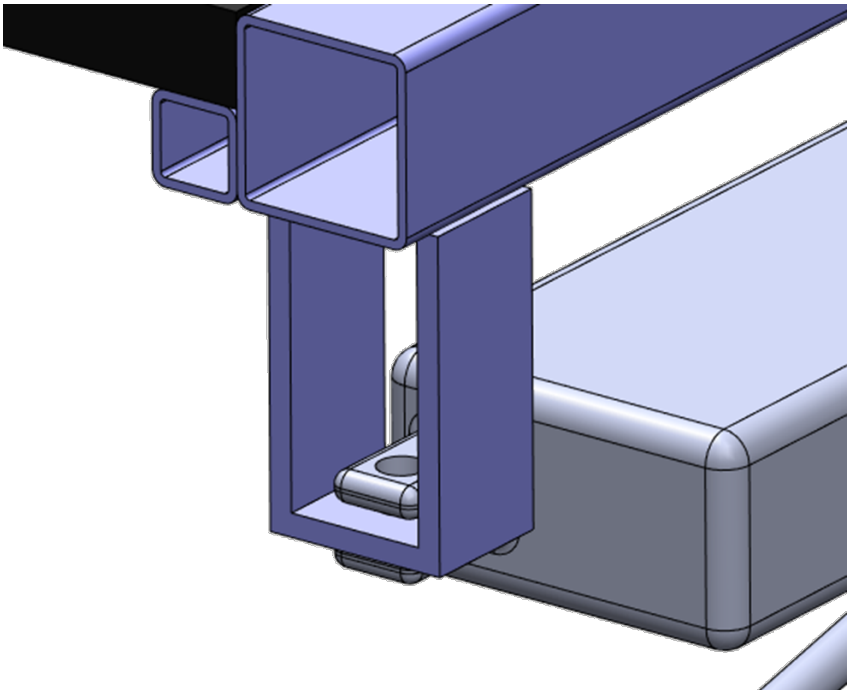
Under første inspeksjon av ATV var fokuset på å bli kjent med den lineære aktuatoren og hvordan denne kan bli implementert på ATV. Tidligere bachelorgrupper har allerede laget festepunkter for aktuatoren. Det ene festepunktet er en plate sveiset på rammeverket Figur 5.3. Det andre feste er en brakett som klemmes over girstaget Figur 5.4. Denne braketten antas å være noe overdimensjonert, men vil kunne fungere bra nok i konseptstadiet. En del av oppgaven er å sette oss inn disse festepunktene for å ta en vurdering om disse kan være hensiktsmessig å gjenbruke. Begge festepunktene ble vurdert til å være gjenbrukbare, så det ble derfor besluttet å gå videre med dette konseptet.

5.2.2 Eksisterende CAD filer

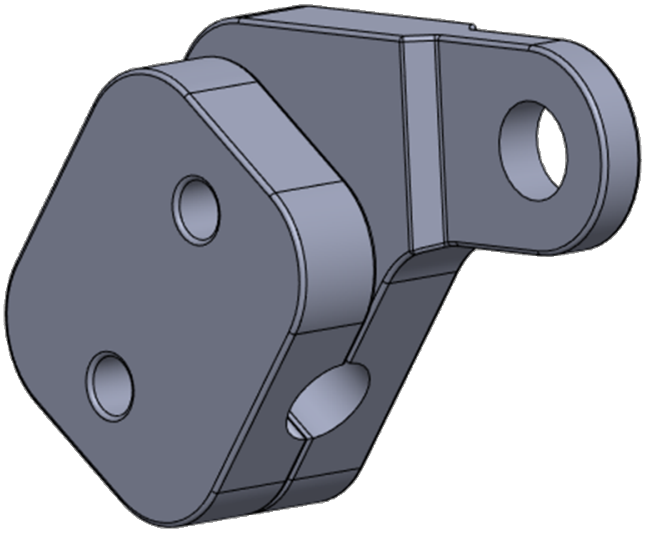
CAD filer for rammeverket som er påmontert ATV'en ble overlevert til gruppen 01.02.2023. Her var det noen ting som ikke samsvarte mellom CAD filer og virkelig ramme. Gruppa har tatt mål av rammeverket og bruker dette til å oppdatere de eksisterende filene. Oppdateringen har gått ut på å legge til en $40\,\text{mm} \times 40\,\text{mm} \times 2\,\text{mm}$ firkant ramme på utsiden av "monteringsplate RWS". Denne firkantrammen blir brukt for brakettfeste til aktuator for girskifte.



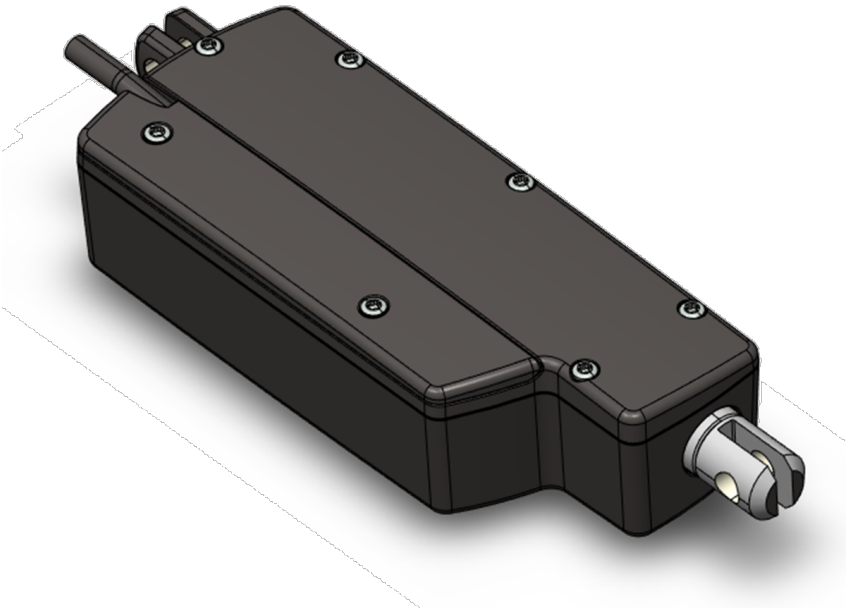
Figur 5.2: Opprinnelig CAD assembly



Figur 5.3: Feste mellom ramme og aktuator

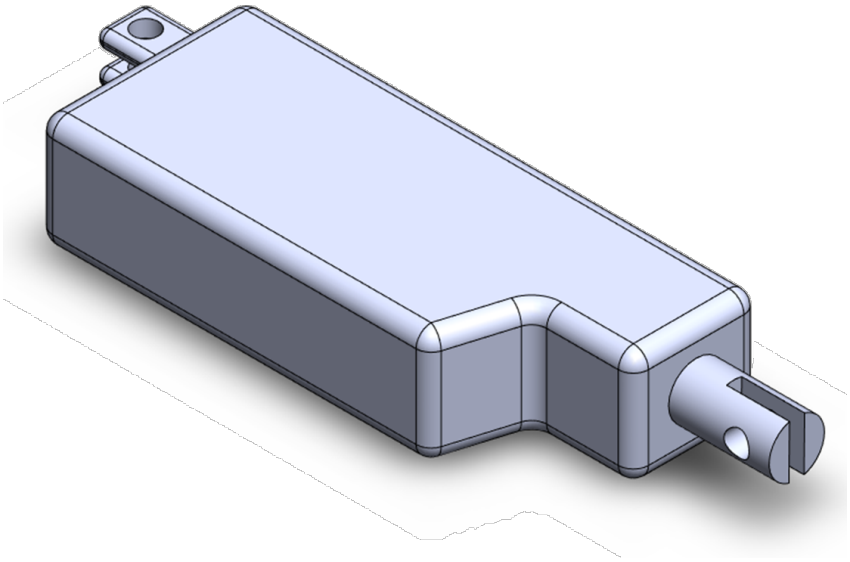


Figur 5.4: Brakett til girstag



Figur 5.5: LA14 lineær aktuator som STP-fil

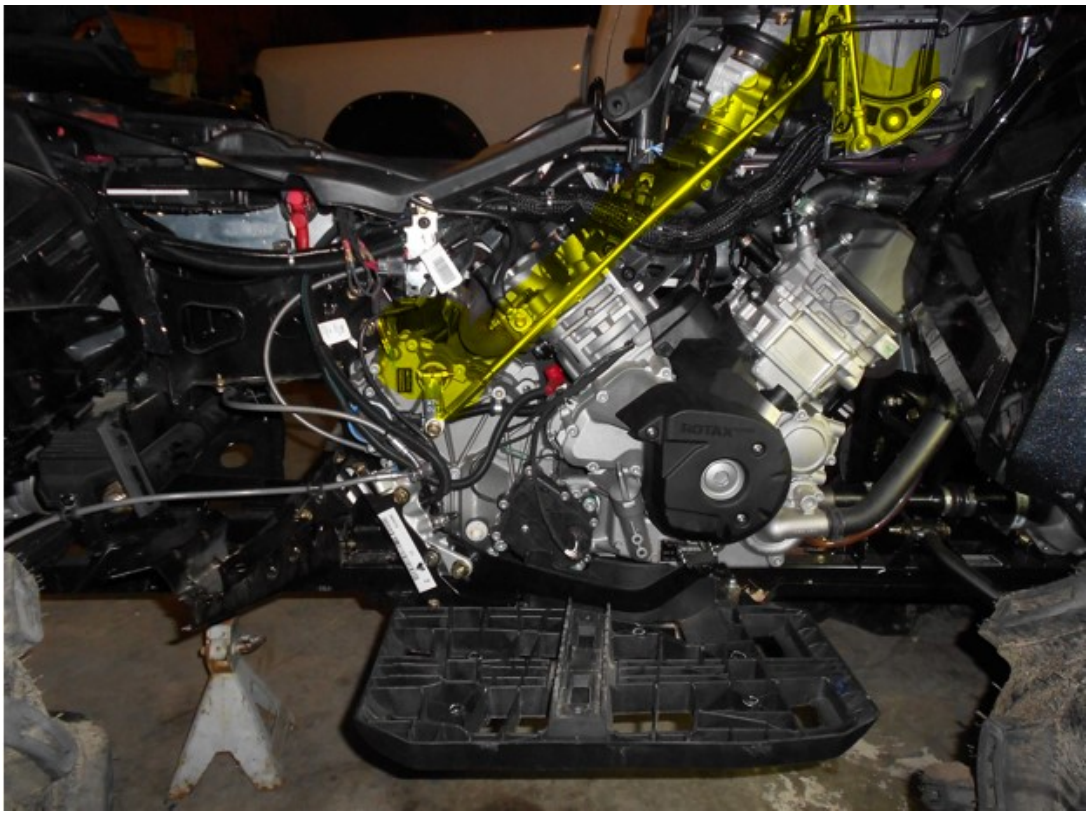
Det ble også kartlagt at festepunktet bakpå aktuator er 90° feil orientert på filen som ble overlevert. Aktuatoren vist i Figur 5.5 er en STP fil og det vil derfor ikke være mulig å flytte på staget. På grunn av dette ble det bestemt å tegne en enkel versjon av denne selv i SolidWorks. Denne modellen vil da kunne bevege på stempelet, og kunne monteres på tiltenk vis til rammeverket.



Figur 5.6: Egenmodellert aktuator

5.2.3 Girstagets funksjon

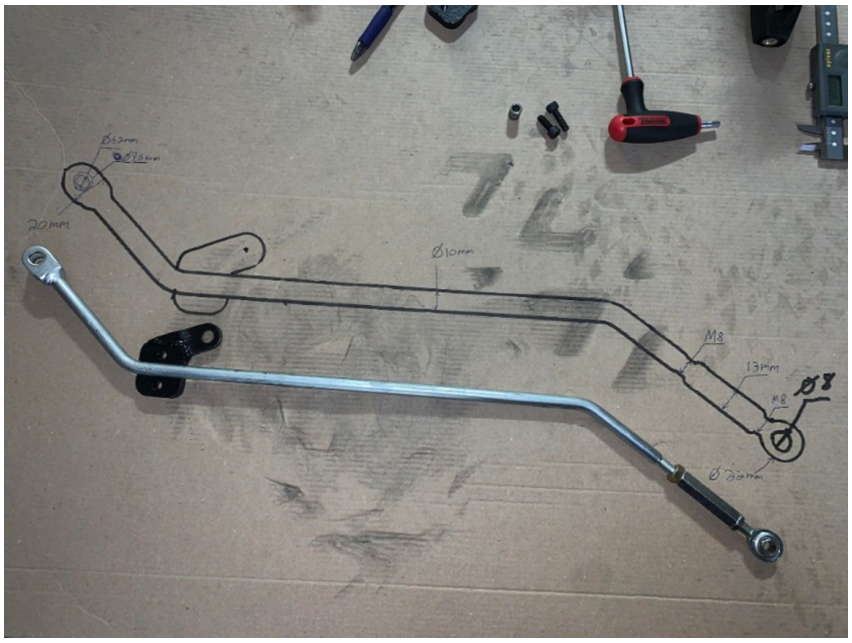
I Figur 5.7 kan man se girstagets posisjon på ATV markert i gult. Girstaget kobler girspaken til en roterende aksling nede ved girkassen til kjøretøyet. Ved å flytte på girspaken frem eller tilbake roterer man denne akslingen. Når akslingen roteres til bestemte posisjoner går girkassen inn i forskjellige gir. Girstaget vist på bildet samsvarer ikke helt nøyaktig med den som gruppen jobber med. Grupppens ATV har en ekstra vinkel som gjør at girstaget kan festet på oversiden av akslingen til girgassen istedet for på undersiden slik som i Figur 5.7. Årsaken til at det blir brukt bilde fra nett er at alt utstyret som er påmontert gruppens ATV tildekker girstaget slik at det ikke lar seg avbildes påmontert.



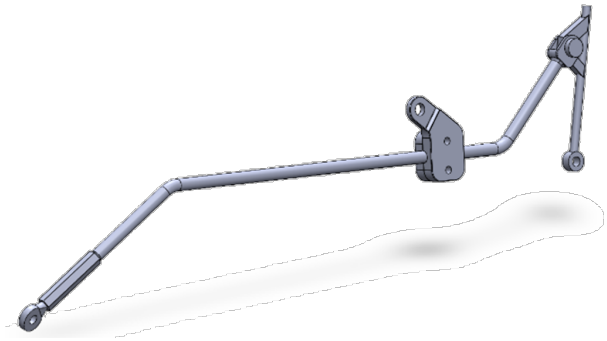
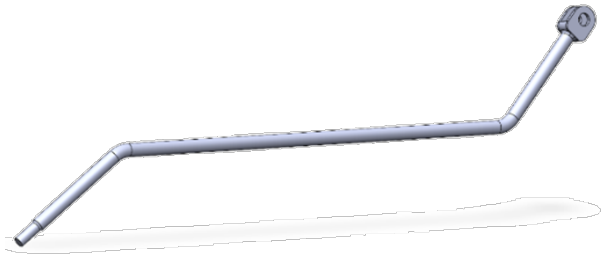
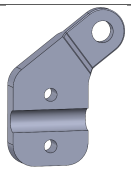
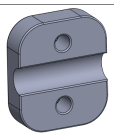
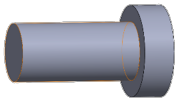
Figur 5.7: Girstagets posisjon på ATV

5.2.4 Girstag sammenstilling

For å enklere kunne ta nøyaktige mål ble girstaget av-montert. Det ble deretter tegnet opp en 1:1 tegning av staget med målsetting, som du kan se på Figur 5.8. Denne informasjonen ble deretter benyttet til å tegne opp en 3D-modell av staget. Girspak ble også tegnet opp og lagt til modellen. I Tabell 5.2 kan du se en total oversikt over alle komponentene i sammenstillingen som skal benyttes i simulasjonen.



Figur 5.8: 1:1 silhuett av girstag skissert på papplate

Komponent	Modell	Materiale
Girstag assy		
Fish eye assy		Plain carbon steel
Girstag		Aluminium 5000- eller 6000-serie (estimat)
Fremside giringsbrakett		Plain carbon steel
Bakside giringsbrakett		Plain carbon steel
Girspak		Plain carbon steel
Girspak bolt		Plain carbon steel

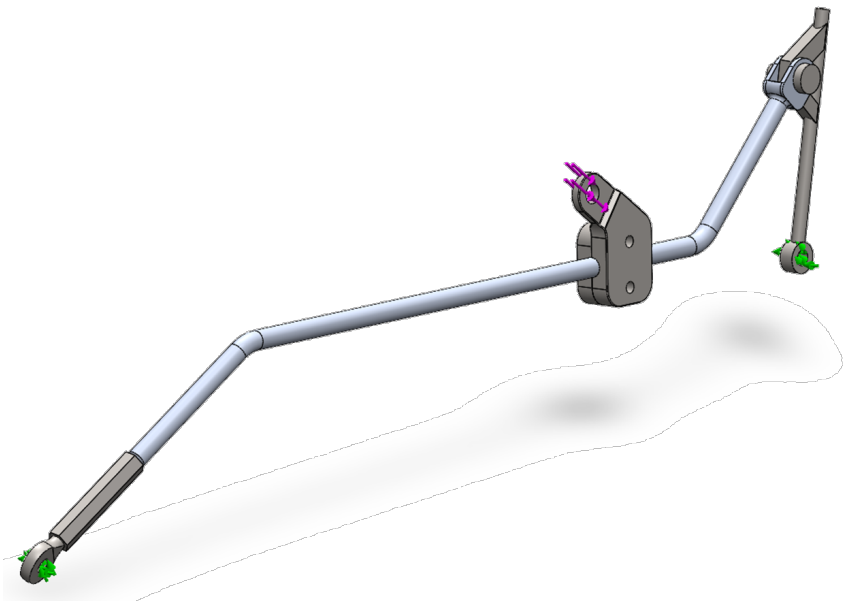
Tabell 5.1: Oversikt over de forskjellige komponentene i girstagets assembly

5.2.5 FEM-analyse av Girstag sammenstilling

Siden det nå vil påføres krefter på girstaget på et annet sted enn det som systemet er opprinnelig designet for, er det viktig å finne ut av hvor stor kraft aktuatoren kan bruke på girstaget uten at den tar skade. Etter at 3D modeller av alle komponentene var tegnet opp og satt sammen til en sammenstilling, ble simulasjonoppsettet satt opp. I simulasjonen er det tatt utgangspunkt i det verst tenkelige scenarioet, for å se hvor store de maksimale spenningene kan bli i dette tilfellet. Dette scenarioet går ut på at lineær aktuator står og presser inn på gir staget med maksimal kraft på 300 N, når giringen har hengt seg fast slik at girsstaget er helt låst. Dette er en situasjon som kan representeres i en statisk kontaktanalyse i SolidWorks ved hjelp av FEM[4].

5.2.5.1 Låsinger

Det ble først satt opp «hinge» låsninger på Fish eye assy og rotasjonspunkt på Girspak, som er markert med grønne piler i Figur 5.9. Dette representerer en bolt som kun tillater rotasjon om rotasjonsaksen (låser all annen rotasjon og translasjon).



Figur 5.9: Simulasjonsoppsett for girstag. Påførte krefter er markert med lilla piler, og låsinger er markert med grønne piler.

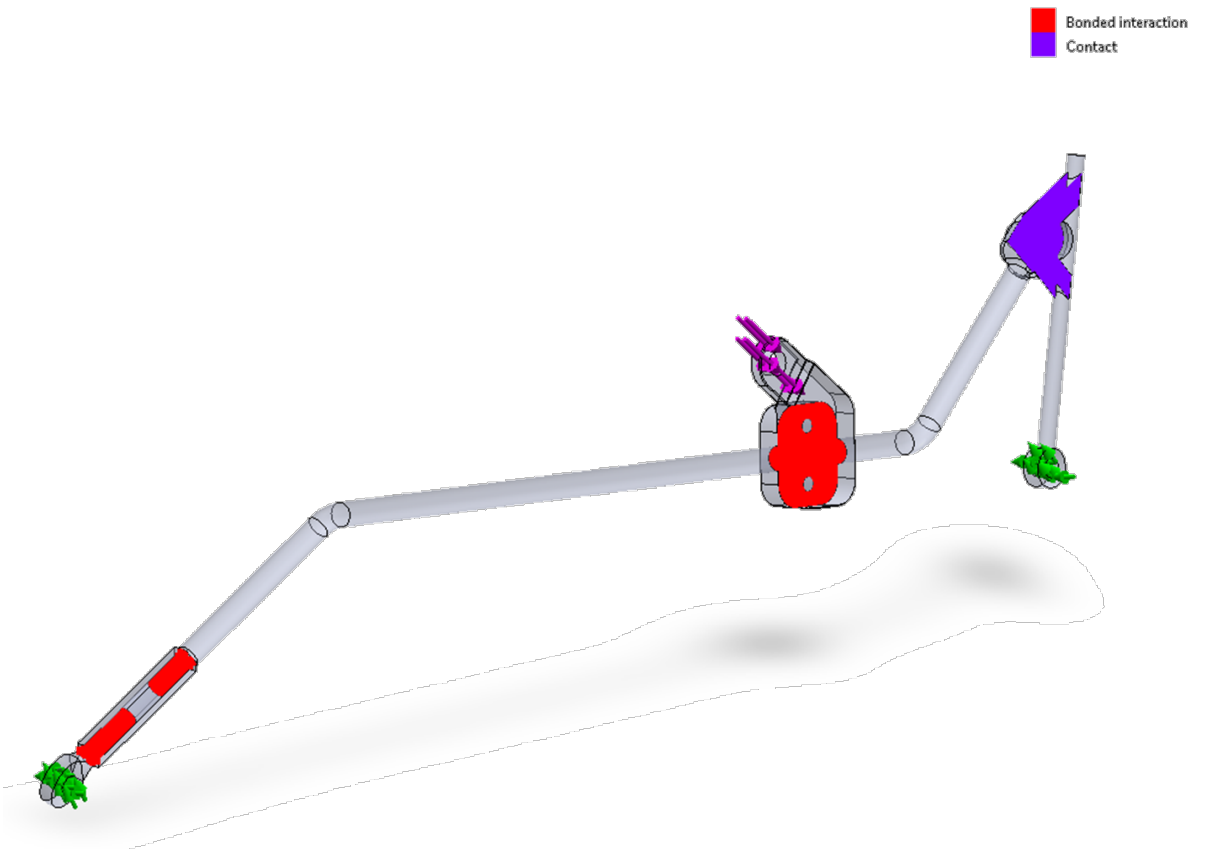
[4]Finite Element Method

5.2.5.2 Påførte krefter

For kraften som blir påført «fremside aktuator brakett giring» ble det brukt «bearing load». Denne representerer kraftoverføringen til bolteforbindelsen på en god måte, ettersom den gir samme sinusfordelte spenningskurve slik som en bolt har. «Bearing load» retningen kan også lett styres ved å lage et koordinatsystem i ønskelig orientasjon. Vinkelen som aktuatoren treffer braketten med er tilnærmet parallelt med vinkelen til braketten. Angrepsvinkelen fra aktuatoren på braketten vises med lilla piler i Figur 5.9.

5.2.5.3 Kontakt

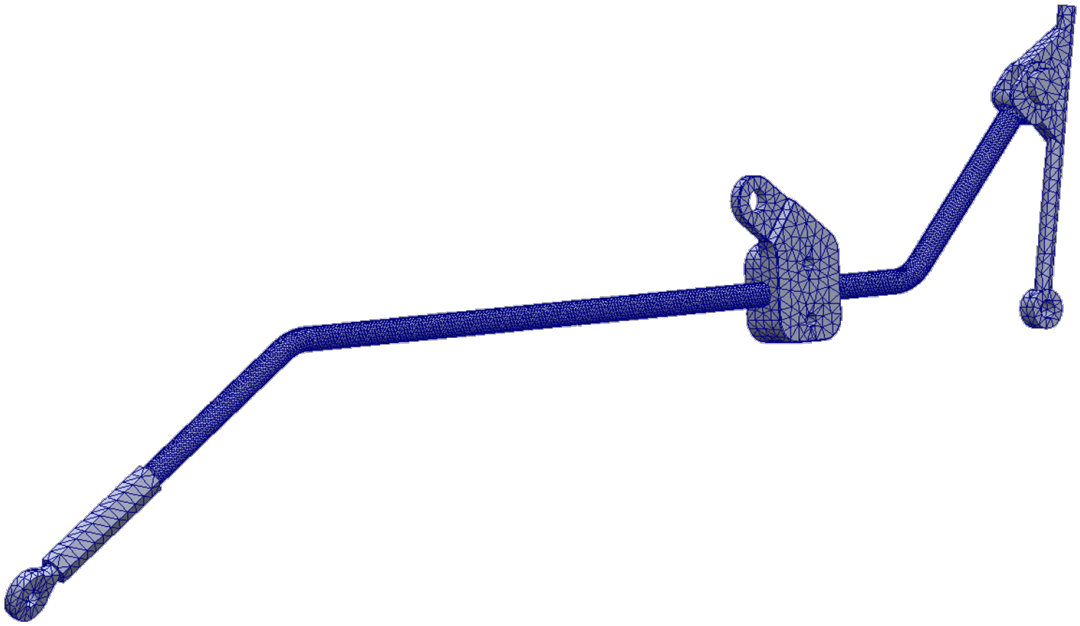
Hvordan type kontakt som er mellom delene er også definert for å få simulert spenningene på best mulig vis, uten at simulasjonen blir for tung å kjøre. Figur 5.10 viser en oversikt over de forskjellige type kontaktene i sammenstillingen. Det er spesifisert «Contact» lokalt på sideflater av ”Giringsstaghode” og medførende bolt, markert i lilla på Figur 5.10. Dette gir en simulering av at delene er i kontakt med hverandre med en virkende friksjonskraft. Friksjonskoeffisienten er satt til SolidWorks sin standard verdi på 0.05. Globalt er det satt «Bonded interaction». Dette medfører at delene oppfører seg som om de er sveiset eller limt sammen. På Figur 5.10 markert i rødt kan man se at gjengefeste av ”Fish eye assy” og ”Girstag” samt feste av ”Giringsbrakett” får «Bonded interaction». Dette representerer kontakten disse delene får når de låses fast ved hjelp av gjenger eller trykk rundt girsstaget.



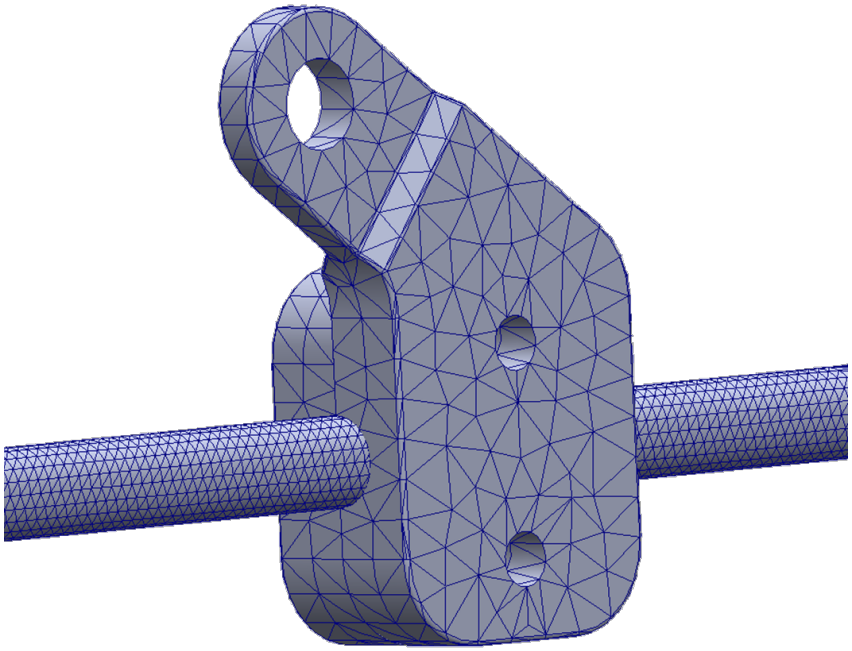
Figur 5.10: Forskjellige typer kontakt i girstaget. Krefter og låsinger er som tidligere markert med lilla og grønne piler. ”Bonded interaction”/”limet overflater” er fargelagt med rødt og kontaktoverflater er fargelagt med lilla.

5.2.5.4 Mesh

Et fint mesh og midtnoder er viktig for å få mest mulig nøyaktige resultater. Dette gjelder spesielt på flater med høy kurvatur. Det er dessverre veldig tidskrevende og tungt for datamaskinen å kjøre igjennom en simulasjon med fint mesh og med midtnoder. Det er derfor påført «Mesh control» på steder som er spesielt utsatte for store spenninger/kurvatur. På denne måten kan man få oversikt over spenningen på de kritiske punktene i konstruksjonen med god nøyaktighet, uten at det tar for lang tid å kjøre igjennom simulasjonen. Globalt er det satt et relativt grovt mesh, med en maksimal elementstørrelse på 5 mm. Alle delene inneholder noe form for kurvatur, så det er også satt på midtnoder globalt. Girsstaget og girspak bolt er både er utsatt for store spenninger og har mye kurvatur. Det er derfor benyttet «Mesh control» lokalt her for å gi et finere mesh. Her er elementstørrelsen satt til 1.5 mm, som gir i rett i underkant av 21 elementer pr 360°. Meshet til hele sammenstillingen er vist på Figur 5.11. Forskjellen på mesh størrelsen på aktuator brakett og giringsstag er vist i Figur 5.12.



Figur 5.11: "Mesh" for girstag assembly.



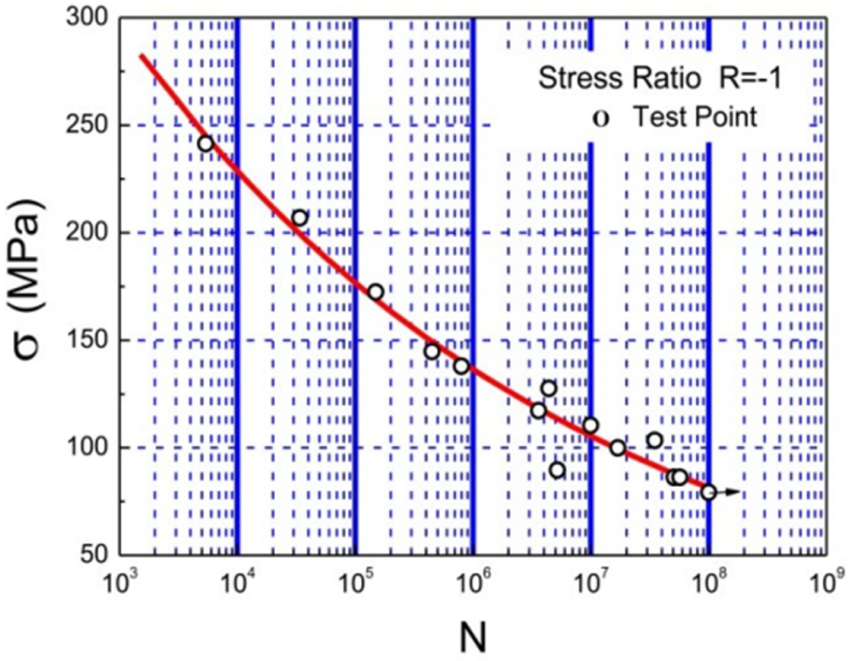
Figur 5.12: Braketten i girstag assembly "Mesh" på nært hold. Figuren viser tydelig forskjell på elementstørrelse i meshet.

5.2.6 Estimering av materiale

Det er uvisst hvilket materiale og legering girstaget består av. Can - Am kundestøtte ble kontaktet for å undersøke om det finnes tilgjengelig informasjon på dette. Tilbakemeldingen på dette var at de normalt ikke utga slik informasjon. Det ble derfor bestemt å gjøre en grov estimasjon på egenhånd. På bakgrunn av farge og vekt antas det å være en aluminium legering. Nøyaktig hvilken aluminiumslegering som er brukt er vanskelig å bedømme. Ettersom dette er et stag som blir utsatt for mye varierende spenninger (både trykk og strekk), antas det å være valgt en aluminiumslegering som er godt rustet for utmatting. Endefestet er også sveiset, som vil si at det er stor sannsynlighet for at det er brukt 5000 eller 6000 serie aluminiumslegering, da disse er de mest brukte aluminiums seriene for sveising. Disse legeringene har minimum 200 MPa flytegrense. Det blir derfor brukt denne verdien som utgangspunkt i beregningene. Det er også gjennomført et estimat på de andre komponentene tillhørende girsstaget. Basert på farge og vekt er det tatt utgangspunkt i at dette er en form for stål legering, men nøyaktig hvilken stål legering som er brukt er usikkert. For å være konservative er det tatt utgangspunkt i at delene er laget i plain carbon steel som er en relativt svak legering til å være stål. Denne legeringen har en flytegrense på sirka 220 MPa.

5.2.7 Sikkerhetsfaktor

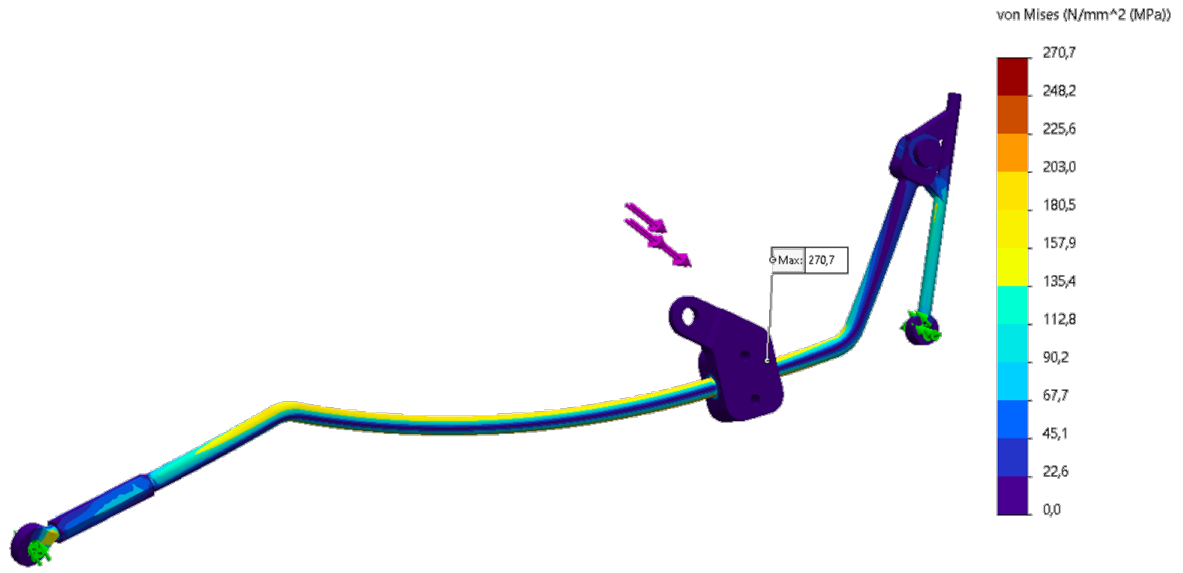
Aluminium som materiale er svært utsatt for utmattelse. Levetiden er direkte knyttet til hvilken sikkerhetsfaktor man velger, ettersom SN kurven til aluminium fortsetter å synke i stor grad når N (antall spennings sykluser) stiger. Dette kan ses på SN kurven til 6061 T6 aluminium som er vist i Figur 5.13. Staget er utsatt for svært varierende spenninger og vibrasjoner fra motor, som gjør at det er svært utsatt for utmattelse. På grunnlag av dette er det valgt en relativt høy sikkerhetsfaktor på 3 slik at vi får en lang levetid på staget.



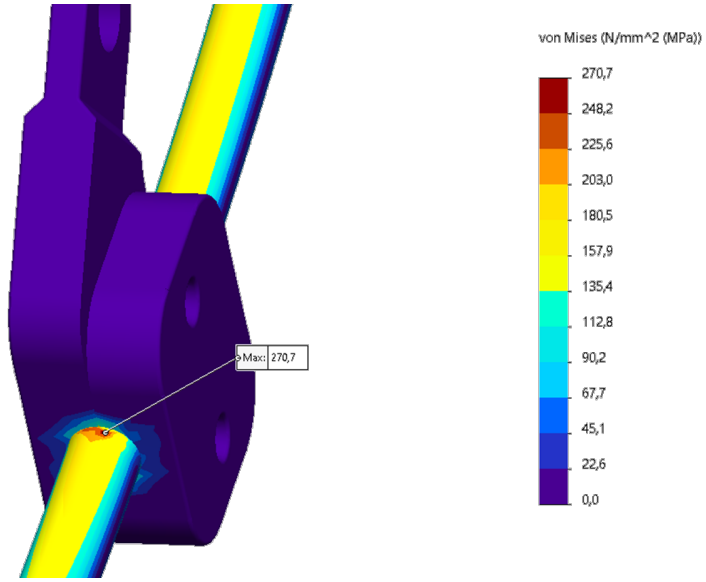
Figur 5.13: SN-kurve for Aluminium 6061 T6. Kurven viser antall stressykluser før feil for et gitt syklisk stress, hvor x-aksen gir antall sykluser før feil og y-aksen gir stress (i MPa). [3]

5.2.8 Resultater

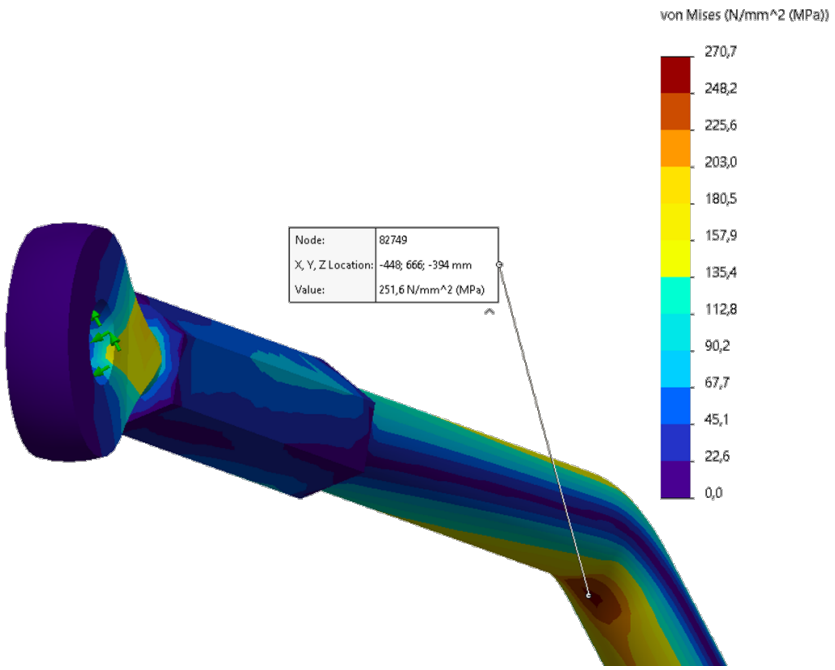
Resultatet av spenningene i sammenstillingen kan ses i Figur 5.14. Som du kan se oppstår de største vonMises spenningene i girsstaget rett på baksiden av brakett (Figur 5.15) samt på undersiden av knekk ved fiskeøye (Figur 5.16). Maksimal vonMises spenning er på 270.7 MPa når staget belastes med 300 N. Dette vil si at en reduksjon i kraft er helt nødvendig for å få en sikkerhetsfaktor på 3.



Figur 5.14: FEM-analyse av girstag assembly. Fargen tilsvarer Von Mises spenning på girstaget i MPa som vist i fargespekteret til høyre i figuren.



Figur 5.15: Maksimal spenning ved FEM-analyse av girstag assembly, på 270.7 MPa.



Figur 5.16: Spenning ved kurven til stag ved FEM-analyse, på 251.6 MPa

For å finne hvor stor kraft som kan påføres staget på aktuator med enn sikkerhetsfaktor på 3, må vi først finne tillatt spenning. Den er gitt ved Formel 5.1.

$$\sigma_{tillat} = \frac{\sigma_{flyt}}{sikkerhetsfaktor} = \frac{200\,\text{MPa}}{3} = 66.66\,\text{MPa} \tag{5.1}$$

Nå som tillatt spenning er kjent, kan man finne ut hvor stor fraksjon av maksimal spenning den tillatte spenningen utgjør. Denne fraksjonen kan deretter multipliseres med påført maksimal kraft, for å finne ut hvor stor den tillatte kraften aktuatoren skal skyve med er. Dette kan ses i Formel (5.2), der kraften fra aktuator ikke skal overstige 74 N.

$$F_{tillat} = \frac{\sigma_{tillat}}{\sigma_{max}} \cdot F_{max} \Rightarrow \frac{66.66\,\text{MPa}}{270\,\text{MPa}} \cdot 300\,\text{N} = 74\,\text{N} \tag{5.2}$$

Ettersom spenningen er konstant, vil det være et lineært forhold mellom kraften aktuator produserer og strømmen som går igjennom aktuatoren. For å forsikre oss at aktuator ikke overstiger dette, er det satt på et overstrømsvern på 1.23 A. (se Formel (5.3)). Detaljer rundt dette arbeidet er nærmere beskrevet i avsnitt 6.4.

$$I_{tilatt} = \frac{F_{tillat}}{F_{max}} \cdot I_{max} = \frac{74\,\text{N}}{300\,\text{N}} \cdot 5\,\text{A} = 1.23\,\text{A} \tag{5.3}$$

5.2.9 Begrensing av kraft fra aktuator

Nå som det er bestemt hvor stor kraft aktuatoren kan bruke må det gjennomføres en test der det verifiseres at aktuatoren overholder dette. Denne testen baserer seg på krav MEK2 og den tillhørende testen T_MEK2 som kan ses i Figur 5.17.

Prosjekt:	1651		
Krav ID:	MEK2	Dato:	18.04.2023
Kravklasse:	A	Status, krav:	V
Test ID:	T_MEK2	Status, test:	G

Kravspesifikasjon	Maksimal kraft påført girspak fra lineær aktuator skal ikke overstige de mekaniske begrensningene til systemet.
Beskrivelse av krav	Den lineære aktuatoren som skal brukes til girmekanismen har en maksimal kraft på 300N. Dette kan være nok til å skade komponenter på ATV. Det må derfor gjøres tiltak for å sikre at aktuatoren ikke tilfører større kraft enn det girsystemet tåler. Ved bruk av FEM (Finite Element Method) analyse og manuelle beregninger skal det estimeres hvor stor kraft aktuatoren kan bruke uten å skade komponenter.
Kriterium for bestått test:	Aktuatoren stanser dersom den overstiger 74 N.
Utførsel av testprosedyre:	Sett opp en testbenk som gjør det mulig å måle kraften som aktuatoren drar med. Monter aktuator i testbenk før deretter gjennomføre ett girskift. Testbenken skal nå hindre aktuatoren å bevege seg og det skal være mulig å lese av på en måler hvor stor kraften fra aktuatoren er.
Hvis kriteriet feiler:	Modifiser strømsperre og arduino kode.
Resultat av test:	Denne testen er godkjent. Testen ble gjennomført flere ganger med nye modifikasjoner på strømsperra, helt frem til lineær aktuatoren dro med en kraft på 73,5N.

Figur 5.17: Krav MEK2 og tillhørende test T_MEK2

Dette ble gjennomført ved å binde fast lineæraktuator og en hengevekt til et fast opplager og feste den sammen. Lineæraktuatoren ble deretter satt til å gjennomføre et girskifte. Verdien som leses av vil være den maksimale kraften lineæraktuatoren greier å trekke med innenfor grensen som er satt av strømsperren. Testen ble gjennomført flere ganger helt til strømsperren ble nøyaktig instilt slik at aktuatoren til slutt stoppet på 73,5 N. Oppsettet for testen og oppnådd kraftverdi kan ses i Figur 5.18.

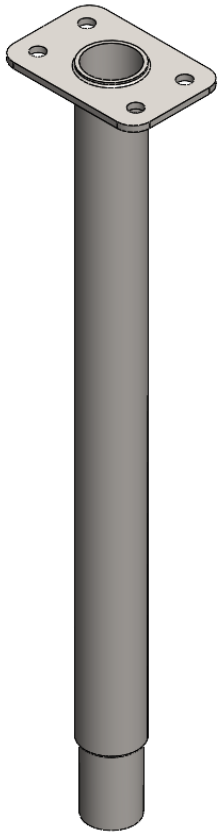
Figur 5.18: Test av påførte krefter på lineæraktuator

5.3 Styresystem

Som en alternativ utvidelse til hovedoppgaven, er det foreslått i oppgavebeskrivelsen å se på kraftoverføringen til styrestammen. Hovedoppgaven som omhandler automatisk giring har vist seg å være en større utfordring når det kommer til software og elektronikk, i forhold til det mekaniske. Det har derfor vært mulig og relativt tidlig begynne å se på andre mekaniske problemstillinger for ATV.

5.3.1 Problemstilling styresystem

Etter å ha observert hvordan styresystemet fungerer i praksis, er det tydelig at det har forbedringspotensialet. Først og fremst er kreftene som blir påført styrestammen svært store. Dette resulterer i at hele styrestammen bøyes når ATV skal svinge. Som følge av dette, er faren for utmattingsbrudd stor. For å få en bedre forståelse av hvordan denne styrestammen er utviklet, ble det tatt noen raske undersøkelser på nett. Her dukket det raskt opp klager fra forbrukere i forskjellige forum at styrestammen til denne typen ATV har en tendens til å få utmattingsbrudd på grunn av bøyspenninger under «tøff» bruk. Etter å ha observert hvor mye styrestammen bøyes slik styresystemet er satt opp i dag, kan man med trygghet si at styrestammen er utsatt for «tøffe» påkjenninger slik som blir beskrevet i forumene. På bakgrunn av dette ble det konkludert at det er stor sannsynlighet for at styrestaget ikke vil tåle de påkjenningene som den blir utsatt for i lengden. Siden faren for utmattingsbrudd er stor, er det derfor behov for å gjøre tiltak som kan forsterke styrestammen. I Figur 5.19 vises en enkel 3D-modell av styrestammen til ATV.



Figur 5.19: Styrestamme

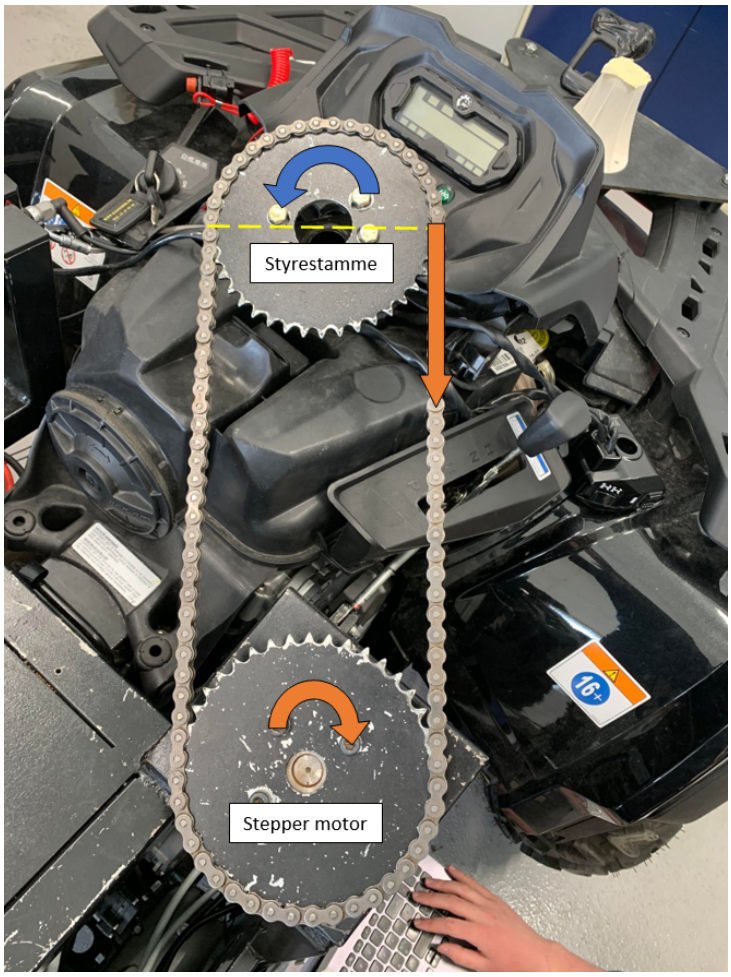
Under kjøring av ATV, opplevdes styresystemet også svært tregt. Dette medfører at ATV'en er lite manøvrerbar. For eksempel i de situasjonene hvor man går fra fullt rattutslag fra en side og vil over til fullt utslag mot motsatt side. I disse situasjonene bruker ATV lang tid på å rotere styret over i ønsket retning, noe som fører til en svært stor svingradius. Det er derfor ønskelig å øke hastigheten på styresystemet. Steppermotoren som brukes er svært kraftig og kan justeres til å rotere raskere, men dette vill resultere i større krefter på styrestammen som igjen øker behovet for å avstive styrestammen.

5.3.2 Reverse engineering

Dagens styresystem består av 2 tannhjul, kjede og en stepper motor. ATV'ens styre er fjernet og blitt erstattet med et tannhjul. På det ettermonterte rammeverket er det festet en steppermotor. På denne motoren sitter det ett nytt tannhjul som er koblet sammen med tannhjulet på styrestammen ved hjelp av et kjede. ATV kan så svinge ved å styre stepper motoren til ønsket posisjon. Steppermotoren er presis og sterk, og egner seg derfor godt til jobben.

For å lage en best mulig løsning for avstivning er det viktig å kartlegge hvor kreftene i det eksisterende systemet virker i fra før. Som nevnt, bruker systemet et kjede for å overføre dreimomentet fra stepper motor og over til styrestaget. Ulempen med dette er at vi får en stor strekkkraft i kjedet på den ene siden av tannhjulene. Kjedet greier i svært liten grad å overføre trykk spenninger, som medfører at krefter som overføres på den andre siden av tannhjulet er neglisjerbare. Dette resulterer i store krefter som prøver å dra tannhjulene mot hverandre.

Figur 5.20 viser hvordan kreftene om tannhjulet som driver styrestammen virker. Stepper motoren er festet svært godt i rammeverket, som medfører at den står støtt selv ved store kraftpådrag. Den oransje pilen representerer kraften steppermotoren påfører tannhjulet festet til styrestaget, imens den blå momentpilen representerer motmomentet som holder igjen for rotasjonen. Motmomentet markert i blå kommer i all hovedsak av friksjonskreftene mellom underlag og hjulene som må overvinnes for å skape rotasjon. Med andre ord må den kraften som påføres tannhjulet i fra stepper motor (oransje) overvinne friksjonskreftene fra underlaget (blå) for å skape rotasjonen. Dette medfører at summen av krefter om aksen som står normalt på kreftene (markert i gul) virker i retningen til den oransje pilen. Det er denne resultant kraften som skaper bøyemomentet i styrestaget.



Figur 5.20: Krefter i styremekanisme

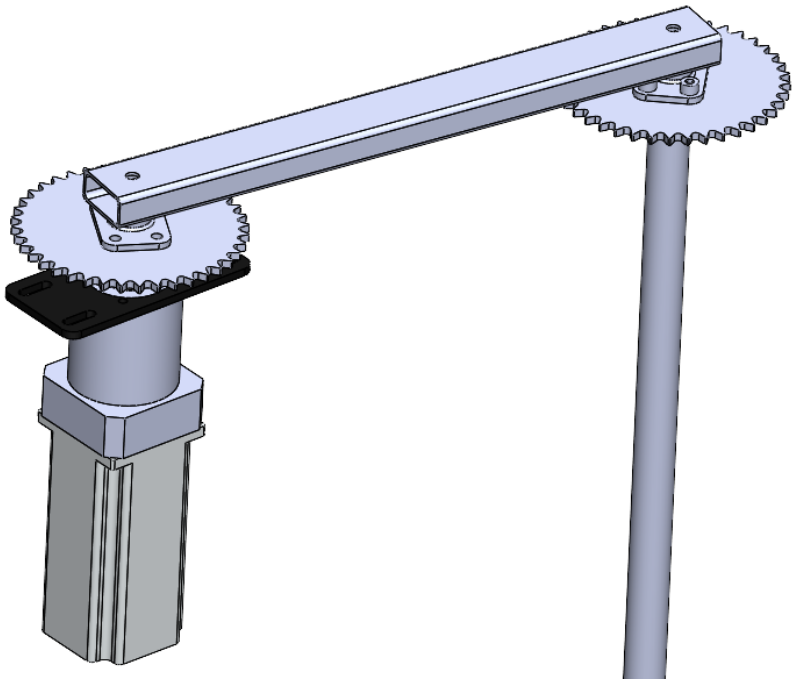
En annen svakhet med systemet er at det oppstår slakk i den delen av kjede som ikke er under strekk. Som nevnt, fører kreftene som oppstår til at styrestammen og steppermotoren blir dratt mot hverandre. Siden styrestammen ikke er stiv nok, bøyer denne seg mot steppermotoren som resulterer i at avstanden mellom rotasjonsaksene blir kortere. Det er denne avstandsreduksjonen som fører til slakket i kjede. Etter skifte av rotasjonsreting på steppermotoren, må motoren alltid rotere noen grader for å få strammet opp slakket i kjede helt til det blir stramt nok til å overføre krefter i motsatt retning på styrestammen. Et resultat av dette er at det oppstår en forsinkelse hver gang steppermotoren endrer rotasjonsretning som igjen fører til at styringen oppleves lite responsiv når den fjernstyres. Styresystemet bør derfor endres eller utbedres slik at også denne problemstillingen elimineres.

5.3.3 Alternative løsninger

Nå som de forskjellige problemstillingene for styresystemet er kartlagt, kan forskjellige løsninger på problemene undersøkes. Det er hovedsakelig fire hovedalternativer som er undersøkt:

Alternativ 1:

En mulig løsning kan være å koble tannhjulene sammen med et mellomledd som holder igjen kreftene som presser tannhjulene mot hverandre. Det er flere forskjellige måter dette kan gjennomføres på. Det kan benyttes flere forskjellige typer mellomledd som for eksempel forskjellige type rør, stenger eller plater. Et kulelager som forminsker friksjonen i bindeleddet mellom roterende del og mellomleddet er gunstig. Dette kan både plasseres i en brakett festet direkte i tannhjulet, eller i selve mellomleddet. I Figur 5.21 kan du se et av konseptene som viser hovedprinsippet. I dette eksempelet er det brukt et firkantrør som er festet til 2 braketter som er skrudd fast til hvert tannhjul. Inne i brakettene er det plassert kulelager som holder en bolt fra firkantrøret.



Figur 5.21: Forslag på konsept for alternativ 1

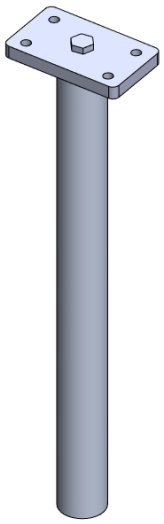
Et slikt system vil effektivt avlaste styrestammen ved å hindre at tannhjulene kan bli dratt mot hverandre. Løsningen krever også svært lite modifikasjoner på det som allerede er laget. Ved å feste braketter til boltene som allerede holder tannhjulene på plass kan

man enkelt legge til et system som dette. Dette systemet har også noen negative sider. Det vil oppstå friksjon mellom brakett og mellomledd som vil føre til at stepper motor må jobbe hardere. Det er heller ikke gunstig og legge til mer vekt på ATV. Tannhjulene sitter også svært høyt oppe på kjøretøyet, som medfører at all vekt som blir lagt til i dette område vil påvirke tyngdepunktet til ATV i større grad.

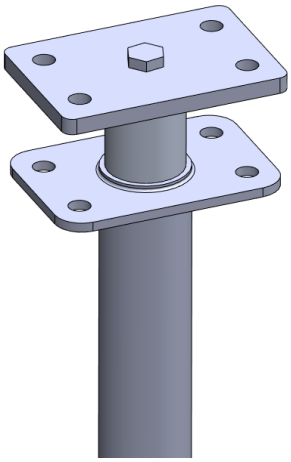
Alternativ 2:

Dette alternativet går ut på å stive av styrestammen ved hjelp av ett nytt rør. Det er denne delen som er det svake leddet i styresystemet ettersom stepper motoren er festet svært godt til det ettermonterte rammeverket. Ved å forsterke denne delen kan man minske faren for at delen får utmattingsbrudd. Styrestammen består av et metallrør med et påsveiset festehode. For å få denne delen stivere kan man plassere ett nytt rør/stang inne i styrestammen for å øke den totale veggtykkelsen og avlaste den originale styrestammen. Et konsept på hvordan denne delen kan se ut er vist i Figur 5.22. Det påsveisede festhode er også utsatt for store spenninger i boltforbindelsen som fester tannhjulet til styrestammen. I Figur 5.23 ser man hvordan avstiveren skal monteres i styrestammen. Her kan man også se at når disse delene blir festet sammen med 4 bolter vil avstiveren også avlaste det påsveisede festehode i tillegg til å stive av for bøyspenningene.

Dette er en svært enkel løsning som vil være rask og produsere. Den vil ikke øke friksjonen slik som i alternativ 1, men den vil i likhet øke vekten på styresystemet. Styrestammen vil fortsatt være utsatt for store krefter fra stepper motoren men vil med dette være i stand til å motstå kreftene bedre.



Figur 5.22: Styrestamme avstiver

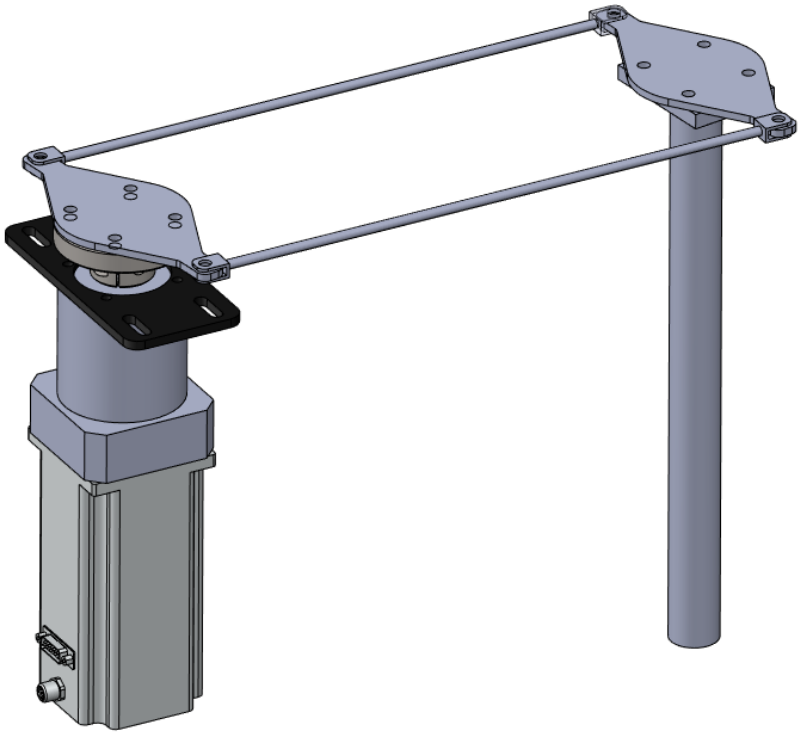


Figur 5.23: Avstiver delvis montert i styrestamme

Alternativ 3:

Ett av hovedproblemene med bruk av kjededrift til å svinge er at kraften kun blir overført i form av en drakraft. Som nevnt fører dette til at styrestammen blir bøyd, som igjen fører til dårlig overføring av krefter og fare for utmattingsbrudd. Ved å finne en løsning der trykk og strekk krefter er like på hver side av rotasjonsaksen vil summen av krefter mellom tannhjulene bli null. På denne måten kan bøyemomentet i staget elimineres helt. Det finnes mange metoder for å overføre rotasjonskraft på denne måten fra aksling til aksling. Et eksempel på dette er vist i Figur 5.24, her brukes to stag til å overføre krefter mellom to roterende plater. Konseptet er simpelt og består av enkle deler. Styrestaget

trenger kun å roteres ±45 grader for å få maksimalt rattutslag, som vil si at det ikke er fare for at stagene vil kollidere med hverandre.

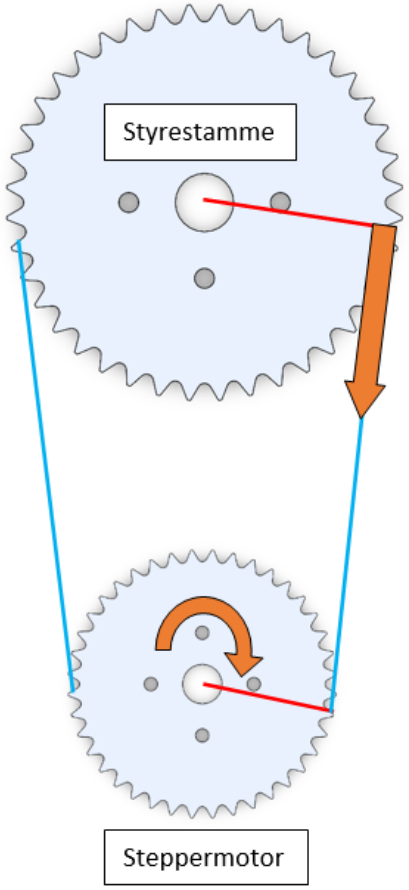


Figur 5.24: Krafoverføring ved bruk av to stag

Alternativ 4:

Dette alternativet går ut på å endre på tannhjulforholdet. I dag har tannhjulene et 1 til 1 forhold. Ved å øke diameteren på tannhjulet på styrestammen, vil man få en lengre kraft arm. Dette medfører at kraften påført tannhjulet kan være mindre for å oppnå samme dreiemoment inn på styrestammen. Ettersom påført kraft på tannhjulet til styrestammen kan være mindre, vil også bøyemomentet på staget bli mindre. Som en konsekvens av dette får kjedet en lengre avstand å reise rundt tannhjulet til styrestammen, som resulterer i at steppermotor må rotere raskere for å opprettholde samme svingehastighet. Oppsettet med påsatte krefter er vist i 5.25.

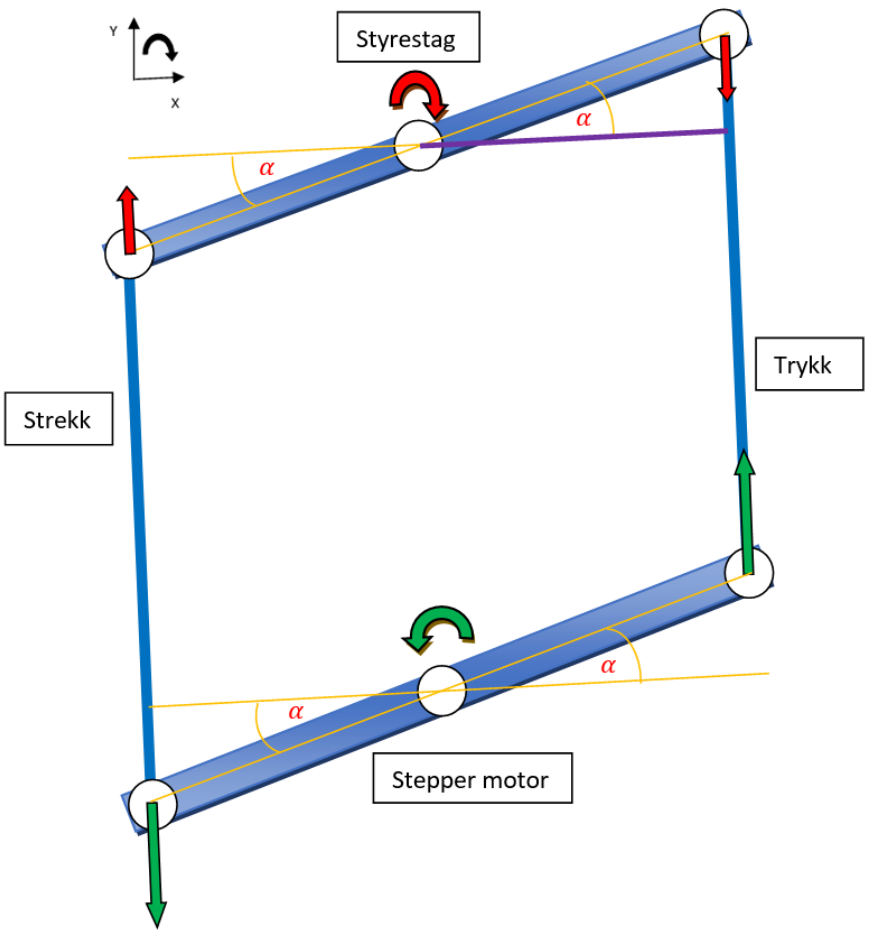
Dette alternativet vil være raskt og enkelt å implementere mekanisk. Her må kun kjøpes inn nye tannhjul for å endre forholdet etter ønske og eventuelt endre lengde på kjede. Ulempen med dette alternativet er at det fortsatt vil være en kraft som medfører bøyemoment selv om den i utgangspunktet vil være mindre enn før. Stepper motor må også tunes inn på nytt med tanke på dreiemomentet den påfører og hastigheten den skal rotere med.



Figur 5.25: Kjededrift med endret tannhjulforhold

5.3.4 Valg av løsning

Etter å ha sett på de forskjellige alternativene ble det bestemt å se nærmere på alternativ 3. Et fritt legeme diagram ble satt opp for å undersøke nærmere på hvor kreftene virker i systemet. Dette kan ses i Figur 5.26. I denne figuren representerer de grønne pilene kreftene som påføres fra stepper motoren, imens de røde pilene representerer motkreftene som kommer fra friksjonskreftene fra underlaget som må overvinnes for å skape rotasjon. På grunn av retningen til kreftene vil det blir trykk i stagene på høyre side og strekk i staget på venstre side i figuren.



Figur 5.26: Krefter ved kraftoverføring med stag

Stagene som overfører kreftene i Figur 5.26 vil alltid være parallelle og peke rett frem (Y-retningen til koordinat systemet). Hvis det benyttes like armlengder på stepper motor og styrestag, vil vinklene på armene dermed også alltid være identiske. Dette vil si at rotasjonen på styrestammen vil være identisk med rotasjonen til motor.

Kraftarmen til styrestaget er markert med lilla i Figur 5.26. Formelen for kraftarmen, der L representerer kraftarmen og a representerer armlengden til de roterende skivene, er gitt ved:

$$L = Cos(\alpha) \cdot a \tag{5.4}$$

Ettersom både armelengder og vinkelen (α) er like både ved stepper motoren og styrestammen, vil også kraftarmene være like.

Nå som det er kjent at kraftarmene til styrestammen og stepper motor alltid er like kan det også vises at dreiemomentet på stepper motor og styrestamme også er likt og

konstant under hele rotasjonen. La oss ta tilfellet der stepper motor påfører et konstant dreiemoment på den roterende skiven. I dette tilfellet vil en økt vinkel resultere i at kraftarmlengden minker, som medfører at kraften stepper motor påfører staget øker. Dette kan demonstreres ved å se på ligningen for dreiemoment(T):

$$T = F \cdot L = F \cdot a \cdot cos(\alpha) \Rightarrow F = \frac{T}{a \cdot cos(\alpha)} \tag{5.5}$$

Ettersom vinkelen til styrestaget kun skal bevege seg ±45 grader, vil cosinus verdien alltid minke når absolutt verdien til vinkelen øker innenfor vinkelrammene. Når cosinus verdien i nevneren minker, vil dette medføre at kraften øker.

På styrestammen vil en stor kraftarm føre til at dreiemomentet inn på styre stammen blir stor. På samme måte som ved stepper motoren, vil kraftarmlengden ved styrestammen blir mindre når absolutt verdien til vinkelen øker. Her vil altså en større vinkel medføre en kortere armlengde, som medfører et lavere dreiemoment.

Ettersom både dreiemomentet inn på styrestammen og kraften ut fra stepper motoren påvirkes av cosinus verdien, vil de nøytralisere hverandre slik at dreiemomentet på styrestammen teoretisk sett vil være identisk med dreiemomentet fra stepper motoren. En forutsetning for at dette skal stemme er at like armlengder benyttes og at friksjonen i bindeleddene neglisjeres.

Dette kan demonstreres ved et eksempel med vilkårlige verdier for dreiemoment og kraftarmlengde. Bakgrunnen for at vilkårlige verdier er benyttet her, er fordi det kun er prinsippet om at dreiemomentet fra steppermotor og styrestaget er det samme som skal vises frem her. La oss ta tilfellet der det påføres et konstant dreiemoment på 110Nm fra stepper motor. Hvis 100Nm av av dreiemomentet går tapt til å overvinne friksjonskrefter mellom underlaget og hjulet, vil stepper motor ende opp med et overskudd i moment T1 på 10Nm. Hvis vi ser for oss en vinkel på 25 grader på en armlengde på 0,2m, vil kraften fra staget som går opp i styrestammen være gitt ved formel 5.6:

$$F = \frac{T1}{a \cdot cos(\alpha)} = \frac{10Nm}{0,2m \cdot cos(25)} = 55,17N \tag{5.6}$$

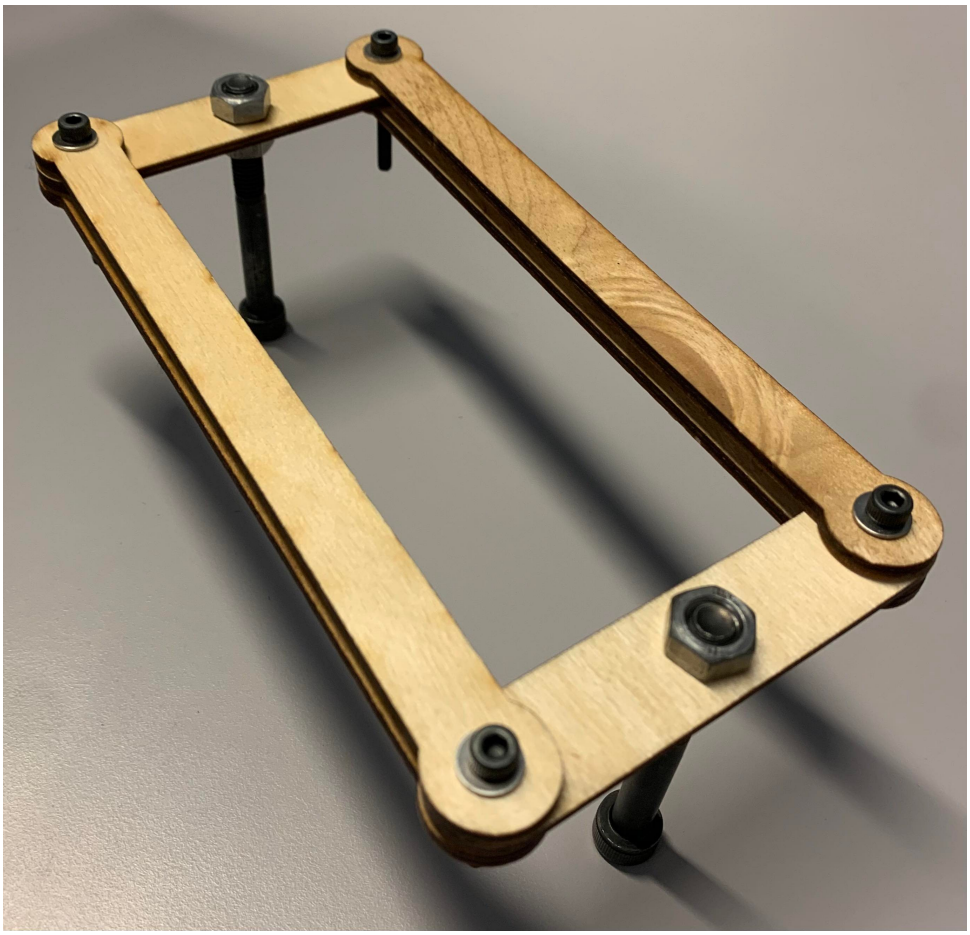
Vi kan nå sette inn verdien for kraften inn i ligningen for dreiemomentet inn på

styrestammen:

$$T = F \cdot L = F \cdot a \cdot cos(\alpha) = 55,17N \cdot 0,2m \cdot cos(25) = 10Nm \tag{5.7}$$

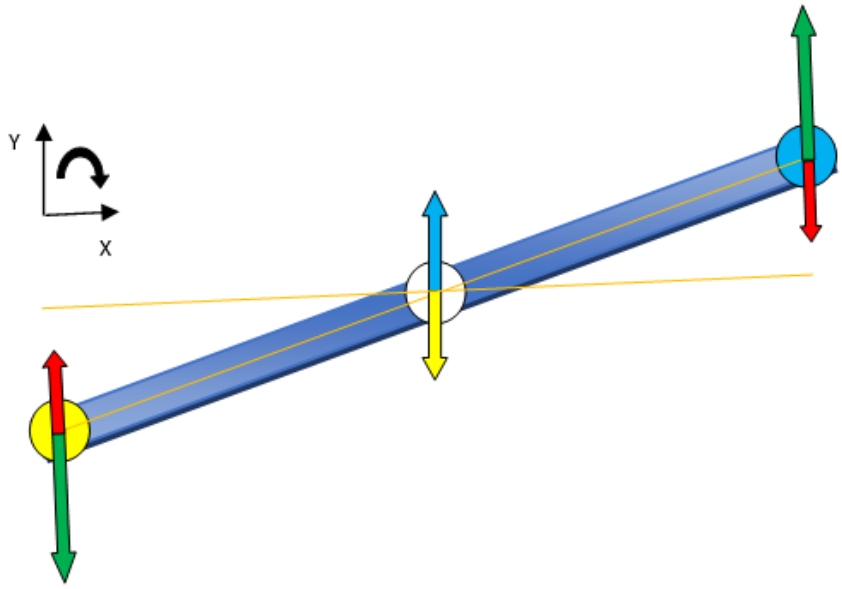
Ut i fra dette eksempelet kan det vises at dreiemomentet fra stepper motor og styrestammen vil være identiske til enhver tid.

For å verifisere at dette prinsippet også fungerer i praksis ble det laser kuttet ut en miniatyrprototype i kryssfiner som kan ses på Figur 5.27. I denne prototypen vil de to store boltene i senter av hver skive representere stepper motor og styrestammen. Disse er festet til skivene ved å stramme til muttere på hver side, slik at rotasjonskrefter kan overføres. Ved å prøve å vri på den ene bolten og holde igjen på den andre kan en kjenne på hvordan dreiemomentet blir påvirket av forskjellige vinkler på armene. Denne lille testen verifiserte teorien om at dreiemomentet er konstant ved at det føltes like tungt ut å vri uansett hvilken vinkel armene stod i. Dette vil i praksis si at en kan benytte et lineært motorpådrag på stepper motoren for å oppnå en konstant svingehastighet.



Figur 5.27: Miniatyrprototype av styresystemet

I Figur 5.28 kan man se kreftene som påføres deler i kontakt med styrestammen. Hvis kreftene som påføres bindeleddene mellom stag og roterende skive summeres i Y-retning vil man ende opp med to like store resultant krefter med motsatt retning. Disse to resultant kreftene er illustrert som blå pil for høyre side, og gul pil for venstre siden. Hvis vi summerer disse kreftene i Y-retning på styrestammen kan det ses at de eleminerer hverandre, slik at summen av krefter på styrestammen blir null. I praksis betyr dette at det ikke vil være krefter som bidrar til å bøye styrestammen.

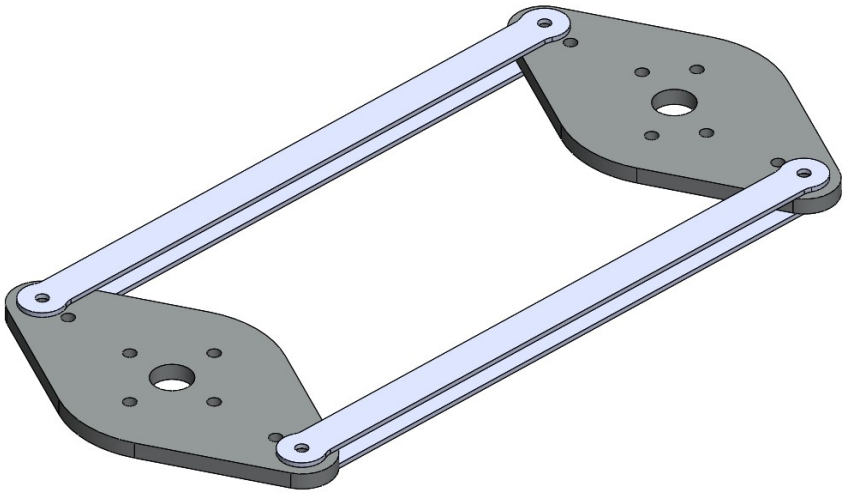


Figur 5.28: Krefter på styrestamme

Etter å ha vurdert fordelene og ulempene med alle alternativene falt valget til slutt på alternativ 3. Alle alternativene ble framlagt under progresjonsmøte med veilederne fra KDA. Etter å ha diskutert for og imot disse forslagene ble det enighet mellom gruppen og veilederne om å gå videre med dette alternativet.

5.3.5 Konseptuell design

For å få bedre kjennskap til hva som kreves for å implementere et slikt styresystem ble det bestemt å lage en enkel prototype i kryssfiner. Ved å bruke en laserkutter kan man enkelt skjære ut kryssfinerplater som egner seg godt til testing av forskjellige konsepter. I Figur 5.29 kan man se prototypen som ble laget. For å få en bedre forståelse av hvordan lengden på momentarmene påvirker bevegelsen ble det laget et ekstra festepunkt til stagene for å kunne justere dette. I figuren ser man stagene montert på ytterste posisjon.



Figur 5.29: Prototype i kryssfiner

Når prototypen var montert på ATV ble det først testet å svinge uten at hjulene var i kontakt med bakken for å minimere friksjonsmotstanden. Dette fungerte svært godt, og styringen føltes svært responsiv og presis. Imotsetning oppstod det slakk i kjede ved raske rattomslag som førte til litt dødgang når rotasjonsretningen skiftet i det gamle Styresystemet. Dette ble nå unngått. Det ble også verifisert at styrestammen ikke bøyde seg som tidligere. Dette bekreftet igjen at utregningene for kraftoverføringene er korrekte. Til slutt ble ATV jekket ned igjen på gulvet for så å teste hvordan prototypen fungerte under større motstand. Her også fungerte prototypen svært godt. Steppermotoren var nå i stand til å rotere styrestammen til fult rattutslag imens ATV var stillestående. Dette var ikke tilfellet med det gamle systemet. Dette indikerer at kraftoverføringen allerede er mer effektiv enn det gamle styresystemet.

Figur 5.30: Prototype montert på ATV

Prototypen viser bare en av mange mulig løsninger til å lage et styresystem ved bruk av stag. Ved produksjon av denne prototypen var det fokus på å lage en enkel og solid modell som kunne gi svar på hvordan mekanismen fungerte i praksis og hvordan det var å montere på ATV. Det ble derfor besluttet å modellere forskjellige konsepter for så å evalure disse for å kunne ta en bedre avgjørelse på hva som er det beste valget å produsere.

5.3.6 Testing av krefter fra steppermotor

For å få en bedre forståelse av hvor store krefter som påføres styresystemet er, ble det utført en test for å måle hvor stort moment steppermotoren klarer å påføre. Dette er basert på kravet MEK4 og den tillhørende testen T_MEK4, som kan ses i Figur 5.31.

Prosjekt:	1826		
Krav ID:	MEK4	Dato:	18.04.2023
Kravklasse:	A	Status, krav:	V
Test ID:	T_MEK4	Status, test:	G

Kravspesifikasjon	Styrestammen skal ikke påføres krefter som overstiger de mekaniske begrensningene til styrestammen.
Beskrivelse av krav	Dersom styrestammen blir utsatt for store krefter er det fare for at den tar skade. Ved å gjennomføre en FEA (Finite element analysis) skal det estimeres hvor stort moment den kan påføres uten at det går utover levetiden til komponenten.
Kriterium for bestått test:	Styrestammen skal ikke påføres krefter som medfører fare for brudd. Det tas utgangspunkt i en sikkerhetsfaktor på 2.
Utførsel av testprosedyre:	Dreiemomentet stepper motor leverer må først kartlegges før en simulasjon kan settes opp: 1. En testarm festes til stepper motor, og vil fungere som en kraft arm. 2. Hengevekt festes deretter til enden av kraft armen, og festes til et fast opplager som holder igjen hengevekten i andre enden. 3. Maksimal sving blir deretter iverksatt, og kraften på hengevekten noteres ned. 4. Dreiemomentet stepper motor leverer kan deretter bli regnet ut. Etter at dreiemomentet fra stepper motor er kartlagt, kan simulasjonen bli satt opp for å verifisere at spenningsnivået ligger under halve flytegrensen.
Hvis kriteriet feiler:	Motor styrke kan reduseres slik at spenningsnivået i styrestammen ikke overstiger begrensningen.
Resultat av test:	Denne testen ble godkjent. Ut ifra simuleringene, ble det kartlagt at styresystemet har en statisk sikkerhetsfaktor på 4,2, og at spenningsnivået er lavt nok for at det ikke er fare for utmattelse brudd.

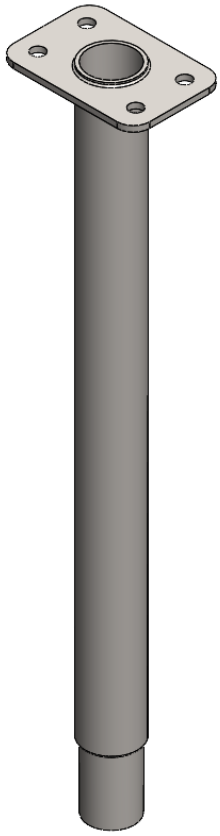
Figur 5.31: Krav MEK 4 og tillhørende test T_MEK4

I Figur 5.32 kan man se hvordan testen ble utført. En hengevekt ble ankret fast til rammeverket og festet på prototypen montert på steppermotoren med en vinkel på 90 grader. Når man da prøver å svinge mot venstre vil steppermotoren prøve å rotere og man kan i dette øyeblikket lese av på hengevekten hvor stor kraft motoren klarer å dra med. Som vist i figuren ga dette en maksimal kraft på 725.6 N. Hengevekten er festet til et hull 100mm fra rotasjonsaksen til steppermotoren. Dette resulterer dermed at vi får et maksimalt moment på 72,56 Nm. Fra databladet tilhørende steppermotoren blir det gitt at motoren kan holde et maksimalt moment på 9,3 Nm, men siden motoren er montert på en girkasse med 1:9 girutveksling skal den teoretiske maksimale momentet ligge på 83,7 Nm. Slike girkasser vil alltid føre til et lite tap av krefter på grunn av friksjon. Kraften som ble målt under testing anses dermed som den maksimale kraften som steppermotoren klarer med denne konfigurasjonen.

Figur 5.32: Måling av moment fra steppermotor

5.3.7 Mekanisk analyse av styrestamme

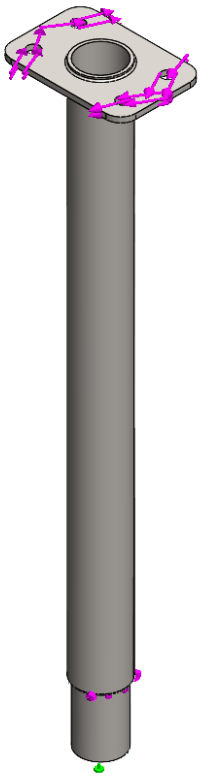
Som fortsettelse av testprossedyren i T_MEK4, kan det nå settes opp en statisk analyse med det maksimale dreiemomentet som ble funnet. Dette er viktig for å verifisere at styrestammen ikke tar skade av denne belastningen. Før analysen kunne bli satt opp, ble det tatt fysiske mål av styrestammen slik at den kunne bli modellert opp i SolidWorks. Denne 3D modellen kan ses i Figur 5.33.



Figur 5.33: Styrestamme

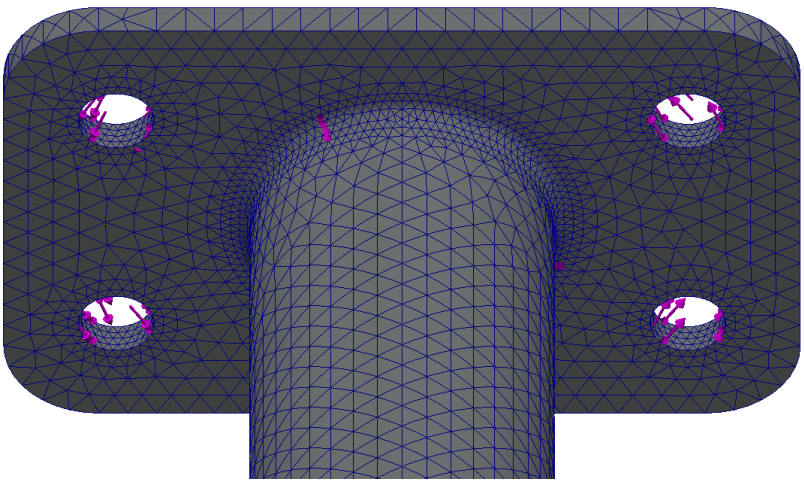
Den statiske analysen tar utgangspunkt i det verst tenkelige situasjonen, der steppermotor overfører maksimalt dreiemoment til styrestammen, og hjuelene på ATV låses helt. Dreiemomentet som påføres styrestammen, overføres videre til hjulene ved hjelp av en del som er festet på nedsiden av styrestammen. For å simulere låsingen av hjulene, er det dermed satt på "Fixed geometry" på innsiden av dette hullet. Dette kan ses nederst på Figur 5.34 markert med grønn pil.

For å simulere dreiemomentet som overføres fra steppermotoren, er det benyttet SolidWorks sin dreiemoment funksjon på de fire hullene, med rotasjonsakse om styrestammens senter akse. Denne funksjonen regner ut vinkel og størrelse på kreftene som blir påsatt hullene, slik at totalen blir det maksimale dreiemomentet til steppermotor på 72,56 Nm. Dette kan ses markert med lilla piler i Figur 5.34.



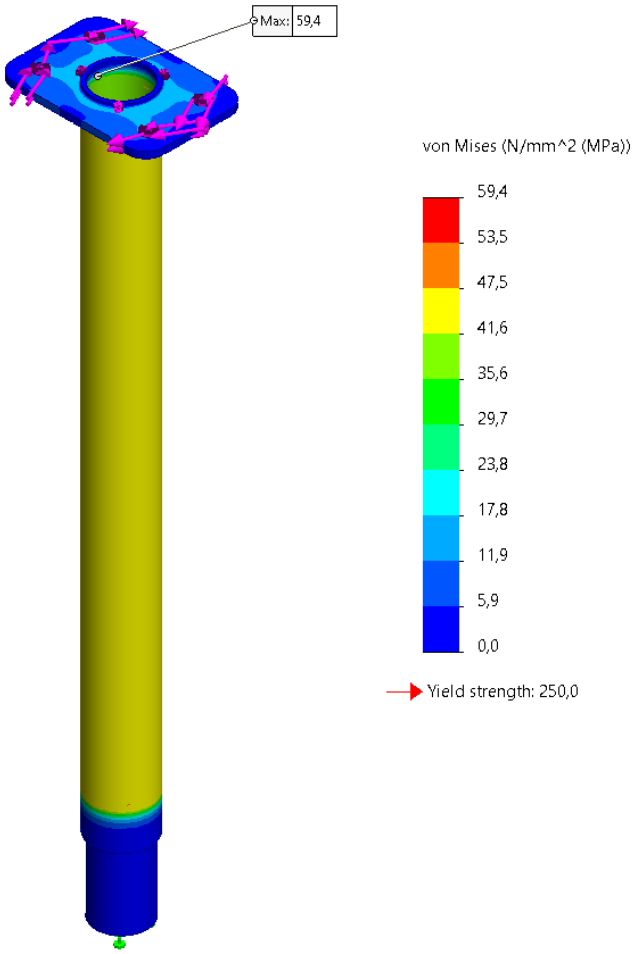
Figur 5.34: Oppsett for statisk analyse av styrestamme

Ettersom dette kun er en analyse av en enkelt del, kan det settes et relativt fint mesh på globalt på delen uten at analysetiden blir alt for lang. Element størrelsen er derfor satt til å være 3mm eller mindre. Det er også spesifisert et enda finere mesh med 1mm maksimalt elementstørrelse lokalt på filets med høy kurvatur, og på hullene som belastes av kreftene fra dreiemomentet. Forskjellen på global og lokal elementstørrelse kan ses på Figur 5.35.



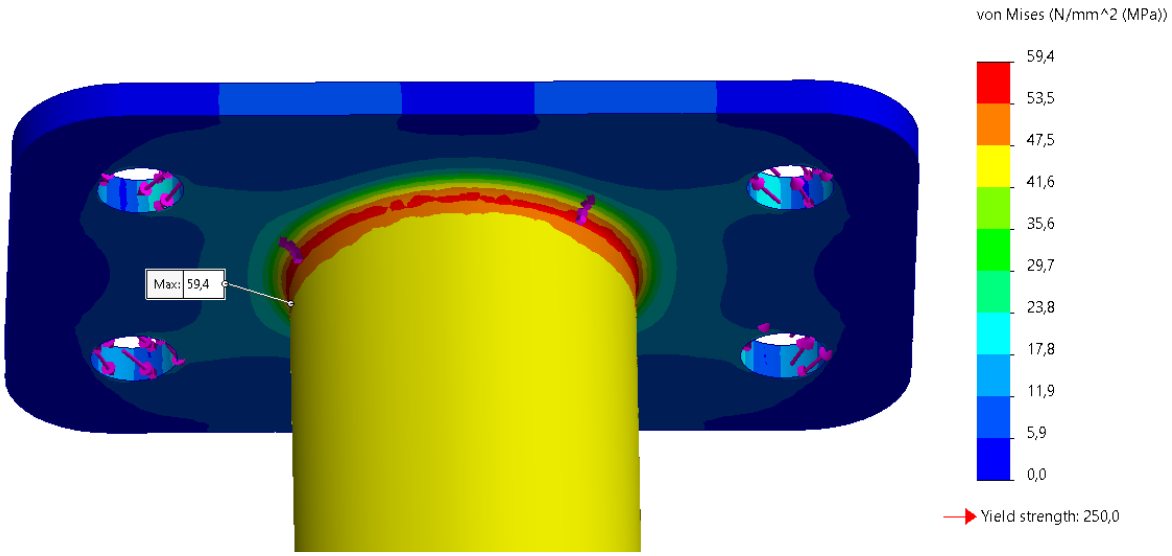
Figur 5.35: Mesh i statisk analyse av styrestamme

Det må deretter bli gjort et estimat på hvordan materiale styrestammen er laget av. Styrestammen er malt på utsiden, som i utgangspunktet gjør det vanskelig å estimere materiale ut i fra farge. Etter undersøkelser på nettet, ble det funnet bilder av styrestammen etter deler av malingen har falt av. På disse bildene var det klare tegn på rust som indikerer at staget er laget av en stål legering. Ettersom staget er sveist, kan det tas utgangspunkt i at dette er en lavkarbon legering, ettersom disse har best sveisbarhet. For å være mest mulig konservativ i analysen, kan det tas utgangspunkt i en av de svakere lavkarbon stål legeringene. Et eksempel på en slik lavkarbon legering kan være A36 stål, som har en flytegrense på 250 MPa.



Figur 5.36: Resultat av statisk analyse av styrestamme

I Figur 5.36 kan resultatet av analyses ses. Den største spenningen som ble oppnådd, ligger i en radius ved overgangen fra røret til platen i toppen og ligger på 59,4 MPa. Dette kan ses i mer detaljer i Figur 5.37.



Figur 5.37: Maksimal spenning på styrestamme

Med en flytegrense på 250 MPa, vil sikkerhetsfaktoren n til styrestammen bli:

$$n = \frac{\sigma_{yield}}{\sigma_{oppndd}} = \frac{250MPa}{59,4MPa} = 4,2 \tag{5.8}$$

Ut i fra dette resultatet kan det konkluderes med at styrestammen vil tåle de statiske belastningene de er utsatt for.

Styrestammen vil også bli utsatt for alternerende spenninger ved svinging til hver side. Det er derfor viktig å undersøke om den er motstandsdyktig mot utmattelse brudd. For stål legeringer med en flytegrense under 1400 MPa, er det normalt med en enduranse limit som ligger på mellom 40 og 60 prosent av strekkfastheten. Dette kan ses i formel 6-10 [4, s. 305]. Enduranse limit er en teoretisk grense for hvor høy spenning et materiale kan ligge under for å aldri gå til utmattelse brudd. Hvis den mest konservative verdien på 40 prosent benyttes vil enduranse limiten til A36 legeringen ligge på:

$$S_e = 0,4 \cdot S_{ut} = 0,4 \cdot 400MPa = 160MPa \tag{5.9}$$

Ettersom den høyeste oppnådde spenningen kun ligger på 59,4 MPa, kan det konkluderes med at styrestammen vil være svært motstandsdyktig mot utmattelse brudd.

Basert på resultatene i fra denne analysen, kan det konkluderes med at det er trygt at

steppermotor påfører styrestammen sitt maksimale dreiemoment på 72,56 Nm.

5.3.8 Konsepter for stag mellom rotordisker

Stagene bestemmer i stor grad hvordan fasongen til rotordiskene kan se ut. Det ble derfor bestemt å finne den beste løsningen for stagene før designet på rotordiskene ble ferdigstilt. Siden steppermotoren kan gi et moment på hele 72,56Nm må stagene være konstruert til å tåle disse kreftene godt. Stagene vil også være utsatt for både strekk og trykk krefter når rotordiskene roterer. Armlengden på rotordiskene fra rotasjonsaksen og ut til festepunktene til stagene er allerede bestemt fra testingen med prototypen. Det er derfor mulig å regne ut hvor store kreftene blir på stagene.

$$T = F \cdot a \Longrightarrow F = \frac{T}{a} \Longrightarrow F = \frac{72,56Nm}{0,1m} = 725,6N \tag{5.10}$$

Denne kraften på 725,6N vil bli fordelt på begge stagene slik at kraften som blir påført et stag blir 362,8N. Denne kraften vil bli påført som både som en strekk og trykk kraft avhengig av hvilken retning steppermotoren roterer. Det er også her viktig og ta hensyn til utmatting etter som de vil bli utsatt for varierende spenninger. Selv om steppermotoren kun klarer å påføre en kraft på 362,5N på hvert stag, kan det fortsatt oppstå situasjoner der stagene blir utsatt for høyere spenninger. For eksempel kan hjulene til ATV treffe på ujevnheter i veien som kan påføre slag på styresystemet. Det må derfor legges inn en god sikkerhetsmargin for å sikre at systemet kan tåle slike påkjenninger. Det er vanskelig og sette eksakte tall på slike uforutsette hendelser. Det bør derfor brukes en høy sikkerhetsfaktor under styrkeberegningene.

5.3.8.1 Stag - Konsept 1

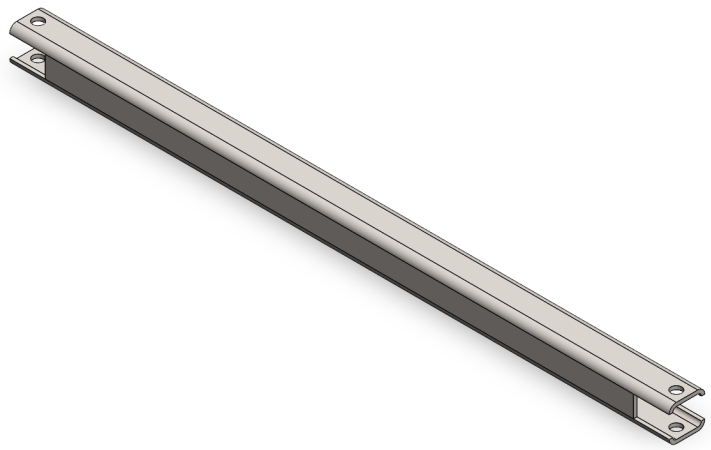
Dette enkle konseptet består av benytte to kvadratiske stag som festes til over og undersiden av rotordiskene. Dette er den samme løsningen som ble brukt på prototypen, bare denne gangen med kvadratisk tverrsnitt. Dette er ettersom staget bør være like motstandsdyktig mot bøy mot alle sideflatene. Fordelelen med dette konseptet er at man står mer fritt for å velge utforming for rotordisken da den aldri vil kollidere med rotordisken uansett hvor mye den roterer. Ulempen med dette konseptet er vanskeligheten med å få høy stivhet samtidig som vekten blir holdt lav. Dette er på grunn av at platenes nøytralakse vil ligge

langs midten av tykkelsen til hver av platene. Stagene vil dermed ha mye materiale langs nøytralaksen, som øker vekten samtidig som stivheten ikke øker i stor grad.

Figur 5.38: Stag - Konsept 1

5.3.8.2 Stag - Konsept 2

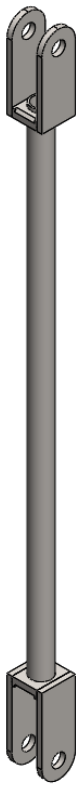
Dette konseptet går ut på å bruke et ekstrudert firkantrør der det skjæres vekk 2 av veggene i begge ender. Dette er for å kunne tre den inn på rotordiskene. Dette staget vil da kunne festes med bruk av bolter til rotordisken. Topplatene til røret vil på mange måter være lik som i konsept 1, men i dette tilfellet vil de sammenkobles ved hjelp av firkantrørets sideflater. Denne sammenkoblingen medfører at nøytralaksen nå vil bli langs midten av sideflaten, fremfor midten av topplatens tykkelse, slik den er i konsept 1. Resultatet av dette er at dette konseptet vil ha svært mye materiale langt i fra nøytralaksen, som medfører at staget vil bli veldig motstandsdyktig mot bøy og knekking. Ekstruderte rør er billig og enkelt å få tak i i forskjellig dimensjoner. De kommer også både i stål og aluminium.



Figur 5.39: Stag - Konsept 2

5.3.8.3 Stag - Konsept 3

Konsept 3 består av et sirkulært rør med en gaffel påmontert i hver ende for å festes til rotordiskene. Sirkulære rør er svært motstandsdyktig mot knekking og bøying, som gjør at vekten kan holdes lav. De er også effektiv mot bøying i alle retninger i motsetning til firkantrør som vil være sterkere mot bøy om den aksen der materialet er lengst unna nøytralasken og vil være svakere der materialet er næremere. Problemet med denne løsningen er å få festet gaflene godt nok til røret. Her er det også flere løsninger. Figur 5.40 viser et eksempel der gaflene er sveiset fast på utsiden av røret. En annen mulighet er å gjenge innsiden av røret for så å feste gaffelen med en bolt. Det kan også være mulig og lime fast en gjengeinnsats i vær ende av røret. Med dette kan man for eksempel bruke rør laget i karbonfiber eller glassfiber. Etter å ha undersøkt på nett virket det vanskelig å få tak i gafler som har de dimensjonene som trengs for å passe til dette. Disse må derfor produseres selv. Ved bruk av en plateknekker eller å sveise sammen flatstål vil det være mulig å lage disse selv.



Figur 5.40: Stag - Konsept 3

5.3.8.4 Valg av konsept for stag

Etter å ha vurdert for og imot de forskjellige alternativene falt valget til slutt på alternativ 3. Bakgrunnen for dette valget var den lave vekten og gode styrken til røret. Det ble forsøkt lage en u-brakett ved å bøye en stålplate med en plateknekker, men det viste seg å være vanskelig å lage 2 90-gradersvinkler tett på hverandre med det verktøyet som var tilgjengelig. Det ble derfor bestemt å lage u-brakettene ved å sveise sammen stålplater. Det ble også bestemt å gjenge røret for så å feste u-braketten med en bolt fremfor å sveise den fast til røret. Det største faremomentet med denne løsningen vil være å få sikret at sveiseforbindelsen i U-braketten blir sterk nok. For å kunne med sikkerhet si at denne delen vil tåle påkjenningene den vil bli utsatt for ble det bestemt å foreta en strekktest av en tilsvarende sveiseforbindelse for å få estimert en verdi på styrken til sveisen.

5.3.8.5 Valg av materiale for stag

For å få best mulig kraftoverføring og samtidig minimere faren for knekking er det viktig å bruke et materialet og et design med høy stivhet. Stål er svært sterkt og har også en

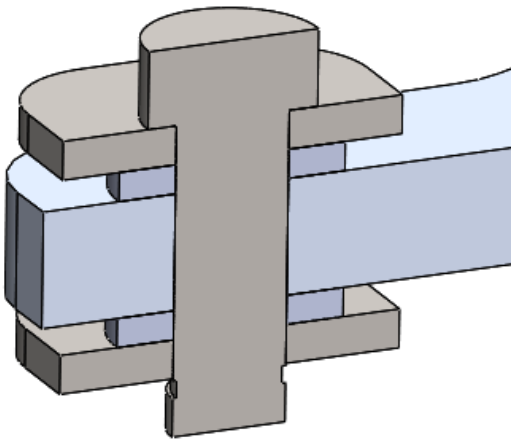
høy stivehet. Det ble derfor valgt å bruke dette materialet. Det er også tenkt å sviese u-braketten. Med tanke på dette er det også en fordel velge dette kontra aluminium som kan være problematisk å sveise. En annen mulighet ville vært å bruke rør i kompositt materiale som glassfiber eller karbonfiber. Dette vil krevd at det limes inn en gjengeinnsats i hver ende av kompositttrøret. En slik limforbindelse er svært sterk så dette kunnen vært et godt alternativ, men på grunn av den høye prisen på slike rør falt valget til slutt på å bruke stålrør.

5.3.9 Konsepter for bindeledd i rotordisk

Nå som designet på stagene er bestemt er det mulig å gå videre til å se på bindeleddet mellom rotordiskene og stagene. Her er det viktig med så lav friksjon som mulig for å unngå at krefter fra steppermotoren går tapt. Det er også her vurdert 3 forskjellige konsepter.

5.3.9.1 Bindeledd - Konsept 1: Rotor disk uten kulelager

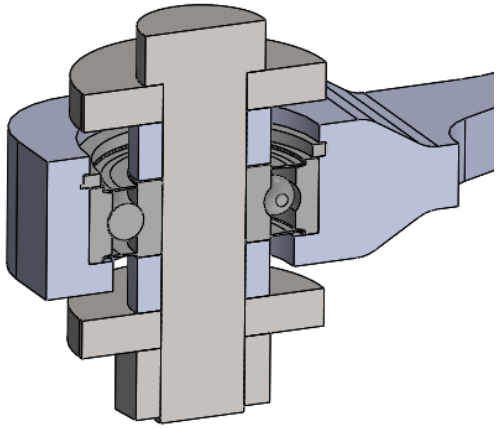
Siden testingen med prototypen i kryssfiner var svært vellykket tok vi utgangspunkt i denne utformingen. Hovedfordelen med denne utformingen er at det er lett å produsere og implementere. I Figur 5.41 kan man se hvordan en tilsvarende løsning kan se ut i metall. Her er det lagt til skiver for å minske friksjonen noe. Et av problemene med prototypen var at dersom boltene i bindeleddene ble skrudd til for hardt økte friksjonen i leddene betraktelig. Løsningen ble dermed og kun stramme boltene nok til at mutteren ikke løsnet. Friksjonen i bindeleddene var da på et akseptabelt nivå. Siden boltene aldri kunne tilstrammes skikkelig var det alltid en liten fare for at de løsnet. Det er derfor her valgt å låse boltene fast ved bruk av låsering. Boltene kan dermed sitte låst fast til rotordisken og rotere med lavere friksjon.



Figur 5.41: Tversnitt av bindeledd med bolt

5.3.9.2 Bindeledd - Konsept 2: Rotordisk med et senter montert kulelager

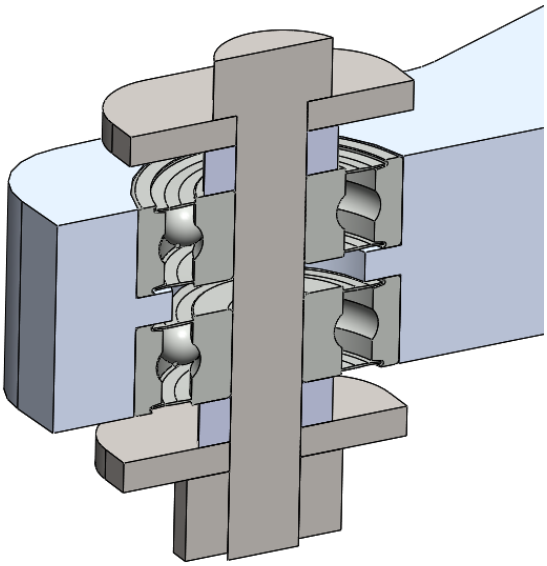
Dette konseptet går ut på å benytte et kuleledd for å minske friksjonen mer. Kulelageret ligger oppe på en liten hylle i bunn av innkuttet i rotor disken, og låses fast i toppen med en låsering i toppen. Både hyllen og låseringen er kun i kontakt med den ytre ringen i kulelageret, slik at den indre ringen fortsatt kan rotere fritt. Braketten festet til staget festes til kulelageret med en bolt som låses i bunn med en mutter. Det benyttes også to foringer på hver side av kulelageret, som sikrer at braketten ikke kommer i kontakt med rotordisken. Fordelen med dette konseptet er at det vil redusere friksjonen betraktelig. På den andre siden vil det også øke kompleksiteten og kostanden til rotordisken. Konseptet er vist i Figur 5.42.



Figur 5.42: Tversnitt av bindeledd med kulelager

5.3.9.3 Bindeledd - Konsept 3: Rotor disk med to kulelagre

Dette konseptet går ut på å benytte to kulelagre fremfor et. Her blir kulelagerene holdt på plass av en bolt som presser de inn mot en kant i midten av rotor disken. Her benyttes det også foringer for å hindre at u-braketten på staget er i kontakt med rotor disken. Bolten holdes på plass ved hjelp av en mutter. Fordelen med denne løsningen er at bolten har stor kontakt flate mot kulelagerene, som bidrar til høy stabilitet av bolten ved kraftoverføring. Ulempen med dette designet er at kulelagerene som benyttes må være svært små for å holde tykkelsen til rotor disken nede. Slike små kulelagre er og ofte ikke er dimensjonert for store radiale belastninger, selv ved fordeling over to lagre. Det er derfor nødvendig å øke tykkelsen til rotorskiven der kulelageret sitter, for å få plass til store nok kulelagre til å tåle belastningen.



Figur 5.43: Tversnitt av bindeledd med 2 kulelager

5.3.9.4 Valg av konsept for bindeledd i rotordisk

Etter å ha sett nærmere på de forskjellige konseptene endte valget på konsept 2: Rotordisk med et senter montert kulelager. Det ble først og fremst valgt å gå for en løsning med kulelager for å redusere friksjonen så mye som overhode mulig, slik at det blir minst mulig tap av krefter ved kraftoverføringen. Konsept 2 ble valgt fremfor konsept 3 ettersom det lettere lar seg gjennomføre med en tynnere rotorskive. En tynnere rotorskive vil gjøre delen lettere, billigere å produsere og øke energieffektiviteten til delen.

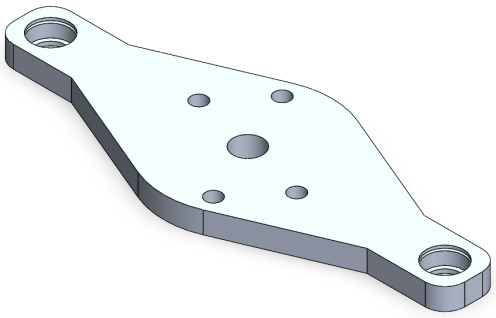
5.3.9.5 Valg av kulelager

Før kulelageret blir valgt er det viktig å fastslå hvilke spesifikasjoner som trengs for valgt konsept. Først og fremst trengs det et kulelager som kan ta opp store radielle krefter ved kraftoverføringen. Bolten som skal igjennom kulelageret må også være i stand til å tåle disse kreftene, så det er viktig å velge et kulelager med tillstrekkelig stor indre diameter. Ut i fra utregninger som kommer mer i detalj i avsnitt 5.3.11.4 ble det fastslått at en bolt med 6mm diameter vil være tilstrekkelig for å tåle disse kreftene. Det er også viktig å velge et kulelager med tilstrekkelig tykkelse og ytre diameter. Dette er viktig for å skape stor kontaktflate, som bidrar til stabilitet av bolt og kulelager. Samtidig vil dette også kreve en tykkere og tyngre rotor disk, så her burde det gjøres et kompromiss

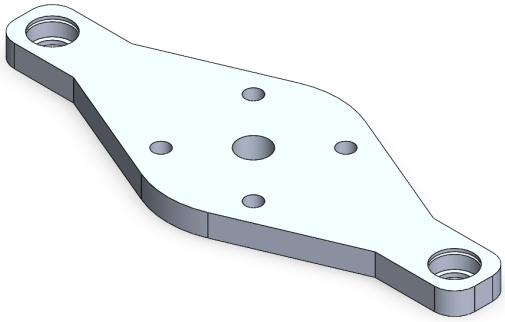
mellom vekt av rotor skive og tykkelse på kulelager. Ettersom ATV skal kunne benyttes i terrenget, vil kulelageret være utsatt for støv og andre urenheter som kan trenge inn i kulelageret. Det er derfor særdeles viktig med et kulelager med tette pakninger på begge sider. Det er vanlig med både metall og gummipakning for å unngå slike urenheter. Metall pakning på kulelageret har mindre friksjon ved rotasjon, men er på den andre siden ikke like tett som en gummi pakking. For denne applikasjonen vil denne forskjellen i friksjon være neglisjerbar, så det vil være mest hensiktsmessig med gummi pakking for å unngå mest mulig urenheter. Etter å ha vurdert de ulike punktene nevnt ovenfor, gikk valget på modellen 626-2RSH fra SKF. Dette kulelageret har 6mm indre diameter, 19mm ytre diameter og en tykkelse på 6mm. Den har også gummipakning .

5.3.10 Konsept for ytre geometri på rotordisk

Det er flere faktorer som spiller inn for hvordan den ytre geometrien til rotor diskene kan se ut. Steppermotoren og feste på styrestammen har forskjellige hullbilder, det må derfor lages to forskjellige rotordisker. På bakgrunn av valgt kulelager, settes det også minimums krav på hvordan utformingen rundt kulelageret må være. Dette er for å sikre at det er tilstrekkelig veggtykkelse rundt kulelageret. Ettersom stag konsept 3 skal benyttes, settes det også ytterligere begrensninger for hvordan ytre geometrien rundt kulelageret må utformes. Dette er for å sikre seg at staget ikke kolliderer inn i rotordiskene ved maksimalt vinkelutslag på ±45 grader. Det har også vært et stort fokus på å lage delene så lette som mulig. På bagrunn av disse faktorene har det vært lite rom for varierende form på ytre geometri av rotor disk. Det er derfor tatt utgangspunkt i en oval utforming, med innsnevrede endepunkter, der disse faktorene er tatt hensynn til. Denne utformingen med de to forskjellige hullbildene som skal benyttes kan ses på Figur 5.44 og 5.45. Det ble også bestemt lage delene av metall for sikre lang levetid på produktet.



Figur 5.44: Konsept 1: Oval rotor disk til styrestamme



Figur 5.45: Konsept 1: Oval rotordisk til stepper motor

5.3.10.1 Valg av materiale for rotordisk

Det har vært et stort fokus på å få Styresystemet så lett som mulig. Dette er generelt viktig på applikasjoner som skal bevege seg for å spare energi. Dette gjelder både for drivstoffforbruket til ATV ved kjøring, men også energien stepper motor bruker på akselerere rotorskivene ved svinging. Massetettheten til materiale vil dermed være en viktig faktor for å minimere vekten. Flytegrensen til materiale er også en viktig faktor for å spare vekt, ettersom et sterkt materiale kan dimensjoneres mindre, slik at vekten blir lavere. For å ende opp med et lettest mulig sluttprodukt vil dermed flytestyrken pr. vekt enhet være en viktig egenskap. Denne egenskapen kalles for spesifikk-flytestyrke. Aluminium eller stål vil være to gode alternativer, på grunn av den høye styrken og tilgjengeligheten de har.

La oss sammenligne en sterk stållegering med en sterk aluminiums legering. Et høykarbon stål som for eksempel AISI 1080 vil være et godt alternativ her, ettersom den har en svært høy flytegrense på 585 MPa. For aluminiums legeringen vil det være hensiktsmessig å velge en legering som er kunstig aldringsherdet for å oppnå høyest mulig flyte styrke. Et eksempel på dette er 6082 T6 aluminium, som har en flytestyrke på 250 MPa. Massetettheten til 6082 T6 ligger på 2700kg/m^3, imens Aisi 1080 stål ligger på 7850kg/m^3. Den spesifikke flytestyrken blir dermed:

$$Spesific yield_{AISI1080} = \frac{\sigma_{yield}}{\rho} = \frac{585Mpa}{7850kg/m^3} = 74522Pa \cdot m^3/kg \tag{5.11}$$

$$Spesificyield_{6082T6} = \frac{\sigma_{yield}}{\rho} = \frac{250Mpa}{2700/m^3} = 92592Pa \cdot m^3/kg \tag{5.12}$$

Ut i fra beregningene i fra formel 5.11 og 5.12 kan det konkluderes i at aluminiums legeringen har høyere spesifikk styrke. Det ble dermed besluttet å gå videre med aluminium som materiale til rotor diskene.

5.3.11 Dimensjonering

Delene som benyttes må tilpasses ettter hvor store krefter de er utsatt for. Dette er viktig for å sørge for at delene tåler belastningene de påføres, samtidig som de ikke er overdimensjonerte som resulterer i at delene får unødvendig høy vekt og volum. Dimensjoneringen i dette kapittelet vil bestå av mye manuelle håndberegninger, men også ved hjelp av simulasjons verktøy i SolidWorks.

5.3.11.1 Dimensjonering av rør

Det er ønskelig å ha stagene så lette som mulig. En lav vekt på stagene vil føre til lavere treghetsmoment som vil redusere belastningen på steppermotoren. Ellers er det alltid ønskelig med lavest mulig vekt på deler til motoriserte kjøretøy. Siden det ble bestemt å bruke bruke bolter for å feste u-braketten til røret må man ha minst 1,5mm veggtykkelse på røret for å ha nok gods til å lage gjenger i røret. Den minste rørdiameteren som var mulig å bestille hos vår lokale forhandler med denne veggtykkelsen var på 8mm. Dette røret er laget av St. 37.4 som er et vanlig lavkarbon konstruksjonstål med en flytegrense på ca. 230 MPa. Spenningen dette røret vil påføres ved en last på 362,8N kan enkelt regnes ut. Røret har et tversnittsareal på $30,63\text{mm}^2$. Spenningen i røret blir da:

$$\sigma = \frac{F}{A} = \frac{362,8N}{30.63mm^2} = 11,84MPa \tag{5.13}$$

Dette røret skal etter beregningene være mer en sterkt nok, men siden det ikke var mulig å få tak i mindre var dette det beste valget. Vekten på røret med denne dimensjonen kommer på 71,7g ifølge masseberegningen til SolidWorks. Dette ble vurdert til å være akseptabelt.

Knekking

Siden stagene vil være utsatt for en trykklast, og har en lang og smal fasong, er det viktig å ta hensyn til knekkking. Knekking kan oppstå brått og langt under spenningsnivåer som anses som høyt for et gitt materiale. Deler med et slankehtsforhold på mellom 50 og 200 anses for å ligge i faresonen. Dersom forholdet overstiger 200 er faren for knekking høy. For å finne slankehtsforholdet trenger man først å vite treghetsradien til fasongen på staget. Denne verdien er gitt ved Formel 5.14 der I_0 er tverrsnittets arealmoment og A er tverrsnittsarelet. Når man har denne verdien kan man finne slankhetsforholdet med Formel 5.15 der l_K er knekklengden til staget. Knekklengden er en verdi basert på innspenningstilfellene for et stag. Stagene til styresystemet er planlagt og festes med bolter i begge ender og får derfor 2 dreibare innspenninger. For dette tilfellet blir knekklengden lik den faktiske lengden på staget, [5], S. 336.

$$i = \sqrt{\frac{I_0}{A}} \tag{5.14}$$

$$\lambda = \frac{l_\mathrm{k}}{i} \tag{5.15}$$

Avstanden mellom rotordiskene er satt til å være 372mm. Stagen må derfor ha tilsvarende lengde til innspenningspunktene. L_K blir derfor 372mm. For vanlig konstruksjonsstål, St 37, med et slankhetsforhold på over 105 gjelder Eulers formel for elastisk knekking, se Formel 5.16.

$$F_K = \frac{\pi^2 E I_0}{(l_K)^2} \tag{5.16}$$

Her står F_K for den maksimale kraften staget kan holde før det er fare for knekking, E for E-modulen til materiale, I_0 er arealtreghetsmoment og L_K er knekklengden på staget.

Arealtreghetsmomentet for et sirkulært rør er gitt ved:

$$I_\mathrm{x} = I_\mathrm{y} = \frac{\pi}{4}({r_0}^4 - {r_1}^4) \tag{5.17}$$

I Formel 5.18 og 5.19 ser man utregningen av knekklasten for et sirkulært rør med 4mm ytre radius og 2,5 mm indre radius.

$$I_{\mathrm{x}} = I_{\mathrm{y}} = \frac{\pi}{4}(4^4 - 2,5^4) = 170mm^4 \tag{5.18}$$

$$F_K = \frac{\pi^2 \cdot 210000MPa \cdot 170mm^4}{(372mm)^2} = 2546N \tag{5.19}$$

Resultatet fra utregningene tilsier at det sirkulære stålrøret tåler en trykklast på 2546N. Dette er mer en det steppermotoren vil klare å påføre, man kan dermed konkludere med at faren for at røret knekker er minimal med disse dimensjonene.

5.3.11.2 Analyse av boltforbindelse mellom rør og u-brakett

Siden det skal brukes et rør med 5mm indre diameter må det lages M6 gjenger på røret og derfor en M6 bolt for feste U-braketten. Boltene som det er planlagt å bruke har en gjenget lengde på 20mm og har 12.9 gradering. Disse boltene har en bruddgrense på 1200 MPa og en bruddkraft på 24500N. Det er høyst usannsynlig at boltene vil oppleve noe i nærheten av slike krefter. Dersom dette skulle oppstå vil andre komponenter på styresystemet feile før dette. Det svakeste punktet for denne boltforbindelsen er gjengene i røret siden den består av et svakere materiale. Det er også her lite sannsynlig at disse gjengene kommer til å feile men det er foretatt en rask beregning på dette for å være på den sikkre siden.

Dersom gjengene i røret opplever en stor nok kraft kan det oppstå "shear pullout". Dette forårsakes av at skjærkraften mellom gjengene blir så stor at de blir ødelagt.

For å komme frem til den maksimale kraften som gjengene kan utsette for må man først estimere overflate arealet som er i kontakt med bolten. Formel 5.20 gir et estimat på skjærarealet til gjengene i røret.

$$A_{\mathrm{n}} = \pi n L_{\mathrm{e}} D_{\mathrm{s}} min(\frac{1}{2n} + 0.57735(D_{\mathrm{s}} min - E_{\mathrm{n}} max)) \tag{5.20}$$

L_{e} - Length of Thread Engagement

$D_s$min - Minimum major dia of external thread

$E_n$max - Maximum pitch dia of internal thread

n - 1/p - threads per unit (mm)

Denne formelen er hentet fra Fastenal sin raport "Screw Thread Design". [6].

Tykkelsen på braketten er på 2mm. Siden bolten har en lengde på 20mm vil det da være 18mm av gjengelengden som går inn i gjengene på røret. Setter man inn verdiene for en M6 bolt der L_e er 18mm får man:

$$A_n = \pi \cdot 1 \cdot 18mm \cdot 5.794(\frac{1}{2 \cdot 1} + 0.57735(5.794 - 5.324)) = 252,7mm^2 \tag{5.21}$$

Som nevnt i Avsnitt 5.3.11.4 kan man estimere flytespenning for skjærkraft til å ligge på 57,7% av flytspenningen for strekk på legeringer med lavt karboninnhold. St.37.4 som dette røret består av har en flytespenning på 230 MPa. Flytespenningen for skjærkraften blir da på 133 MPa for dette materialet.

Ut fra disse verdien kan man nå komme fram til maksimal tillat kraft på gjengen før de feiler. Setter man disse inn i formelen skjærkraft får man:

$$F = S_u \cdot A_n = 138MPa \cdot 252,7mm^2 = 34873N \tag{5.22}$$

Dette tilsvarer at bolten holder en vekt på 3555 kg. Man kan dermed konkludere med at denne boltforbindelsen vil være sterk nok.

5.3.11.3 Dimensjonering av sveist U-brakett

U-braketten som er tenkt å benyttes, vil være en egenprodusert del i A36 stål som sveises sammen. Grunnen for at dette lavkarbon stålet er benyttet her, er ettersom det er svært tilgjengelig og har svært god sveisbarhet. Delen vil bestå av to sideflater og en bunnplate med tykkelse på 2mm. Tynnere plater enn 2mm har ofte en tendens til å bøye seg på grunn av de termiske spenningene som oppstår på grunn av lokale forskjeller i temperatur. Disse forskjellene i temperatur medfører at metallet ekspanderer i varierende grad som medfører spenninger. I mange tilfeller er ikke tynnplater sterke nok til å stå i mot disse

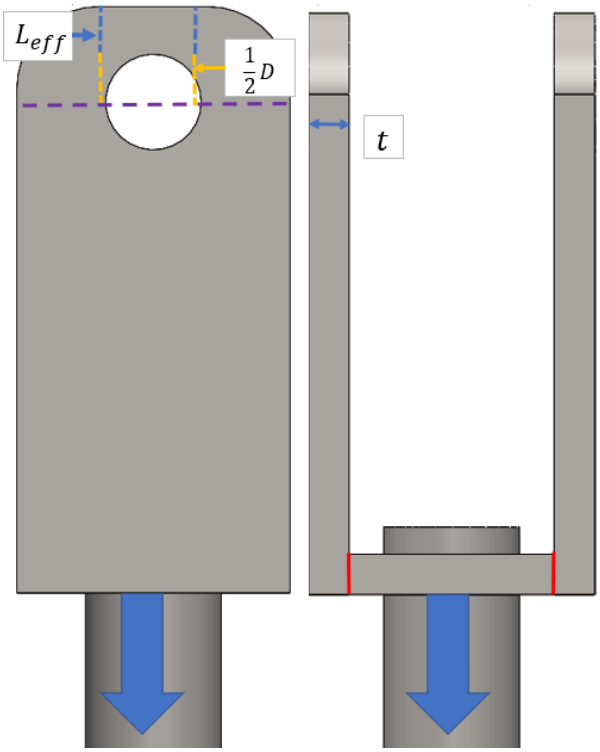
spenningene, slik at platen bøyes. Det er også vanskelig å få fullstendig gjennomtrenging av sveisen uten å sveise hull på platen. Dette er bakgrunnen for at 2mm plater er benyttet her.

Når braketten skal konstrueres, er det viktig å plassere hullet slik at det er nok forankringslengde opp til utsiden. Forankringslengden L effektiv er den korteste lengden fra hullet og opp til utsiden i den retningen kraften vil virke. Dette er illustrert med den blå stipplede linjen i Figur 5.46. I dette tilfellet vil kraften fordele seg over to skjærflater pr. side flate, slik at det blir 4 skjærflater totalt. Ettersom det benyttes A36 stål slik som i bolten, kan det benyttes samme tillatte skjærflytegrensen Ssy med en sikkerhetsfaktor på 3 som i bolten på 48,06 MPa (Formel 5.45). Nødvendig forankringslengde blir dermed:

$$\tau = \frac{V}{A} = \frac{V}{4 \cdot L_{eff} \cdot t} \Rightarrow L_{eff} = \frac{V}{4 \cdot \tau \cdot t} = \frac{362.8N}{4 \cdot 48,06MPa \cdot 2mm} = 0,94mm \tag{5.23}$$

Avstanden til senter av hullet fra toppen må dermed være minst:

$$Hullsenter = L_{eff} + \frac{D}{2} = 0.941mm + 3mm = 3.94mm \tag{5.24}$$



Figur 5.46: Utregning av forankringslengde på u-brakett

Det er også lurt å verifisere at hulltrykkspenningen til braketten ikke blir for stor med 2mm veggtykkelse. Ved praktisk beregning av arealet kraften fordeler seg over, er det vanlig å benytte diameter*platetykkelse fremfor den reele anleggsflaten. Dette er fordi kreftene langs hullflaten ikke fordeler seg jevnt, men følger en sinuskurve, som gir høye spenningsverdier langs midten, og lave verdier langs kantene. Hadde en kun benyttet den reele annleggsflaten til hullet uten sinusfordelingen, ville en dermed ha endt opp med lavere spenningsverdier enn det ville ha blitt i realiteten. Det er dermed vanlig å benytte diameter fremfor halve omkretsen i areal beregningen, ettersom dette gir en lavere verdi for arealet som gir en høyere verdi for hulltrykkspenningen.

$$\sigma_h = \frac{F}{A} = \frac{F}{2 \cdot D \cdot t} = \frac{362N}{2 \cdot 6mm \cdot 2mm} = 15,12MPa \tag{5.25}$$

Ettersom tilatt strekk/trykk spenning ligger på 83,3 MPa når en sikkerhetsfaktor på 3 benyttes, kan det konkluderes at braketten vil tåle hulltrykkspenningene.

Høyden fra bunn og opp til senter av hullet til U braketten ligger på 36mm. Denne høyden er tilpasset slik at rotor disken får plass til å rotere ±45 grader. Dette ble verifisert ved

hjelp av testing av sammenstillingen i SolidWorks.

Nødvendig bredde til braketten kan også dimensjoneres opp. Her vil det svakeste punktet være i tverrsnittet langs hullet markert med lilla stiplet linje i Figur 5.46, ettersom det er minst materiale til å holde igjen kreftene her. Hvis den tillatte strekk/trykk spenningen til materiale på 83,3 MPa benyttes får vi følgende resultater:

$$\sigma = \frac{F}{A} = \frac{F}{2 \cdot (B - D) \cdot t} \Rightarrow B = \frac{F}{2 \cdot t \cdot \sigma} + D = \frac{362,8N}{2 \cdot 2mm \cdot 83,3MPa} + 6mm = 7,09mm \tag{5.26}$$

Sveiseforbindelsen vil også være et kritisk punkt som blir påvirket av bredden til U-braketten. Denne forbindelsen er markert med rød strek i Figur 5.46. Langs disse to forbindelsene vil det oppstå følgende skjærspenninger:

$$\tau_{sveis} = \frac{F}{A} = \frac{F}{2 \cdot t \cdot B} = \frac{362.8N}{2 \cdot 2mm \cdot 7,09mm} = 12,79MPa \tag{5.27}$$

Dette resultatet viser at sveisen burde være mer enn sterk nok for å tåle belastningene. Det er verdt å merke at dette kun er en teoretisk verdi der sveisen ligger på linje med brakettens underflate, og det er 100% forbindelse i sveisen.

Etter å ha sett nærmere på målene i fra dimensjonerings utregningene, er det besluttet å gå noe opp i dimensjoner. Først og fremst er dette deler som skal være egenproduserte ved hjelp av manuelle produksjonsprosesser. Dette vil påvirke nøyaktigheten på målene på delene, som igjen vil påvirke styrken. Det kan derfor være lurt å ha litt ekstra materiale å gå på, slik at delen fortsatt vil ha tilstrekkelig styrke hvis målene ikke treffer rett på. Dimensjonerings verdiene er også teoretiske verdier som kun tar for seg de forventede belastnings scenarioer. Målene for bredden på U-braketten på 7,09mm vil for eksempel kun resultere i litt over en halv millimeter veggtykkelse på hver side på utkanten av hullet. I teorien vil dette være tilstrekkelig for kunne tåle strekk/trykk spenningene, men den vil på andre siden være svært lite motstandsdyktig mot slag og uventede belastnings scenarioer. De endelige målene for U-braketten kan ses i 2D tegning vedlegget.

5.3.11.4 Bøyemoment og skjærspenninger i bolt

Før bøyemomentet og skjærspenninger i bolten kan beregnes må situasjonen bolten utsettes for kartlegges. La oss ta utgangspunkt i det verst tenkelige scenarioet, der stepper motoren står og presser med maksimal kraft og hjulene til ATV låses av ytre hindringer. Ettersom det ikke er bevegelse på rotordiskene i en slik situasjon, vil rotordiskene være i statisk likevekt. Dette vil si at opplager kreftene kan beregnes med Newtons første lov. Denne situasjonen er illustrert i Figur 5.47. Her er det gjort en del forenklinger. Først og fremst er kreftene markert i blå som påføres bolten via U braketten satt som midtsentrerte punktlaster. I realiteten ville disse vært jevnt fordelte laster, men ettersom de største bøyspenningene kommer til å oppstå langs midten av bolten, vil dette ikke ha noe å si for resultatene. Reaksjonskreftene som oppstår i fra kulelageret som holder igjen er derimot satt som en jevnt fordelt last markert med de små oransje pilene. Dette vil også være en forenkling av situasjonen, ettersom lasten ikke vil fordeles helt jevnt etterhvert som bolten bøyer seg. I dette tilfellet ville bøyingen av bolten ha ført til at reaksjonskreftene langs 6mm og 12mm hadde blitt noe større enn langs midten (9mm). Tilnærmingen med en jevnt fordelt last vil allikevel være en grei tilnærming å ta, ettersom den er konservativ og vil sikre at bolten blir tillstrekkelig dimensjonert. I realiteten vil det også være en friksjonskraften mellom foringen og kulelageret, slik at foringene også kunne fungert litt som et jevnt fordelt opplager. Ettersom det blir tatt utgangspunkt i det verst tenkelige scenariet for bøyspenningene i bolten i dette eksempelet, vil denne friksjonskraften neglisjeres helt.

Før bøyemoment og skjærkrefter kan bli beregnet, må opplager kreftene regnes ut. Kraften pr stag ligger på 362.8N. Ettersom denne kraften fordeles over to flater vil kraften halveres igjen. Kreftene som påføres blir dermed:

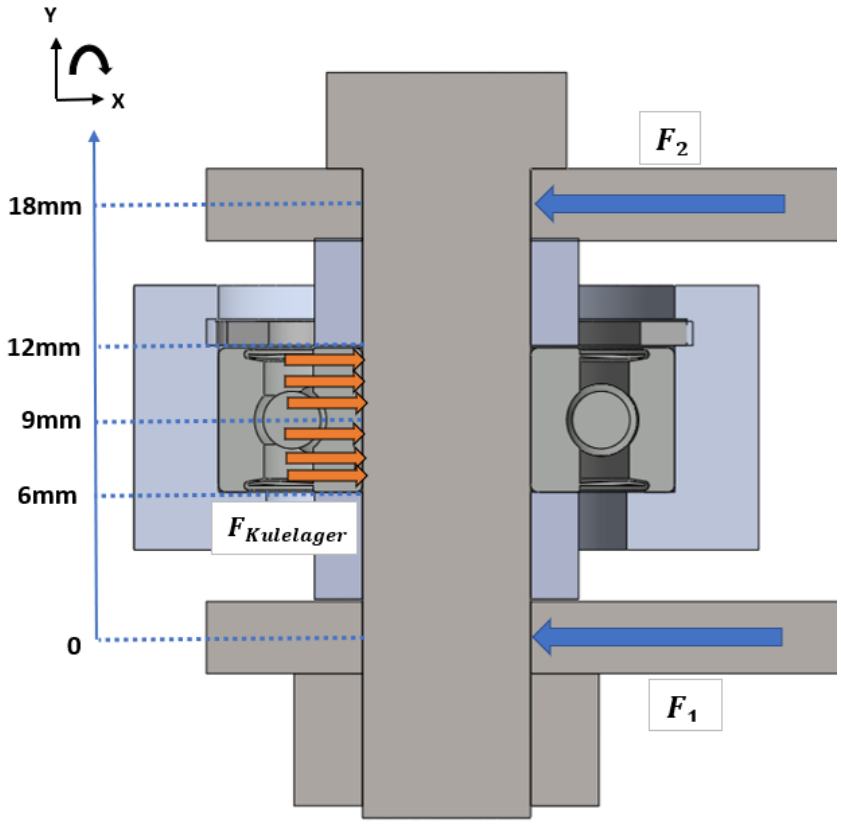
$$F1 = F2 = \frac{362.8N}{2} = 181.4N \tag{5.28}$$

Vi kan nå summere kreftene i X retning for å finne total kraften fra kulelageret:

$$\sum F_x = 0 \Rightarrow -F_1 - F_2 + F_{Kulelager} = 0 \Rightarrow F_{Kulelager} = F_1 + F_2 = 362.8N \tag{5.29}$$

Ettersom kraften fra kulelageret er en jevnt fordelt last som fordeles over kulelagerets tykkelse på 6mm, blir lasten pr mm:

$$F_{Kule/mm} = \frac{362.8N}{6mm} = 60.47N/mm \tag{5.30}$$



Figur 5.47: Krefter i bolt til rotordisk

Nå som opplager kreftene er kjente, kan bøyemomentet langs Y-aksen bli regnet ut. Her vil det bli sett på snitt langs boltens høyde fra 0<Y<18, der det alltid kun vil bli sett på kreftene på nedsiden av snittet. Det defineres at moment som skaper strekk på høyre side av bolt og trykk på venstre siden er positivt. Tillsvarende vil et moment som skaper strekk på venstre side av bolten og trykk på høyre side av bolten defineres som negativ:

$$M_{B0} = 0mm \cdot F_1 = 0Nmm \tag{5.31}$$

$$M_{B6} = 6mm \cdot F_1 = 6mm \cdot 181.4N = 1088.4Nmm \tag{5.32}$$

$$M_{B9} = 9mm \cdot F_1 - 3mm \cdot F_{Kule/mm} \cdot 1.5mm = 9mm \cdot 181.4N - 3mm \cdot 60.47N/mm \cdot 1.5mm = 1360.5Nmm \tag{5.33}$$

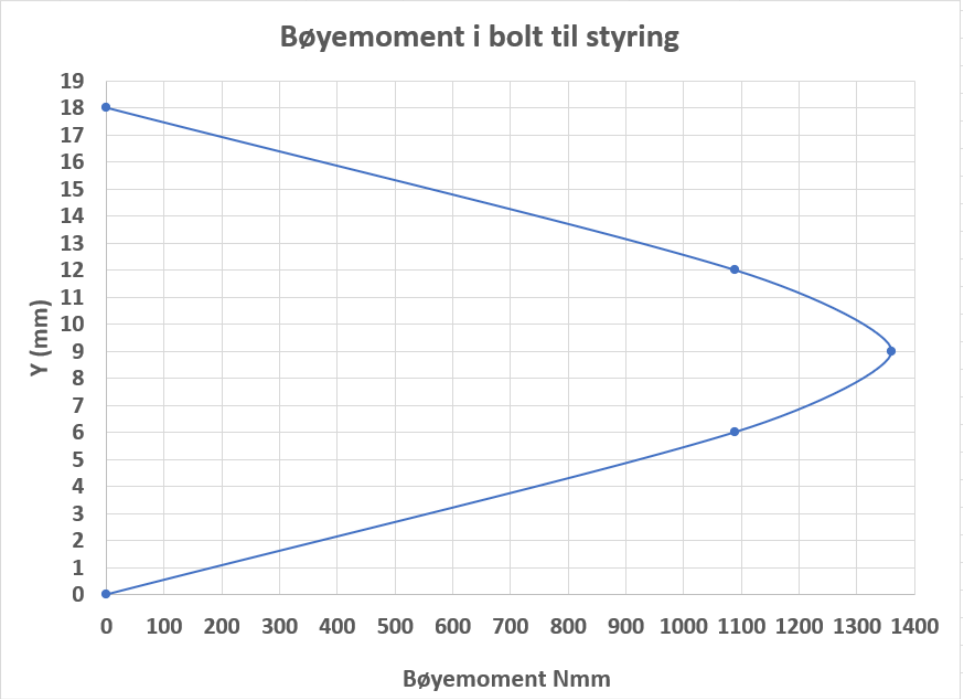
$$M_{B12} = 12mm \cdot F_1 - F_{Kulelager} \cdot 3mm = 12mm \cdot 181.4N - 362,8N \cdot 3mm = 1088.4Nmm \tag{5.34}$$

$$M_{B18} = 18mm \cdot F_1 - F_{Kulelager} \cdot 9mm = 18mm \cdot 181.4N - 362.8N \cdot 9mm = 0Nmm \tag{5.35}$$

Y (mm)	Bøyemoment (Nmm)
0	0
6	1088,4
9	1360,5
12	1088,4
18	0

Figur 5.48: Tabell med bøyemoment langs Y-aksen

I tabell 5.48 kan du se en oversikt over bøyemoment resultatene i de forskjellige snittene. Som du kan se markert i rødt i tabellen var det største bøyemomentet oppnådd i midten av kulelageret 9mm opp fra valgt nullpunkt på 1360,5 Nmm. Disse verdiene kan benyttes til å sette opp et bøyemoment diagram, som også illustrerer formen for hvordan bolten vil bøye seg. Dette kan ses i Figur 5.49.



Figur 5.49: Bøyemoment diagram for bolt i rotordisk

Nå som det største bøyemomentet er kartlagt, kan en dimensjonere hvor stor bolten må være for å tåle belastningen. I dette oppsettet er det særdels viktig at bolten ikke plastisk deformeres og beholder sin opprinnelige geometri. Dette er viktig fordi en plastisk deformasjon av bolten kan medføre at kreftene ikke blir overført kolineært med stagets senter akse. En slik kraftoverføring kan være dramatisk ettersom det kan fremprovosere knekking. I situasjoner der hjulene svinger med maksimal fart og hjulene plutselig blir hindret i å svinge mer, kan bolten også bli utsatt for store belastninger som følge av den kraftige akselerasjonen ved nedbremsingen. For å forsikre seg at bolten tåler disse belastningene uten å bli plastisk deformert vil det dermed bli tatt utgangspunkt i materialets flytegrense med en relativt høy sikkerhetsfaktor på 3. Det tas utgangspunkt i at bolten er laget i A36 lavkarbon stål med en flytegrense på minst 250 MPa. Den høyeste tilatte spenningen som kan oppnås med en sikkerhetsfaktor på 3 blir dermed:

$$\sigma_{Tillatt} = \frac{\sigma_{Flyt}}{3} = \frac{250MPa}{3} = 83,3MPa \tag{5.36}$$

Nå som høyeste tillatte spenning og maksimalt bøyemoment i bolten er kjent, gjenstår det kun å kartlegge motstandsmomentet til bolten sirkulære tverrsnitt. Motstandsmomentet W er gitt av det andre arealmomentet I og avstanden fra nøytralaksen og ut til ytre flaten

y:

$$W = \frac{I}{y} = \frac{\frac{\pi D^4}{64}}{\frac{D}{2}} = \frac{\pi D^3}{32} \tag{5.37}$$

Boltens nødvendige diameter for å motstå kreftene med en sikkerhetsfaktor på 3 kan nå bli regnet ut:

$$\sigma_{Tillatt} = \frac{Mb_{Max}}{W} = \frac{Mb_{Max}}{\frac{\pi D^3}{32}} \Rightarrow D = \sqrt[3]{\frac{32 \cdot Mb_{max}}{\pi \cdot \sigma_{Tilatt}}} = \sqrt[3]{\frac{32 \cdot 1360,5Nmm}{\pi \cdot 100MPa}} = 5.17mm \tag{5.38}$$

Ettersom det kun benyttes hele mm mål på indre diameter på kulelagerene, betyr dette at en må runde opp til 6mm for å sørge for at akslingen tåler belastningene.

For å være sikker på at en 6mm bolt vil være tilstrekkelig er det også viktig å se på skjærspenningene som oppstår i bolten. Her vil det også bli sett på snitt langs boltens høyde fra 0<Y<18, der det alltid kun vil bli sett på kreftene på nedsiden av snittet:

$$V_0 = -F_1 = -181,4N \tag{5.39}$$

$$V_6 = -F_1 = -181,4N \tag{5.40}$$

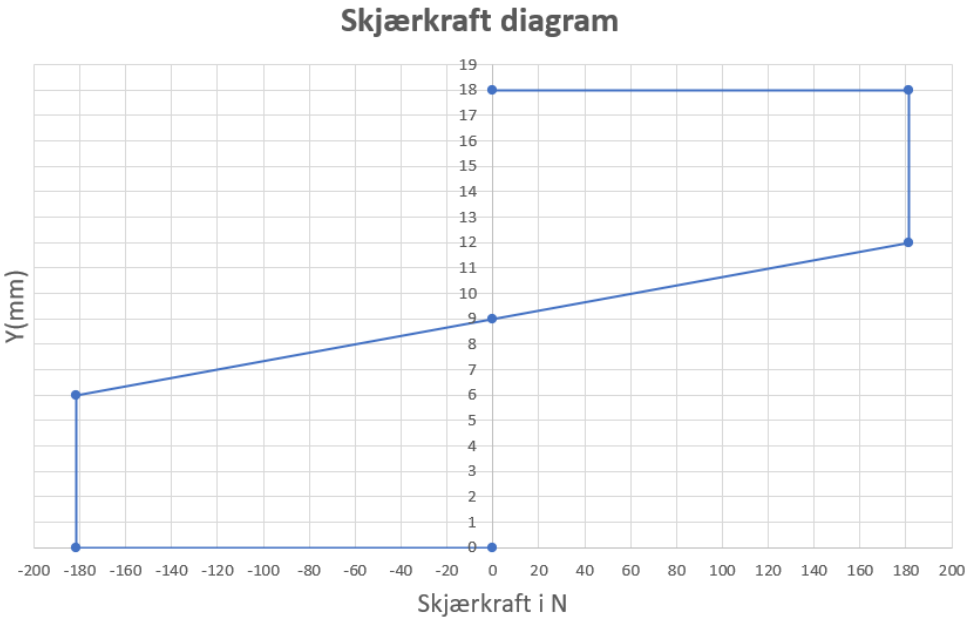
$$V_9 = -F_1 + 3 \cdot F_{Kule/mm} = -181,4N + 3mm \cdot 60,47N/mm = 0N \tag{5.41}$$

$$V_{12} = -F_1 + F_{Kulelager} = -181,4N + 362,8N = 181,4N \tag{5.42}$$

$$V_{18} = -F_1 + F_{Kulelager} = -181,4N + 362,8N = 181,4N \tag{5.43}$$

Y(mm)	Skjærkraft (N)
0	-181,4
6	-181,4
9	0
12	181,4
18	181,4

Figur 5.50: Tabell med skjærkraft langs Y-aksen



Figur 5.51: Skjærkraft diagram for bolt i rotordisk

Figur 5.50 viser en oversikt over resultatene. Disse resultatene er videre vist frem i et skjærkraft diagram som kan ses i Figur 5.51. Hvis en ser nøye etter kan det bli vist at skjærkraft diagrammet faktisk er den deriverte av bøyemoment diagrammet. Dette vil si at ekstremalpunktet til bøyemomentet (som i dette tilfellet er en maksimal verdi) vil oppnås der skjærkraften er 0, som skjer 9mm opp på Y-aksen.

Det kan også observeres at den høyeste absolutt verdien av skjærkreftene ligger på 181,4N fra 0-6mm og i fra 12-18mm opp på Y-aksen. Vi kan bruke denne verdien for maksimal skjærkraft videre for å regne ut maksimal skjærspenning:

$$\tau_{Max} = \frac{V_{Max}}{A} = \frac{V_{Max}}{\frac{\pi \cdot D^2}{4}} = \frac{181,4N}{\frac{\pi \cdot 6mm^2}{4}} = 6,41MPa \tag{5.44}$$

En mye brukt tilnærming på skjærflyte-spenningen til ductile stållegeringer slik som A36, er at den ligger på 57.7% av materialets flytegrense ved strekk. Denne tilnærmingen baserer seg på Von mises flytekriterium gitt ved formel 5-21 [4, s. 252]. Den tillatte skjærflytegrensen Ssy, med en sikkerhetsfaktor på 3 blir da:

$$S_{SY_{Tilatt}} = 0.577 * \sigma_{tilatt} = 0.577 * 83.33MPa = 48.06MPa \tag{5.45}$$

Ettersom den tillatte skjærflyte-spenningen ligger på 48.06 MPa og skjærspenningene i bolten kun er på 6.41 MPa, kan det konkluderes at skjærspenningen i bolten vil være helt uproblematisk.

Ettersom bolten vil bli belastet med sykliske krefter, er det også viktig å dimensjonere bolten på en måte som også gjør den motstandsdyktig mot utmattelsesbrudd. Her tar vi utgangspunkt i samme enduranse limit som tideligere i Formel 5.9 på 40% av strekkfastheten.

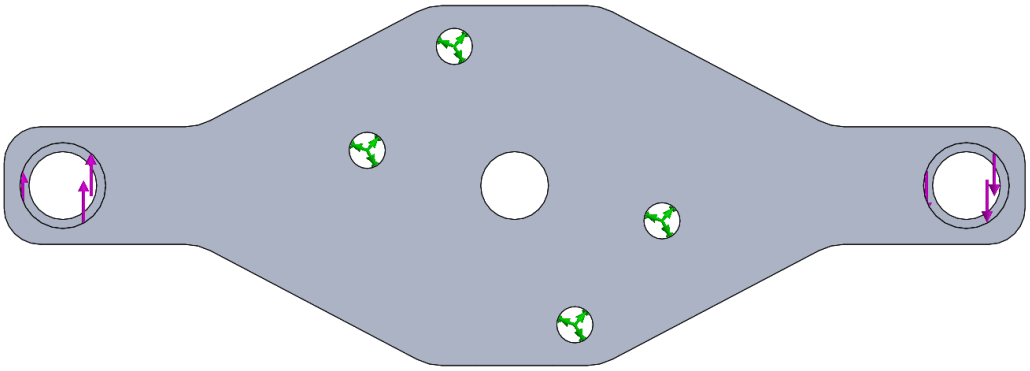
$$S'_e = 0.4 \cdot 400MPa = 160MPa \tag{5.46}$$

Den største oppnådde bøy spenning ved bruk av 5,17mm diameter bolt ligger på 83,3 MPa, og spenningene blir enda lavere enn dette når 6mm diameter benyttes. På bakgrunn av dette kan det konkluderes med at spenningen i bolten vil ligge langt under enduranse limiten på 160 MPa. Dette vil si at bolten vil være svært motstandsdyktig mot utmattelsesbrudd.

5.3.11.5 Dimensjonering og vekt optimalisering av rotordisk

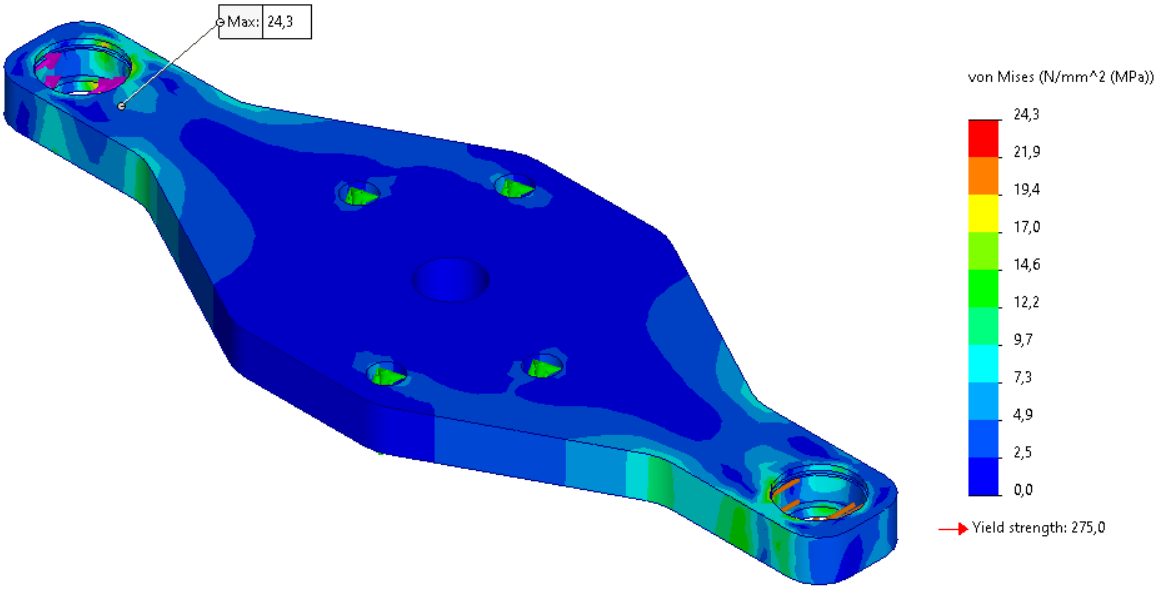
Etter at den ytre geometrien på rotordisken var bestemt, ble det kjørt statiske FEM analyser på delen. I analysene ble det brukt 'Fixed Hinge' som låsinger i bolthullene for å simulere boltforbindelsen. For å simulere kraften fra kulelagerene på rotordisken ble det brukt 'Bearing Load' som gir en sinusfordelt kraftoverføring i hullet. I Figur 5.52 ser man denne lasten markert med lilla piler i hullene der kulelagerene vil ligge. Kraftretningen er her 90 grader på rotordisken. Analysen ble også kjørt med lastene påført med en vinkel på 45 grader som vil simulere situasjonen ved maksimalt rattutsalg. Da dette er en nokså

liten del ble det brukt fint 'Mesh' på hele delen uten at dette gikk utover kjøretiden på analysen.



Figur 5.52: Laster og Låsinger

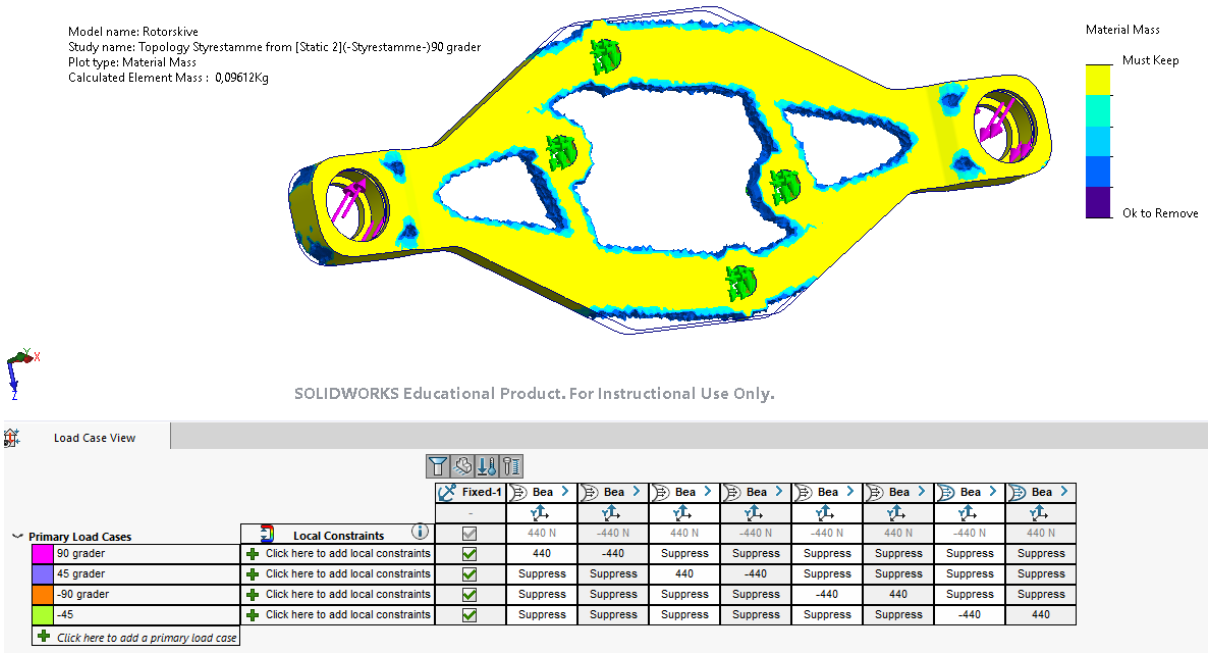
Ut ifra resultatene fra disse analysene ble det raskt kartlagt at spenningene var svært lave slik delen var utformet nå. Dette kan ses i Figur 5.53, der resultatet ble en maksimal spenning på 24,3 MPa. Dette gir rom for å fjerne materiale og optimalisere vekten til delen. For å få en bedre forståelse av hvordan kreftene blir overført fra kulelagrene og inn i rotordiskene ble det kjørt topologi analyser i SolidWorks Simulations. Denne analysen genererer en optimal form innenfor yttergeometrien til delen som blir analysert basert på en statisk analyse. Ut ifra denne genererte formen kan man se hvor man kan fjerne unødvendig materiale.



Figur 5.53: Statisk analyse av rotordisk

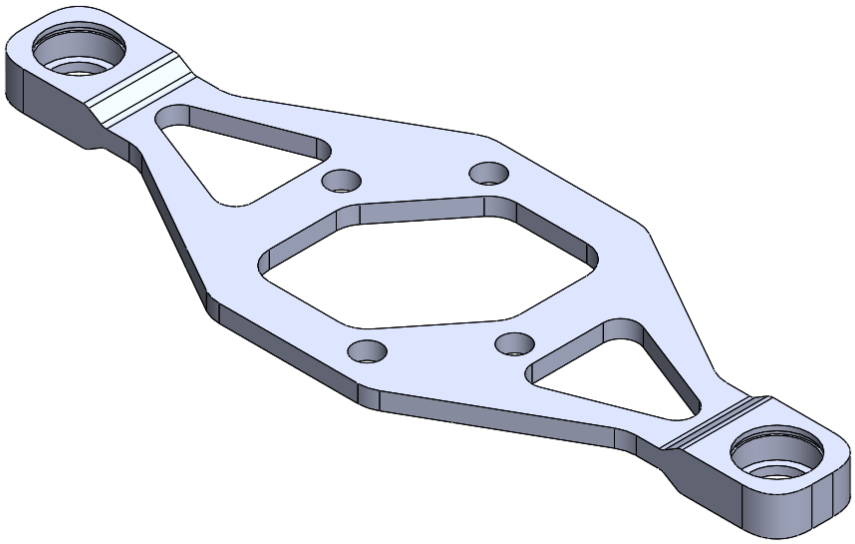
I Figur 5.54 kan man se resultatet av topologistudien av rotordisken til styrestammen. Nederst i figuren ser man en tabell der alle de forskjellige last scenarioene som delen blir utsatt for er oppført. Scenarioene som er tatt i betraktning er når stagene står vinkelrett og når de står 45 grader på rotordiskene. Dette gir da 4 last scenarioer når man inkluderer begge rotasjonsretninger. Under en slik analyse vil da programmet iterere en rekke ganger frem til den kommer frem til en optimal fasong for å stå imot disse kreftene. Siden topologi studien ikke tar hensyn til produksjonsprosesser, står man igjen med en del som ikke er mulig å lage med vanlige maskineringsmetoder. Det man derimot kan gjøre, er å bruke resultatet fra analysen som en mal for så å design en tilnærmet lik del som lar seg produsere med konvensjonelle metoder. I Figur 5.55 kan man se den endelig rotordisken til styrestammen etter å ha brukt denne fremgangsmetoden.

Som man kan se i figurene, er ikke resultatet fra topologi analysen fulgt helt nøyaktig. Det er planlagt og maskinere disse delene i en CNC maskin. Man må derfor ta hensyn til hvordan delen skal oppspennes i maskinen. De ytterste flatene er beholdt rette for at man kan bruke disse for å enklere holde delen på plass under maskinering. Disse flatene forenkler også inspeksjonsprosessen på delen da disse flatene kan brukes som referansepunkter (Datum) under måling. I Vedlegg A42 kan man se 2D-tegningen som ble laget for denne delen.



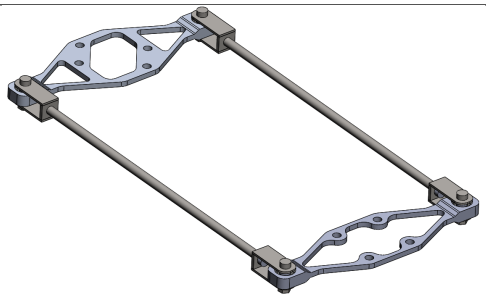
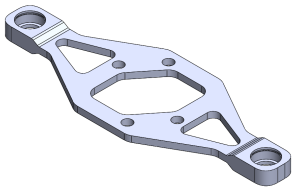
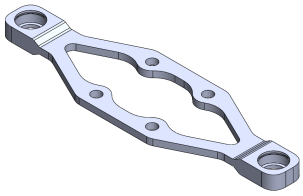
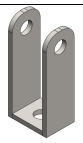
Primary Load Cases	Local Constraints	Fixed-1	Bea	Bea	Bea	Bea	Bea	Bea	Bea	Bea
		-								
			440 N	-440 N	440 N	-440 N	-440 N	440 N	-440 N	440 N
90 grader	Click here to add local constraints	☑	440	-440	Suppress	Suppress	Suppress	Suppress	Suppress	Suppress
45 grader	Click here to add local constraints	☑	Suppress	Suppress	440	-440	Suppress	Suppress	Suppress	Suppress
-90 grader	Click here to add local constraints	☑	Suppress	Suppress	Suppress	Suppress	-440	440	Suppress	Suppress
-45	Click here to add local constraints	☑	Suppress	Suppress	Suppress	Suppress	Suppress	Suppress	-440	440
Click here to add a primary load case										

Figur 5.54: Topologistudie

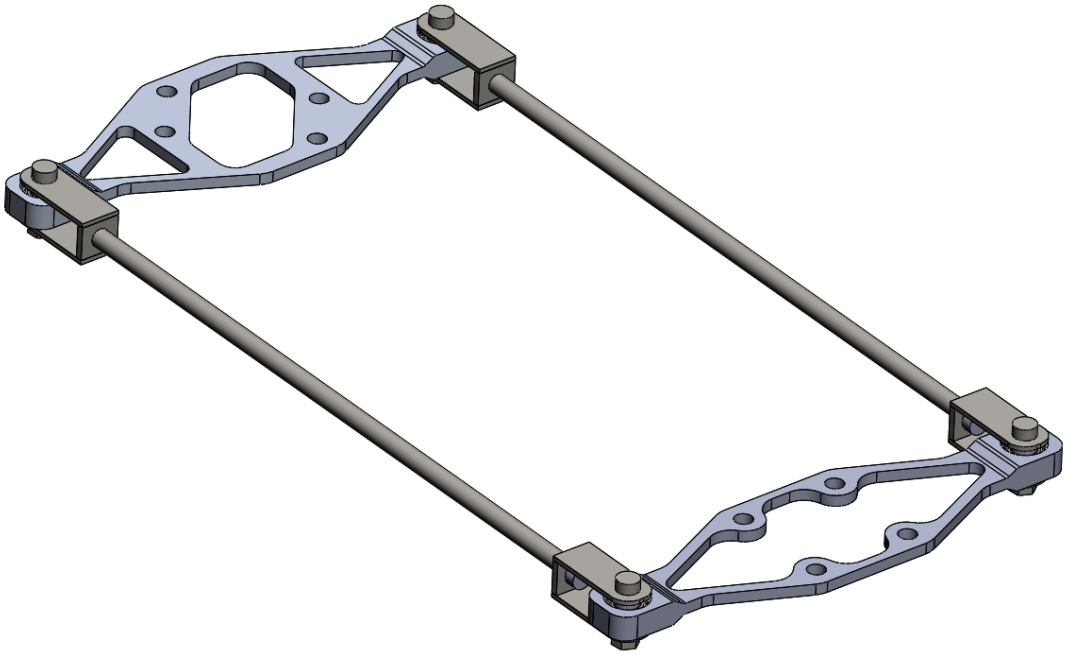


Figur 5.55: Rotordisk til styrestamme

5.3.12 Oversikt over deler i systemet

Komponent	Modell	Materiale
Sammenstilling rotor disker med stag		A36 stål og 6082 T6 aluminium Bearing steel
Rotordisk styrestamme		6082 T6 aluminium
Rotordisk steppermotor		6082 T6 aluminium
4xEgenprodusert sveist U-brakett		A36 stål
2xStag mellom rotordisker		A36 stål
4xLåsering		A36 stål
4xSKF 626 2RSH kulelager		Bearing steel
4xEgenprodusert dreid bolt til kulelager		A36 stål
8xEgenprodusert dreid foring til kulelager		A36 stål

Tabell 5.2: Oversikt over de forskjellige komponentene i girstagets assembly



Figur 5.56: Sammenstilling av styresystemet

5.3.13 Produksjon av stag

5.3.13.1 Rør

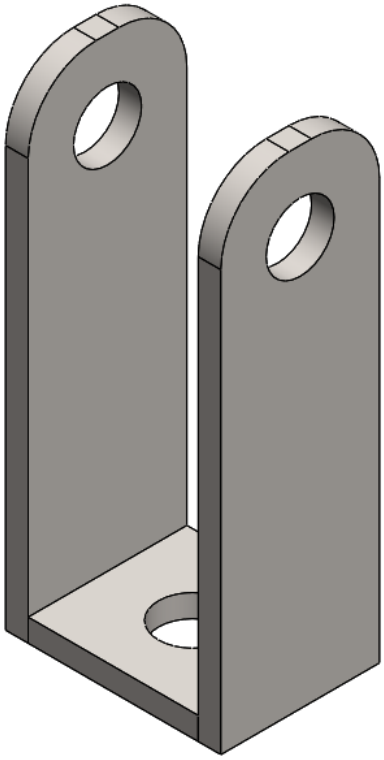
For å feste u-braketten til røret må det gjenges i vær ende etter at rørene er kappet til riktig lengde. Dette ble gjort med bruk av en gjengetapp for M6 gjenger.

5.3.13.2 U-brakett

Det skal lages 4 u-braketter som skal boltes fast til rørene. Platene som skal sveises sammen er kun 2mm tykke. Som vist i Figur 5.57 så består braketten av 3 plater. Bunnplaten og 2 sidevegger. Det er viktig at disse sideveggene står 90 grader på bunnplaten for at delen kan monteres korekt til rotordisken. Sveising kan rask føre til restspenninger i materialet som kan deformere delen. Det ble derfor laget en liten ”jig” som holder disse platene på plass og hindrer sideveggene i å bøye seg under og etter sveisingen. Denne ”jig’en” fungerte svært godt. Ved å skru fast denne i en skrustikke ble platene låst fast i korrekt posisjon uten mulighet for å deformere seg. Dette er gunstig med tanke på korrekt vinkel på sideveggene, men en for høy innspenningsgrad kan gi problemer med restspenninger i platene som videre kan danne varmesprekker. Sveisene vil derfor bli inspisert ved bruk av penetrant testing.

Siden platene er relativt tynne og små, er TIG sveising foretrukket, da denne sveisemetoden egner seg best i slike situasjoner. Siden denne metoden gir bedre presisjon og kontroll over dybde på sveis i forholdt til MIG sveising.

I sveiseforbindelsen ble det anvendt buttsveiser. Det ble vurdert å lage v-fuger, men man konkluderte til slutt at dette ikke var nødvendig på så tynne plater. Sveisetråden som ble brukt under sveising er "OK Tigrod 12.64" fra ESAB. Denne brukes for sveising av lav karbon innholdige stål legeringer slik som i dette tilfellet.



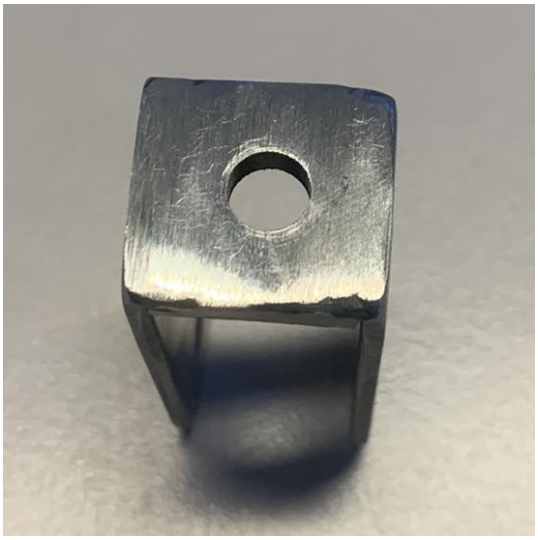
Figur 5.57: U-brakett

5.3.13.3 Testing av U-brakett

Ved nøyere inspeksjon av sveisene, ble det observert noen små porer langs sveisens overflate. Dette er et resultat av at det ikke har vært nok dekkgass tilstede, som kan skyve oksygenet bort i fra sveisen. Porene kan fungere som en inkubasjon, som kan medføre starten av en sprekk som vokser ved utmattelse. Ettersom braketten vil bli utsatt for både strekk og trykk i sykluser, vil braketten være svært utsatt for at slike sprekker vokser videre. Formel 5.27 viser at det vil være svært lave spenninger selv når sveisen ligger på linje med underflaten. Det kan dermed argumenteres for at det er et større faremoment med utmattelse enn statisk belastning. For å unngå faren for utmattelse i størst mulig grad, ble

det dermed bestemt at sveiselarven burde slipes ned til den var på linje med underflaten av U-braketten. En av U-brakettene før og etter sliping kan ses på Figur5.58 og 5.59.

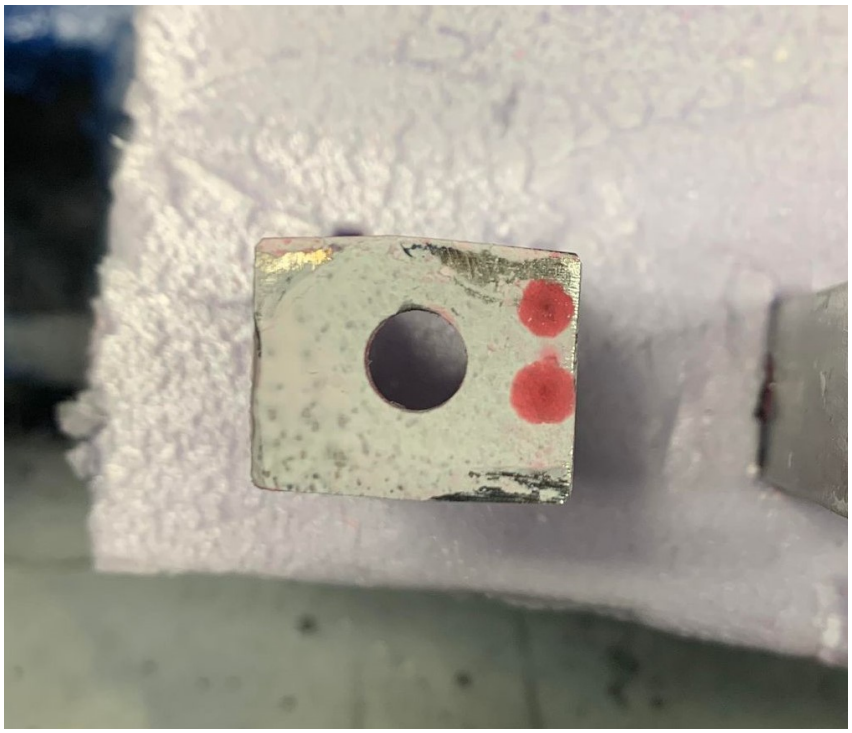
Figur 5.58: Sveist U-brakett før sliping



Figur 5.59: Sveist U-brakett etter sliping

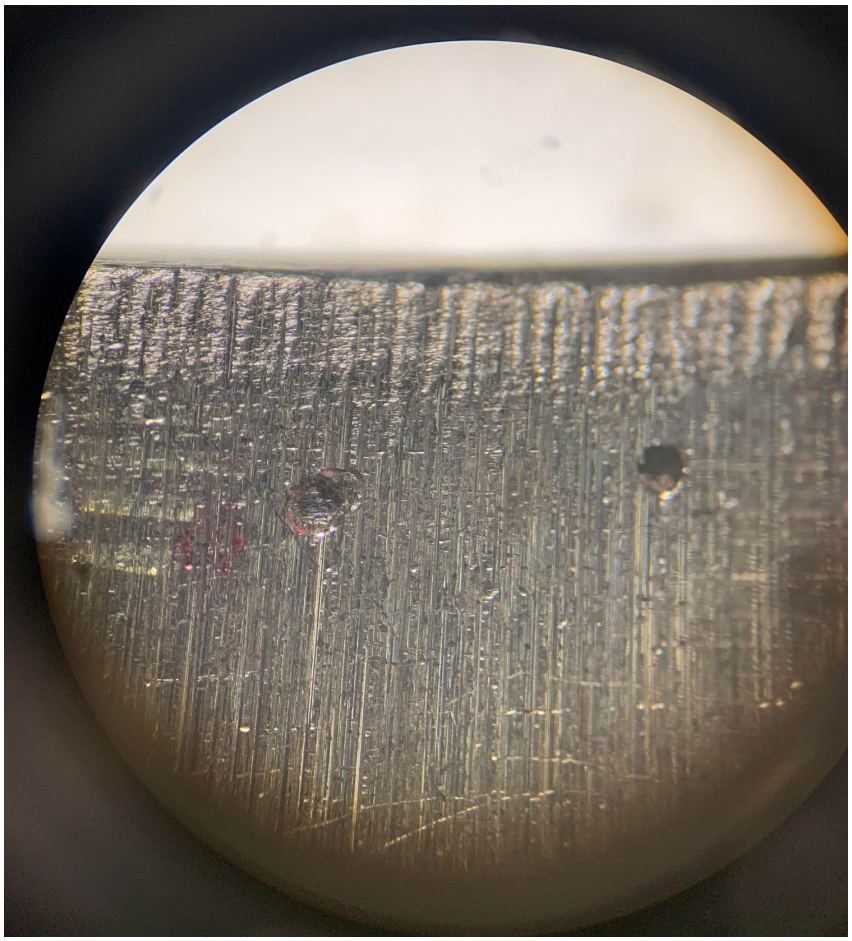
Sprekker og porer kan i noen tilfeller være svært vanskelig å se med det blotte øyet. Det ble derfor bestemt å gjennomføre en veske penetrant test på sveisene til brakettene for å undersøke om det fortsatt er porer/sprekker igjen etter brakettene har blitt pusset ned. Dette er en NDT (Non destructive test) metode som er vanlig å bruke på slike svieser spesielt når det er deler som er utsatt for utmatting. Her vil en sprekk raskt kunne føre til at en del feiler.

Første steg er å vaske delen helt ren. Deretter påføres en penetrant. Dette er en væske med svært lav viskositet som kan trenge ned i selv de minste sprekker som ikke er synlig for de blotte øyet. Når penetranten har fått tid til å trekke inn i eventuelle sprekker tørkes den overflødige væsken vekk slik at kun det som har trengt ned i sprekker og porer gjenstår. Deretter påføres en 'Developer'. Denne legges på som et tynt hvit skum som utnytter capillær effekten til å suge til seg penetranten som ligger igjen i sprekkene. Penetranten brukt her har en rød farge som gjør den lett synlig når den blir absorbert av 'developer'. I Figur 5.60 ser man et bilde fra testen. Her er det da ikke tegn til sprekker, men man ser tydelig at det er porer i sveisen.



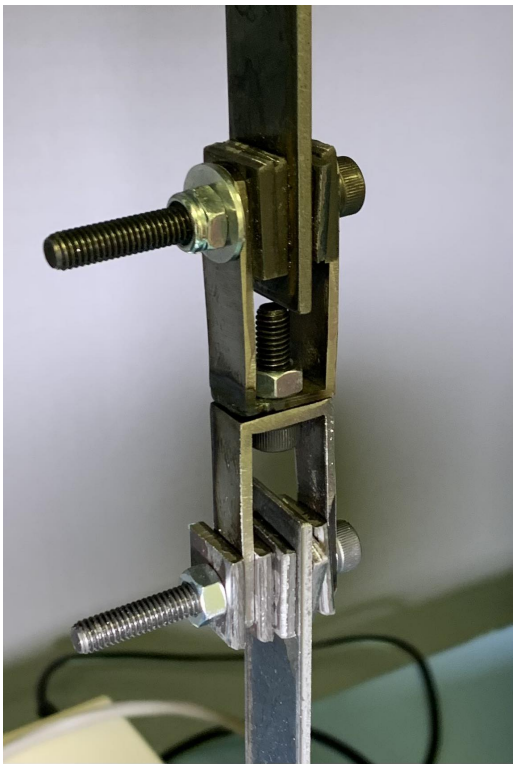
Figur 5.60: Væske penetrant testing av u-brakett

Etter å ha vasket delen ble område så undersøkt næremere med mikroskop, se Figur 5.61. Her ble porene kartlagt. De ble anslått til å være dype nok til at det kunne øke faren for utmatting. Overflaten ble derfor pusset ned til porene ikke lenger var synlige.

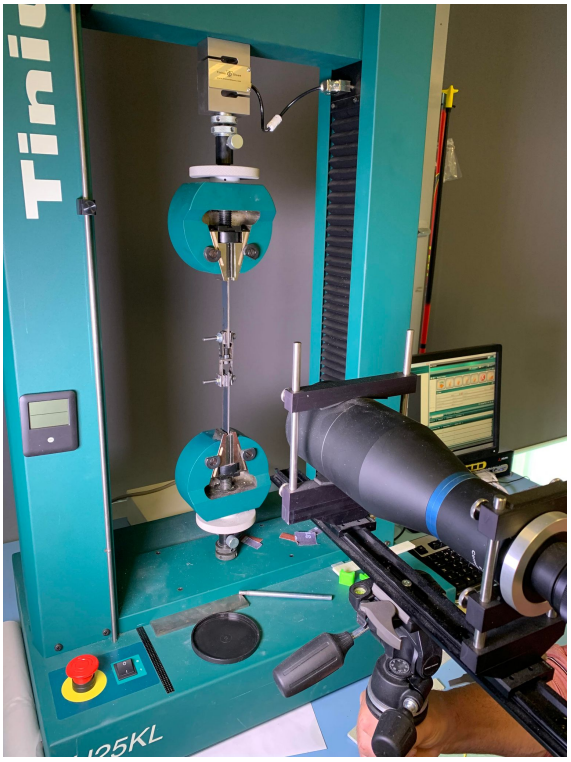


Figur 5.61: Porer i u-brakett under mikroskop

Det ble også gjennomført en destruktiv strekktest av brakettene. Dette er viktig for å verifisere at sveiseforbindelsen har tillstrekkelig styrke. For å få testet ut dette ble U-brakettene skrudd sammen som vist på Figur 5.62. Når disse blir dratt fra hverandre med strekktestmaskinen, vil boltforbindelsen holde igjen brakettene på samme måte som når de er montert på røret. På Figur 5.63 kan du se prøvestykke montert inne i maskinen.



Figur 5.62: Strekktest oppsett

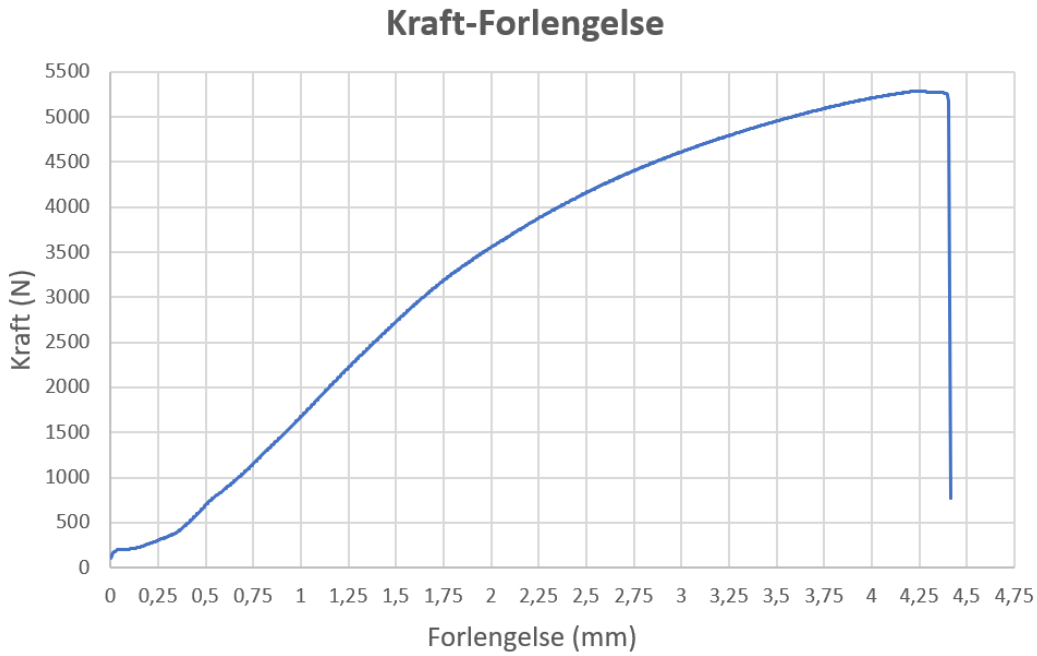


Figur 5.63: Prøvestykke i strekktest maskin

Testen resulterte i at braketten kom til brudd ved en påsatt kraft på 5276N. Bruddet oppsto i sveiseforbindelsen som følge av de store skjærspenningene som oppstår langs sveisen. Et nærbilde av dette bruddet kan ses i Figur 5.64.

Figur 5.64: Brudd ved strekk test

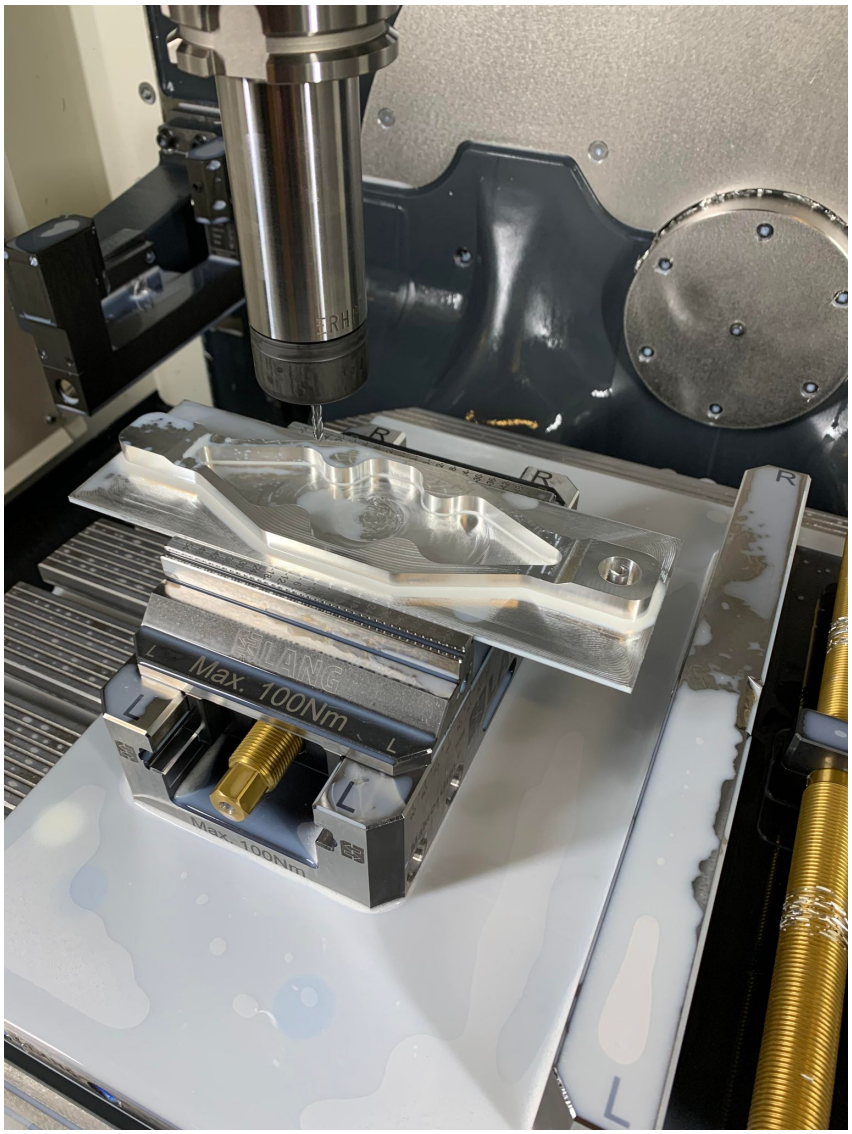
Når strekktesten gjennomføres vil kraften maskinen påfører prøvestykket og lengde endringen bli registrert. Lengde endringen registreres optisk ved hjelp av et kamera mellom to valgte punkter på braketten. De valgte punktene i utført test kan ses på Figur 5.64 markert med røde kryss. Verdiene for denne lengdeendringen og påsatt kraft, kan bli benyttet til å plotte en kraft-forlengelse kurve. Dette kan ses i Figur 5.65. Det er verdt å merke at forlengelsen ikke vil gi verdier som representerer selve sveisen i en av brakettene på en god måte. Dette er på grunn av at det benyttes to braketter fremfor kun en, samtidig som det også oppstod store deformasjoner på grunn av bøy i bunnplaten. Testen var uansett svært vellykket, ettersom det ga god informasjon om hvor stor kraft U-braketten tåler før den går til brudd. Etter som U-brakettene kun skal tåle en strekk kraft på 362N, kan det konkluderes at sveiseforbindelsen vil være mer enn sterk nok.



Figur 5.65: Kraft-Forlengelse diagram

5.3.14 Produksjon av rotordisker

Det ble tidlig bestemt at det ville være gunstig å få rotordiskene maskinert i en CNC maskin. Dette ville gjøre det enklere å få en god pasning mellom rotordisk og kulelager. Det ville også tillate mer kompleks utforming en det som lar seg gjøre dersom delene skulle vært maskinert selv i manuell fresemaskin. Oppdraget med å maskinere disse delene ble sendt til K-tech. I Figur 5.66 ser man rotordisken oppspent i CNC maskin.



Figur 5.66: Rotordisk under maskinering i CNC maskin

5.3.15 Overflatebehandling av deler

For å forbedre korrosjonbestanden på ståldelene ble det bestemt å lakkere disse. Denne prosessen gikk ut på å pusse overflaten med fint sandpapir slik at oksidlag og urenheter blir fjernet og overflaten får en bedre heftelse når den lakkeres . Deretter ble det påført rusthindrende grunning og sort lakk. I Figur 5.67 ser man stagene etter ferdig lakkering.

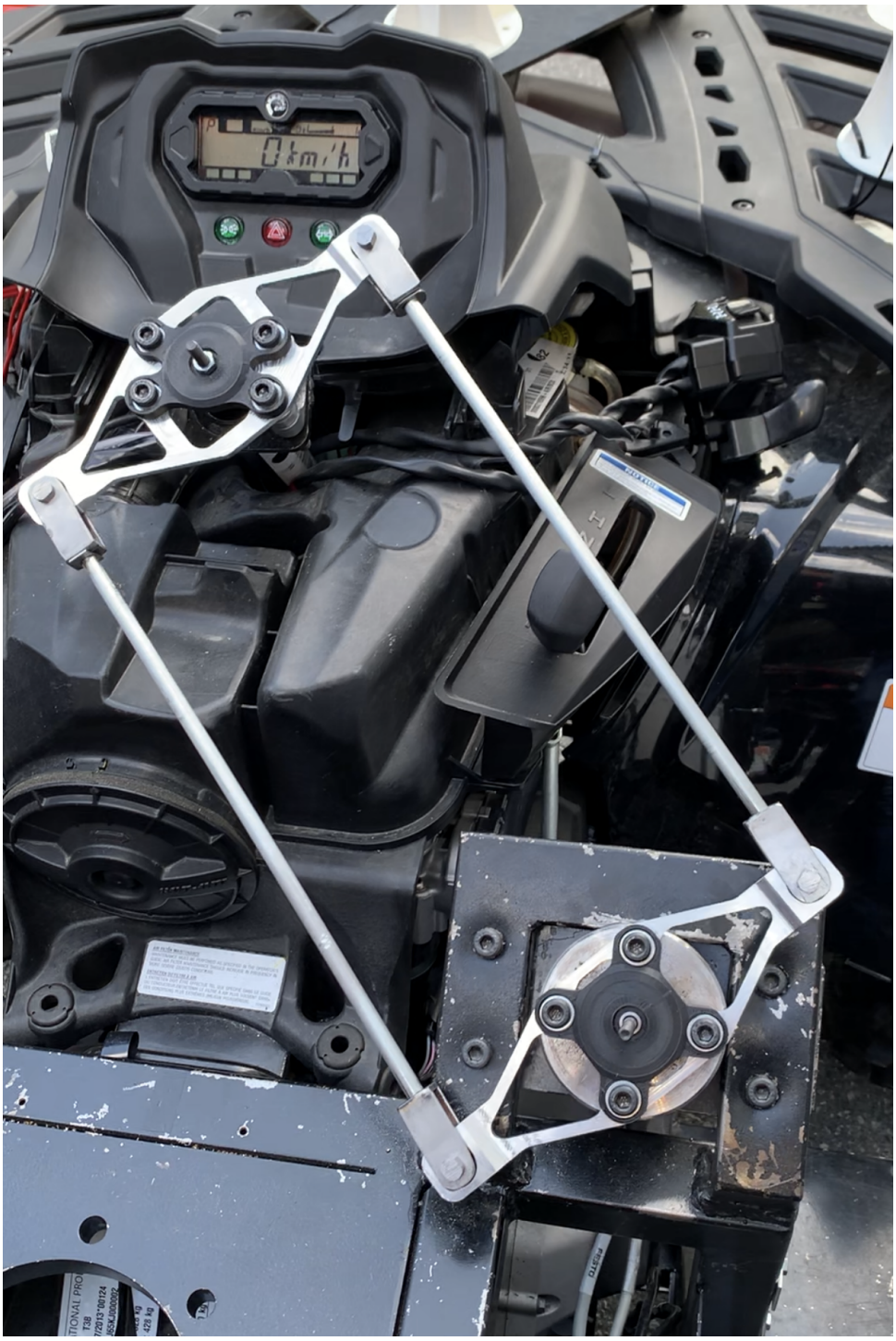
Figur 5.67: Lakkert stag

5.3.16 Resultat av nytt styresystem

Arbeidet med nytt og bedre styresystem har vært vellykket. Hovedfokuset har vært å redusere vekt, øke effektiviteten av kraftoverføring fra steppermotor og redusere bøyspenninger på styrestammen. Som vist i Figur 5.68 og 5.69 har man klart og redusere vekten betraktelig. Fra en opprinnelig vekt på 1,8 kg til 0,5kg. Dette tilsvarer en reduksjon på 72,2 %. Styrestammen påføres nå ingen bøyspenninger, og steppermotoren kan overføre kraften lettere og mer presis. Det er vanskelig å sette konkrete tall på hva endringene har ført til for de sistnevnte, men styringen har blitt merkbart bedre undre kjøring med styring fra fjernkontroll.

Figur 5.68: Vekt på det gamle styresystemet

Figur 5.69: Vekt på det nye systemet



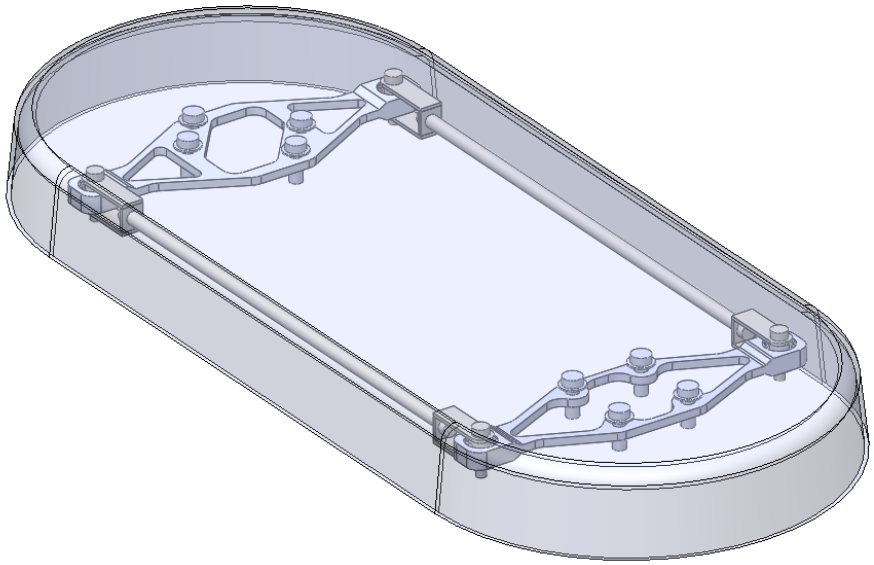
Figur 5.70: Rotordisker montert på ATV

5.4 Beskyttelsesdeksel

Styremekanismen er et farmoment med tanke på klemskader. Det er derfor foreslått for KDA å lage et deksel som kan beskytte mot dette. Dette er ikke en ekstra oppgave som er oppgitt i den opprinnelige oppgaveteksten, men KDA virket uansett svært positive til dette. Faren for klemskader er allerede redusert etter å ha gått fra kjededrift og over til kraftoverføring med stag, men risikoen er fortsatt til stede i område rundt feste mellom stag og roterende disker. Et deksel over rotordiskene vil også kunne beskytte delene for riper og skader. Siden rotordiskene består av aluminium er det ekstra viktig å beskytte disse for riper og andre skader som kan fremprovosere utmattelsesbrudd.

5.4.1 Design av beskyttelsesdeksel

I Figur 5.71 vises design av deksel og hvordan det et tenkt å dekke de bevegelige delene på styresystemet. Med denne utformingen vil det være liten sannsynlighet for at hender befinner seg i klemfarlig område på ATV når dekselet er montert. For designet av dekselet er valgt å gå for en nokså enkel løsning. Mer avansert geometri kan raskt øke tiden på støpeprosessen ettersom det vil ta lenger tid og lage en passende støpeform. Det kan også gjøre det vanskeligere å skille karbonfiberen fra støpeformen etter at herdeprosessen er gjennomført.



Figur 5.71: Design av deksel

5.4.2 Design av festepunkter for deksel

Det har vært en utfordring å finne en god løsning på hvordan man skal feste et deksel som dekker styresystemet på ATV. Det mest optimale hadde vært og lage braketter som festes til rammeverket til ATV som kan holde dekselet på plass over styresystemet. Dessverre består delene til ATV rundt styrestammen av plastikkdeler som det er vanskelig å lage festepunkter på uten å gjøre ikke-reversible endringer. I område rundt steppermotoren ville dette ikke vært et problem siden det er mulig å feste en brakett flere steder på det ettermonterte rammeverket. Siden det ikke ville vært tilstrekkelig å feste dekselet kun ved steppermotoren ble denne ideen skrotet. Siden dette ikke lar seg gjøre må dekselet monteres på rotordiskene. Dette gir da utfordringen ved at rotordiskene roterer mens dekselet må kunne stå i ro. Det må derfor lages festepunkter som kan holde dekselet fra rotordisken uten å skape for stor friksjon mellom disse. Det vil i praksis være umulig å ikke skape en form for friksjon med en slik innfestning, men en liten friksjonskraft ble sett på som et akseptabelt kompromiss for å kunne feste dekselet godt til ATV. Dekselets hovedfunksjon er å beskytte styresystemet å sørge for at ingen hender befinner seg i klemfarlig område. ATV er et terrengkjøretøy bygd for tøff bruk. Et ettermontert deksel bør ikke sette begrensninger for dette. Dekselet må derfor være godt festet og være robust bygd.

5.4.2.1 Valg av materiale for braketter

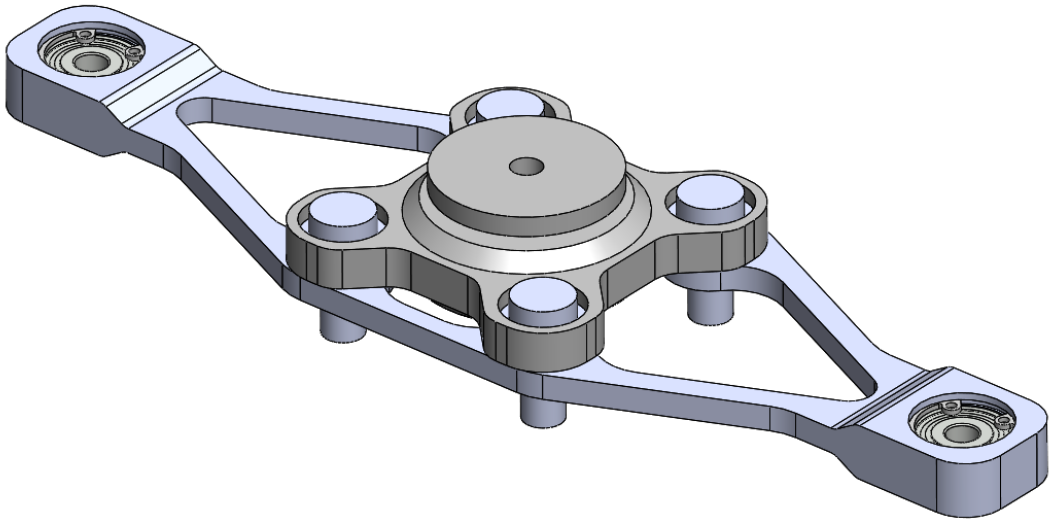
Her ble det valgt å bruke 3D-printing for å raskt og enkelt lage deler som kunne testes og implementeres. 3D-printing er en ”additive manufacturing” prosess som tillatter bruk av avansert geometri. Dette er gunstig her som det er behov for en del som tar begrenset med plass og må være sterk nok til å holde dekselet på plass. For å få den beste styrken ut av de 3D-printede delene ble det bestemt å bruke 100 % fyllingsgrad og ’Tough PLA’ som har en flyt- og fasthetsgrense på 37 MPa. Med en spesifikk vekt på 1.22 har materialet også en svært lav vekt.

5.4.2.2 Konsept for innfestning

For å skape minst mulig friksjon mellom deksel og rotordisker ble det bestemt å bruke kulelager her også. Det ble foretatt en rask test der en 3D-printet del ble boltet fast til

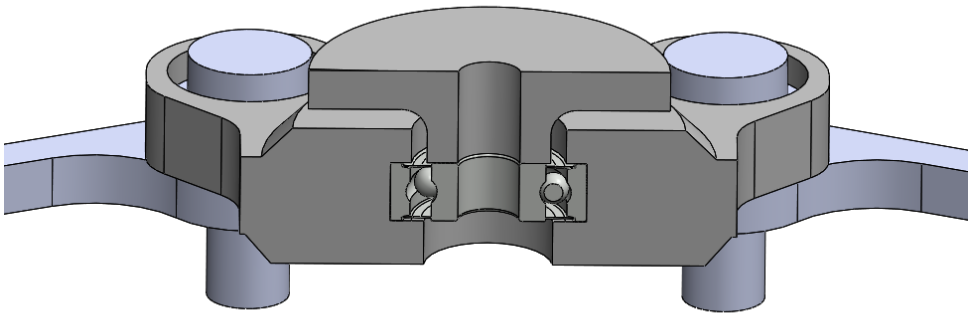
en karbonfiber plate med en metall skive i mellom som et forsøk på å redusere friksjon. Dette viste seg raskt å være en dårlig løsning. Så fort bolten ble tilstrammet oppsto det mye friksjon mellom skiven og delene, selv etter at olje ble påført mellom skive og del. Det ble derfor konkludert at et kulelager var nødvendig.

Det er 4 M8 bolter som holder rotordiskene på plass. Disse kan også brukes for å feste hver sin brakett med kulelager. Kulelagerene vil da kunne bli montert i senter av rotasjonsaksene til steppermotoren og styrestammen. På lik linje som med rotordiskene må det her designes 2 forskjellige braketter som passer til de forskjellige hullbildene til steppermotoren og styrestammen. I Figur 5.72 ser man innfestningskonseptet montert på rotordisken til steppermotoren.



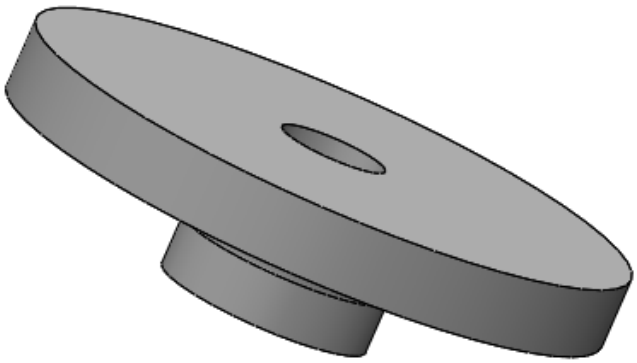
Figur 5.72: Innfestning montert på rotordisk

Siden område rundt kulelageret er svært spenningsutsatt ble kulelageret innkapslet inne i brakettene for å oppnå mest mulig styrke i dette område. Ved å pause 3D-printeren rett før lommen til kulelageret skal lukkes, kan man presse inn kulelageret og få det helt innkapslet i delen. Det ble i tillegg påført et tynt lag med lim på ytterringen til kulelageret for å fordele kreftene kulelageret vil påføre braketten i tillegg til å få en tettere pasning mellom disse. Ulempen med å montere kulelageret på denne måten er at det er umulig å få ut kulelaget uten å ødelegge delen, men siden dette er deler som det er raskt og enkelt å printe nye ble dette sett på som akseptabelt.



Figur 5.73: Section view av innfestning til deksel

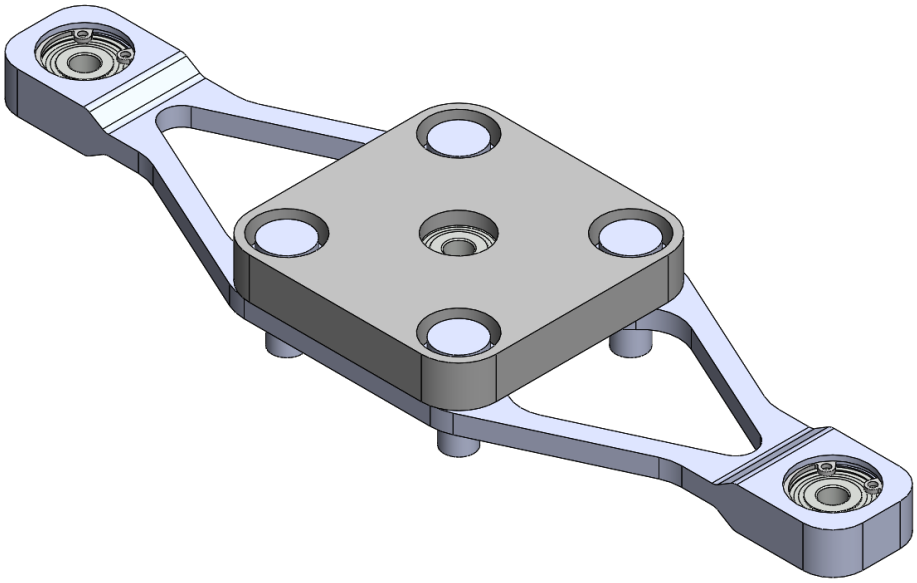
For å feste dekselet fast til kulelageret og braketten skal det brukes en bolt og en tilpasset foring som kun er i kontakt med innerringen til kulelageret. Når braketten da kun er i kontakt med ytterringen får man en roterende forbindelse gjennom kulelageret med svært liten friksjon. I Figur 5.73 ser man hvordan braketten, kulelageret og foringen er satt sammen. Dekselet boltes så fast med en M6 bolt som går gjennom foringen og kulelageret som låses med en mutter. Når denne bolten stammes til vil dekselet være godt fastmontert til styresystemet.



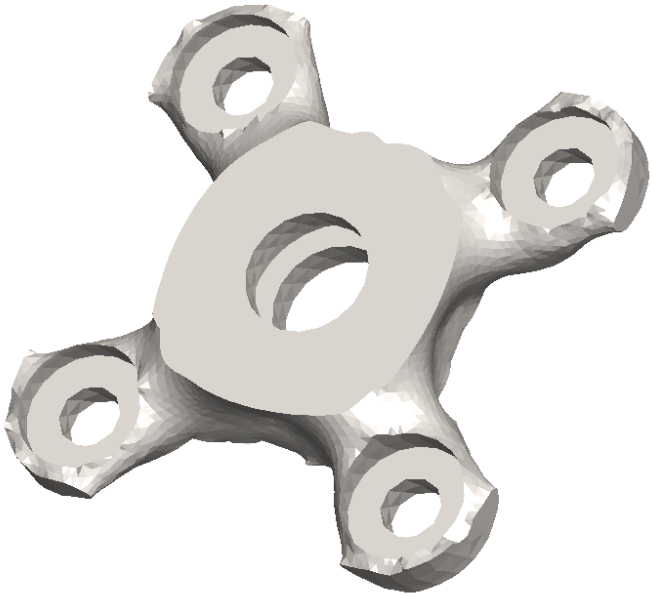
Figur 5.74: Foring mellom deksel og innfestning

Under design av brakettene ble det brukt samme fremgangsmetode som for rotordiskene. De ble først tegnet opp en kloss som dekker hele rommet som brakettene har tilgjengelig av plass mellom rotordisker og deksel. Det er så kjørt en statisk analyse og deretter en

topologi analyse for å generere en optimal geometri på delen. Figur 5.75 braketten før topologi analysen er utført. I Figur 5.76 kan man se resultatet fra topologi analysen.

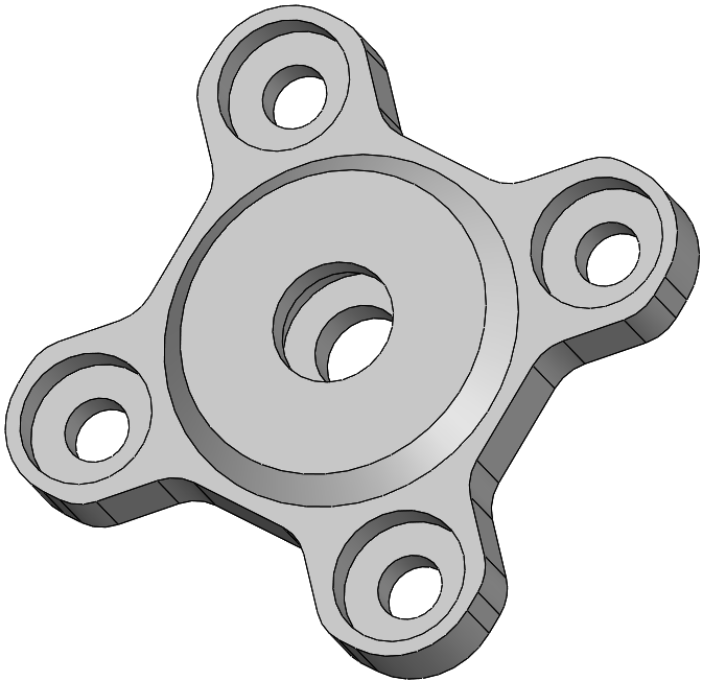


Figur 5.75: Innfestning før topologi studie



Figur 5.76: Resultat av topologi analyse av kulelagerhus på steppermotor

Ved å ta resultatet fra topologi analysen som utgangspunkt har den endelige delen blitt tegnet, se Figur 5.77.



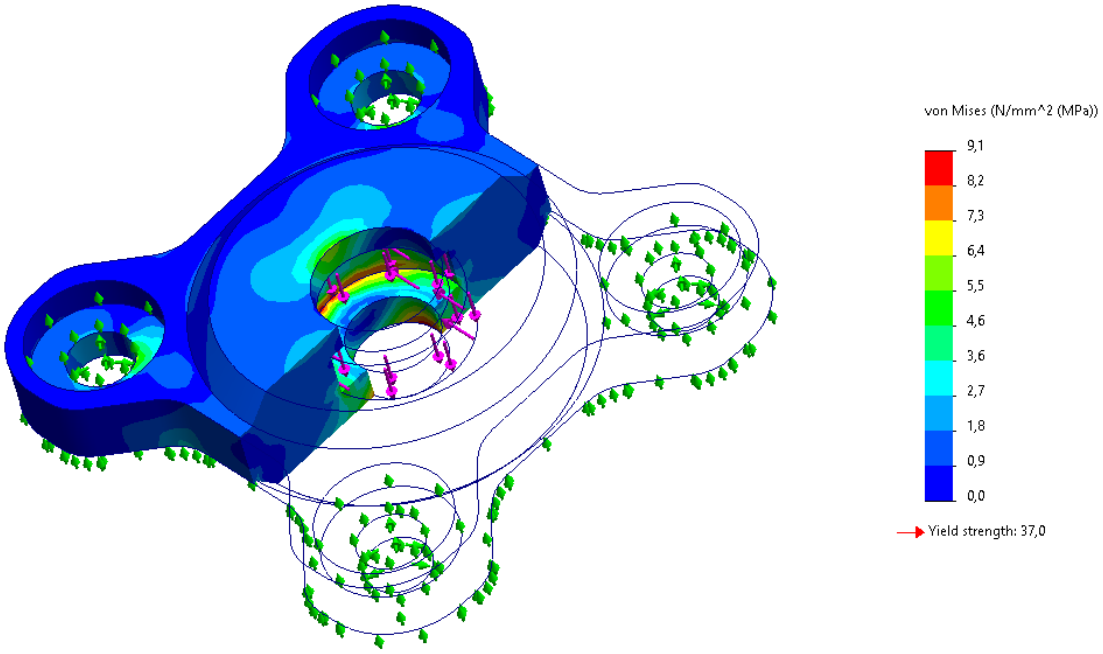
Figur 5.77: Kulelagerhus for steppermotor etter topologi analyse

5.4.2.3 Dimensjonering av brakett til deksel

Det er vanskelig å bestemme en eksakt verdi på styrken som trengs på en innfestning som dette. Dekselet i seg selv vill ikke veie mye, men kan blir utsatt for slag og støt som vil belaste innfestningen. Med denne valgte konfigurasjonen vil kulelageret påføres krefter både radielt og aksielt. Det er valgt å bruke samme kulelager her som i rotordiskene da mange av de samme kravene til kulelageret befinner seg også her. For denne konfigurasjonen vil kulelageret oppleve en større aksial last en for rotordiskene. I følge SKF sine nettsider tåler slike kulelager en statisk aksial last på 25% av den radielle lasten. Disse kulelagerne tåler en statisk last på 950N radielt og dermed 237.5N aksielt. Datablad er lagt i vedlegg A43. Dette vil si at et kulelager alene vil kunne holde et deksel som veier 24kg som statisk last. For dynamiske laster tåler kulelageret vesentlig mer en dette. Dekselet er ikke forventet og veie mer en 1kg så man kan dermed konkludere med at kulelageret vil være robust nok til å tåle de fleste påkjenninger og ha en lang levetid.

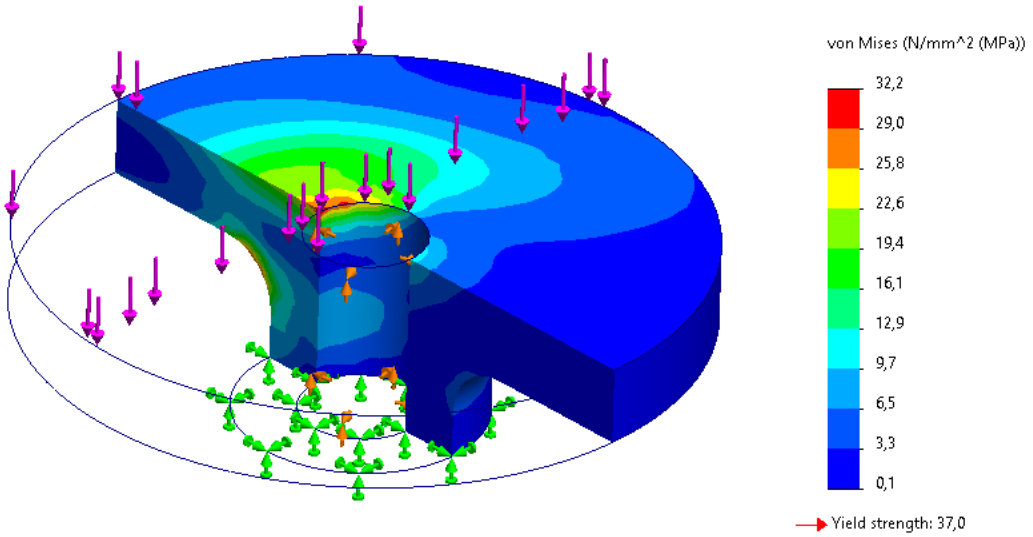
Til slutt ble det foretatt en statisk analyse av delen for å bekrefte topologi analysen og verifisere at den tåler belastningene, se Figur 5.78. Det er her påført en vertikal last som simulerer at ytterringen til kulelageret trykker ned med en last på 50kg. I tillegg er

det påført en tilsvarende horisontal last med bruk av "Bearing load" for å simulere at kulelageret påfører braketten en kraft på sideveggene. Med disse kreftene er braketten påført en kombinert last på 100kg. Resultatet etter å ha kjørt analysen ga en maksimal spenning på 9,9 MPa. Med en flyt- og fasthetsgrense på 37 MPa har man kan dermed konkludere med at braketten vil være sterk nok for de fleste påkjenninger som måtte oppstå. Ettersom braketten kun veier 34g så man ikke et behov for å redusere vekten ytterligere.



Figur 5.78: Statisk analyse av endelig brakett

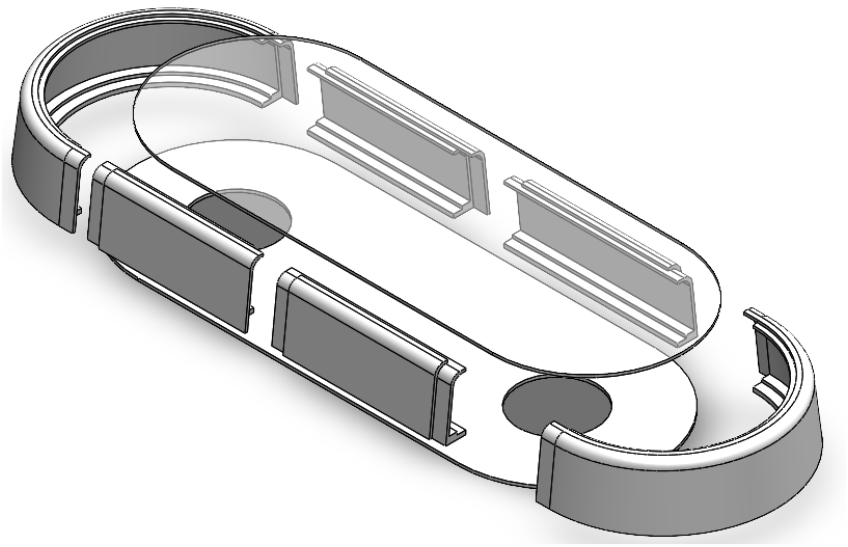
Foringen er også dimensjonert for å tåle en tilsvarende last som braketten. I denne analysen er det påført en last på 50kg på den ene halvdelen av kontaktflaten med dekselet. Dette simulerer at beskyttelsesdekselet blir påført en kraft i område rundt en av sine ytterkanter. Dette er den verst tenkelige situasjonen og vil føre til en høyt konsentrert spenning i det røde område vist i Figur 5.79, hvor det oppstår en maks spenning på 32,2 MPa.



Figur 5.79: Statisk analyse av foring

5.4.3 Produksjon av støpeform

For å kunne støpe et deksel i karbonfiber må det lages en støpeform som brukes for å gi karbonfiberduken den ønskelige fasongen. Her ble flere løsninger vurdert, men valget falt til slutt på å bruke en kombinasjon av 3D-printede deler, kryssfiner- og akrylplater som til slutt limes sammen til en svært solid boks med den innvendige fasongen til dekselet. De 3D-printede delene ble også designet med kilfals i hver ende slik at man fikk en svært sterk sterk skjøt. Delene er vist i Figur 5.80. Bakgrunnen for dette valget var rask produksjonstid og at de 3D-printede delene i kombinasjon med akrylplaten vil gi en god styrke og overflatefinish. Det er svært viktig å å ha en jevn og fin overflate på støpeformen for å klare å skille støpet fra støpeformen etter herdeprosessen er over. Akrylplaten er svært gunstig med tanke på dette. De 3D-printede delene har i utgangspunktet en ru overflate men etter å ha pusset disse delene med finkornet sandpapir fikk man til slutt en overflate som egnet seg til støpeform. Dette var en tidkrevende prosess men resultatet ble svært bra. I Figur 5.81 kan man se det endelige resultatet.



Figur 5.80: Sprengskisse av støpeform

Figur 5.81: Ferdig støpeform

5.4.4 Produksjon av beskyttelsesdeksel

Etter at dekselet var ferdig pusset med sandpapir, ble det lagt på lag med Trennfilm PVA. Dette er en type plast med svært lave friksjonsegenskaper, som legger seg som en fin hinne på utsiden av støpeformen. Dette vil medføre at epoxyen som legges på sammen med karbonfiberen binder seg fast til denne hinnen, fremfor den 3D printede plasten. På denne måten vil denne hinnen fungere som en type slipp-middel, som sørger for at selve karbonfiber formen ikke limer seg fast til støpeformen. Det ble totalt lagt på 3 lag med

trennfilm, som bruker sirka 20min på å herde før neste lag kan legges på. Dette kan ses i figur 5.82.

Figur 5.82: Støpeform etter tre lag med PVA trennfilm

Etter at alle 3 lagene med trennfilm var lagt og herdet, ble det penslet på første lag med epoxy. Det første laget som ble pålagt var svapox 110/700 epoxy med H-2017 bernstein herder. Grunnen for at H-2017 herderen benyttes her, er på grunn av den korte herdetiden og den høye viskositeten den får etter kort tid. Blandingsforholdet som benyttes med denne herderen er 80% epoxy og 20% herder. Før blandingen pensles på, er det særdeles viktig at blandingen blir rørt godt. Dette er viktig for å få fordelt herderen jevnt ut i epoxyen, slik at den herder jevnt over hele materiale. Etter 3 timer herding blir epoxyen svært viskøs og klebrig. Dette er viktig for å få karbonfiber duken som skal legges oppå til

å feste seg godt helt inntill støpeformen langs alle flater.

Karbonfiberduken som er valgt har 12000 filamenter med 6,7 micrometers diameter pr bunte, med en 0-90 orientasjon. Dette gir et relativt grovt vevd mønster. Grunnen for at et grovt veve mønster benyttes er hovedsakelig for å spare innkjøps kostnader, men det vil også spare mye tid ved produksjon ettersom det kreves færre lag for å bygge tykkelse i dekselet. Et grovt mønster vil også gi et visuelt tydelig mønster, som gir et tøff utseene. Duken som ble lagt på er noe lengre enn selve støpeformen, ettersom det er stor sjanse for at det samler seg store dråper med epoxy langs den nederste kanten. Med en litt lengre karbonfiber duk, kan disse endene kuttes av slik at man unngår dette i det endelige karbondekselet. Etter den første duken ble lagt på, ble det penslet enda et lag med svapox 110/700 epoxy og H-2017 herder. Det andre laget med karbonfiberduk kan deretter legges på og festes godt til den første karbonfiberduken. Et bilde av støpeformen etter den første karbonfiberduken var lagt på kan ses i figur 5.83.

Figur 5.83: Første lag med karbonfiber på støpeformen

Etter begge karbonfiberdukene var lagt, ble den mettet med samme type epoxy, men med en ny type herder som bruker lengre tid på herdeprosessen. Herderen som ble benyttet her heter TL1 fra produsenten Ebalta. Blandingsforholdet på denne er 100 deler svapox 110/700 epoxy og 24 deler TL1 herder. Dette tillsvarer at 19,3% av totalen er herder. Før denne blandingen påføres karbonfiber duken, er det viktig at den av-gasses. Dette gjøres ved å sette inn blandingen under vakuum i sirka 10 minutter, eller til blandingen har sluttet å boble helt. Dette medfører at alle eventuelle luftbobler som kan ha kommet inn ved innrøringen, blir eliminert. Dette er viktig for å unngå at eventuelle luftbobler blir til små porer i materiale etter herding. Slike porer er svært ugunstig ettersom det kan svekke dekselets motstandsdyktighet mot utmattelse brudd, ettersom de kan initiere sprekkvekst. Det er også viktig å pensle på mange tynne lag med epoxy fremfor få tykke lag. Dette er ettersom tykkere lag med epoxy har en tendens til å legge seg mer ujevnt. Herdetiden med dette blandingsforholdet ligger på mellom 18 og 24 timer i romtemperatur, men hvis formen plasseres i sola kan herdetiden reduseres betraktelig. Blandingen trenger heller ikke å være fullstendig herdet før neste lag med epoxy kan legges. Totalt ble det lagt på 7 lag med denne blandingen. I figur 5.84 kan karbonfiberdekselet etter syv lag med epoxy ses.

Figur 5.84: Karbonfiber deksel etter syv lag med epoxy

Etter det siste laget med epoxy var lagt, ble dekselet pusset ned med sandpapir. Dette er for å eleminere eventuelle forhøyninger og ujevntheter i overflaten. Det er svært viktig å passe på at det kun er epoxy som blir pusset ned, og ikke selve karbonfiberen. Dette er fordi dette kan medføre en matt gråfarge i dekselet, som ikke er ønsket. Etter formen var ferdig pusset ned, ble den klarlakket. Resultatet etter klarlakking kan ses i figur 5.85.

Figur 5.85: Karbonfiberdeksel etter klarlakk

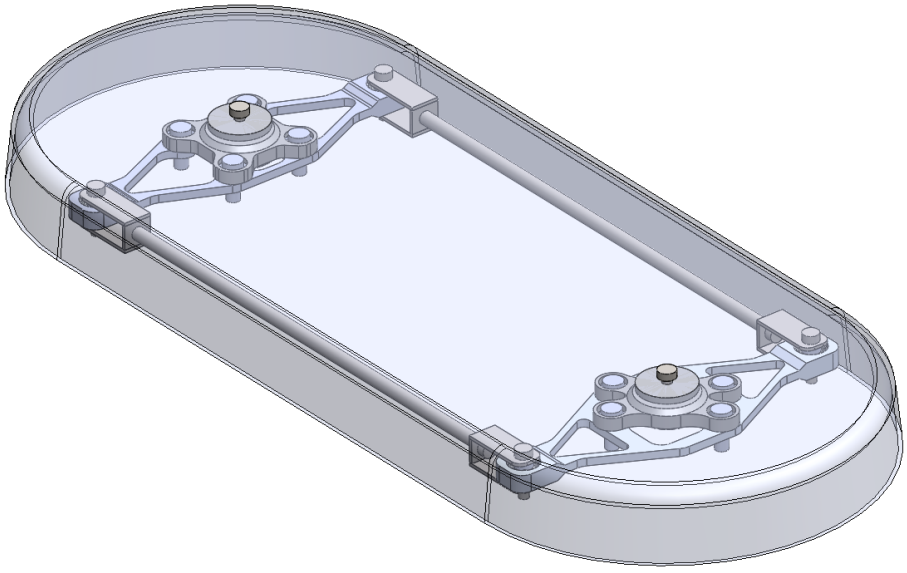
Etter at formen ble klarlakket, ble den trimmet ned med vinkelsliper, til å passe på ATV. Etter kuttingen ble underkanten også pusset over, slik at den ble jevn langs hele. Det ble også borret opp to hull i formen for braketten til festing. Det endelige sluttresultatet montert på ATV kan ses i figur 5.86.

Figur 5.86: Karbonfiber deksel montert på ATV

5.4.5 Resultat etter utvikling av beskyttelsedeksel

Etter den endelige pussingen og klarlakkingen av dekselet, endte det opp med en svært glatt og pen overflate. Karbonmønsteret kommer også svært godt frem, som gir et lekkert utseende. Etter montering av dekselet, ble det også kartlagt at dekselet sitter svært støtt på brakettene for festing. Svinging med styremekanismen ble også testet med dekselet påmontert, og fungerte veldig godt. Figur 5.87 viser hvordan styresystemet ser ut nedenifra montert inne i beskyttelsesdekselet. 3D modellen av hele styresystemet montert inne i dekselet kan ses i figur 5.88.

Figur 5.87: Styremekanisme montert inne i karbonfiberdekselet



Figur 5.88: Sammenstilling av styresystemet

6 Elektronikk

For å oppnå fjernstyrt, og senere autonomt girskift, trenges det et mekatronisk system. Dette kapittelet tar for seg den delen av oppgaven som går omfatter elektronikk, signalbehandling og mekatronikk.

6.1 Aktuator - LA14

Figur 6.1: Bilde av LA14 aktuator, hentet fra brukermanual[7].

Oppdragsgiver hadde kjøpt inn en lineær aktuator på forhånd for å kunne bytte gir på ATV, av type LA14 produsert av LINAK A/S, avbildet i fig. 6.1.

LA14 kommer i mange utgaver med forskjellige spesifikasjoner, men denne, med typenummer 14040130000A0C06=1A002450CT0B0, har følgende spesifikasjoner:

Ty.nr.seg.	Spesifikasjon	Verdi
14	Produkt	LA14
040	Spindelstigning	4 mm
130	Slaglengde	130 mm
00	Sikring	Ingen
0A	Tilbakekobling	Hall-potmeter
0	Platform	Ingen
C	Motortype	12 VDC, Hurtig (V1)
0	Endestopp	Effektbryter (E1)
6	IP	IP66
=	Farge	Mørk Olivengrå NCS S7000-N
1	Bakfeste	0°
A	Stempelstangøye	⌀10.2 mm AISI304 (0231096)
0	Alternativ/posisjon	Ingen EOS (Kun IC Basic)
0	Bremser	Ingen
245	Install. dim.	245 mm
0	Brannkategori	Ingen
C	Pluggtype	”Flying leads”
T	Kabel	Rett, 1.5 mm (8-kjerne)
0	Sikkerhetsfaktor	2.0
B	Tilbakekoblingsnivå	0.5 V—4.5 V
0	Paralellmodus	Ikke-kritisk paralell

Tabell 6.1: Spesifikasjoner for kjøpt inn LA14 aktuator med typenummer 14040130000A0C06=1A002450CT0B0[7]

Aktuatoren trenger å kobles til 12 V spenningsforsyning, og drives av ±12 V pådrag hvor den trekker opp til 5 A. Aktuatoren kan skyve og trekke med opp til 300 N. Typisk strøm for pådrag ligger på 400 mA uten last og 1.7 A med full last. Denne informasjonen er hentet fra tabellen markert ”technical specifications” i [8].

Hall-sensortilbakekoblingen har en toleranse på ±200 mV og kan gi fra seg max. 1 mA[9]. Dersom inngangsimpedansen på endepunktet til tilbakekoblingssignalet er for lav vil det kunne føre til en unøyaktig måling.

Denne aktuatoren ser ut til å være godt egnet for vår bruk, og gruppen har derfor valgt å fortsette å bruke denne for vårt system.

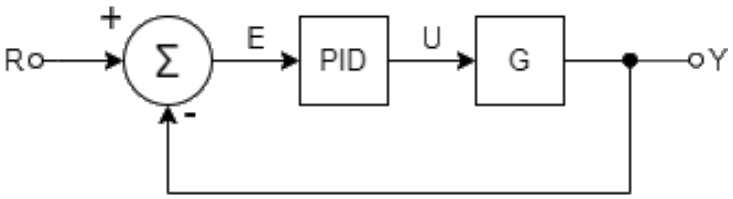
6.2 Design av kontrollsløyfe for gir-aktuator

Dette kapittelet omhandler teorien bak kontrollsystemet for gir-aktuatoren.

Lineæraktuatoren kan tilnærmes som et integrert 1. ordens system på formen:

$$G(s) = A\frac{\omega_n}{s^2 + \omega_n s}. \tag{6.1}$$

For å styre aktuatoren kan en kontrollsløyfe med negativ tilbakekobling settes opp på formen vist i fig. 6.2. Det ønskes å styre posisjonen til aktuatoren, og systemet kan dermed kategoriseres som et type 1 system.



Figur 6.2: Et funksjonelt blokkdiagram for kontrollsløyfen (forenklet). Figuren tar ikke for seg maskinvaret i systemet, kun de forskjellige transferfunksjonene fra referanse (R) til utslag (Y). Pådrag er markert U og avvik er markert E. Aktuatoren inntrer som "plant" i systemet og er markert G. PID-kontrolleren er markert PID og kan brukes for å gi systemet den responsen som er ønsket.

Kontrollsløyfen i fig. 6.2 gir følgende systemtransferfunksjon:

$$H(s) = \frac{Y(s)}{R(s)} = \frac{C_{pid}(s)G(s)}{1 + C_{pid}(s)G(s)}. \tag{6.2}$$

Transferfunksjonen C_{pid} til PID-kontrolleren er gitt ved

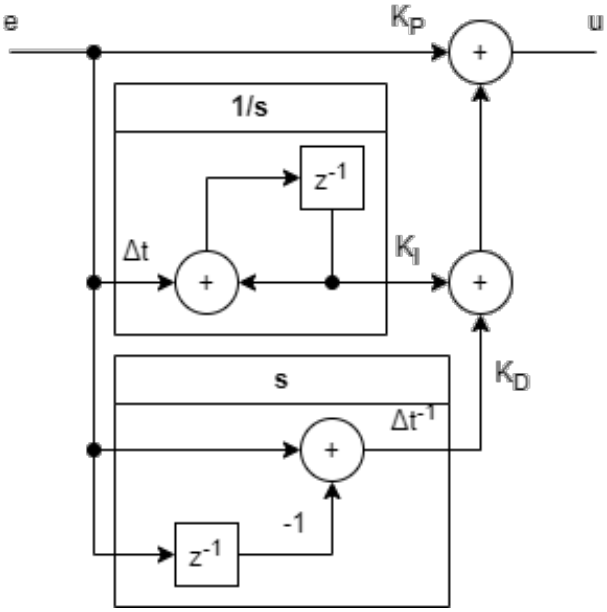
$$C_{pid}(s) = K_P + \frac{1}{s}K_I + sK_D, \tag{6.3}$$

hvor da K_P, K_I og K_D er parametere som stilles inn for å oppnå ønsket respons i systemet.

En diskret-tid tilnærming av en PID-kontroller ved Eulers metode er gitt ved

$$C_{pid}(z) = K_P + \Delta t\frac{z^{-1}}{1 - z^{-1}}K_I + \frac{1}{\Delta t}\left(1 - z^{-1}\right)K_D, \tag{6.4}$$

hvor Δt er tid fra forrige løkke-syklus (ofte kalt "delta-time"), og kan implementeres som vist i fig. 6.3.



Figur 6.3: En diskret-tid implementasjon av en PID-kontroller.

For å oppnå en tilbakekobling som vist i fig. 6.2, trenger man å kunne kontinuerlig måle utslaget Y på aktuatoren. LA14 har en analog tilbakekoblingsutgang som dekker dette behovet[9]. Signalet har noe støy, og må dermed filtreres gjennom et lavpass-filter. Etter å ha analysert støyen i frekvensspekteret, har et 1. ordens filter med knekkfrekvens på 20 Hz blitt brukt som støyfilter, hvilket fjerner mesteparten av støyen uten å ha en betydelig innvirkning på systemets respons.

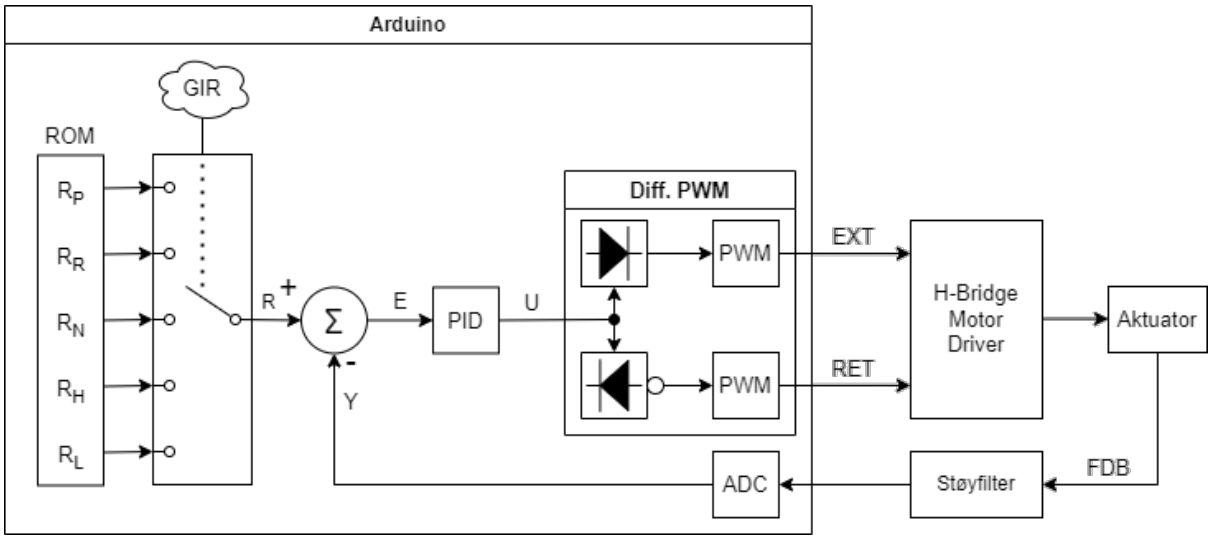
Motoren i aktuatoren må også drives, d.v.s. pådraget må forsterkes slik at det blir levert tilstrekkelig effekt til motoren. Dersom det er mulig å bruke en enkel H-bro driver[5] kan denne driveren opereres med differensiell PWM[6].

Referansen r, altså verdien det ønskes at utslaget skal følge, skal kunne velges ut ifra en satt gir-verdi n_{gir} som vist i følgende uttrykk:

$$r(n_{gir}) = \begin{cases} r_P & n_{gir} = 0, \\ r_R & n_{gir} = 1, \\ r_N & n_{gir} = 2, \\ r_H & n_{gir} = 3, \\ r_L & n_{gir} = 4 \end{cases} . \tag{6.5}$$

[5]Også kjent som "dobbel full-bro driver"
[6]Pulse Width Modulation (norsk: pulsbreddemodulasjon)

Hele systemet, sett bortifra driver og aktuator, kan implementeres digitalt i en mikrokontroller. Kontrollsløyfen, da med maskinvare tatt i betraktning, vil bli seende ut som vist i fig. 6.4.

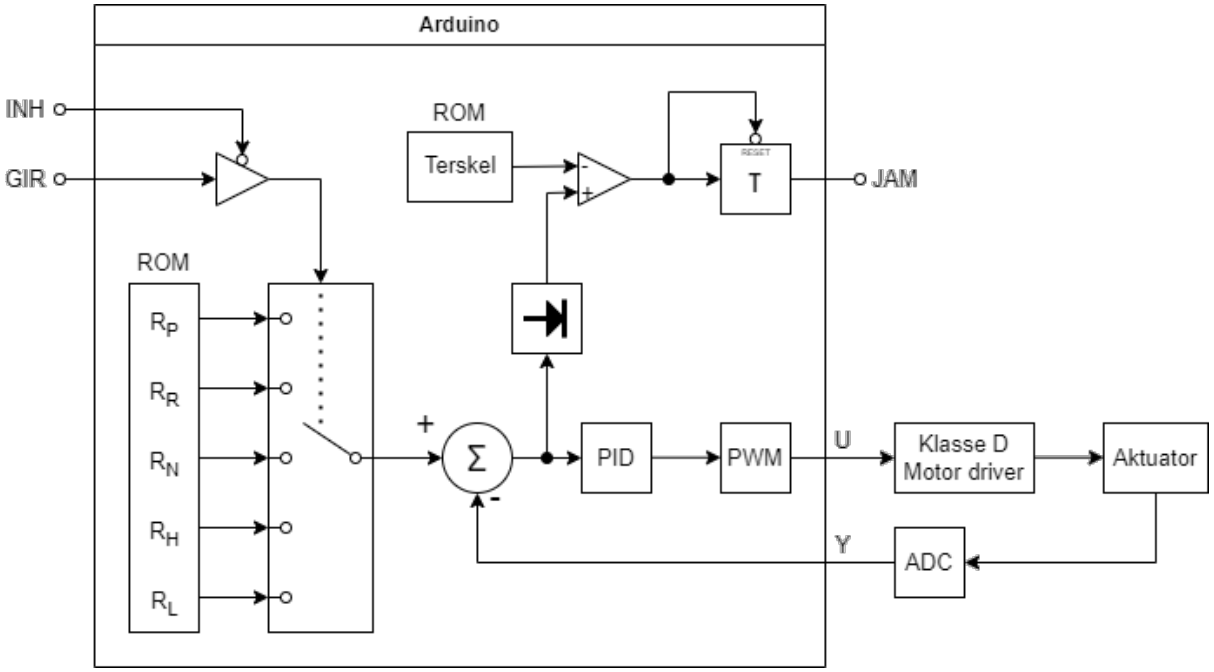


Figur 6.4: Kontrollsløyfen brukt til kontroll av aktuator for girsystemet.

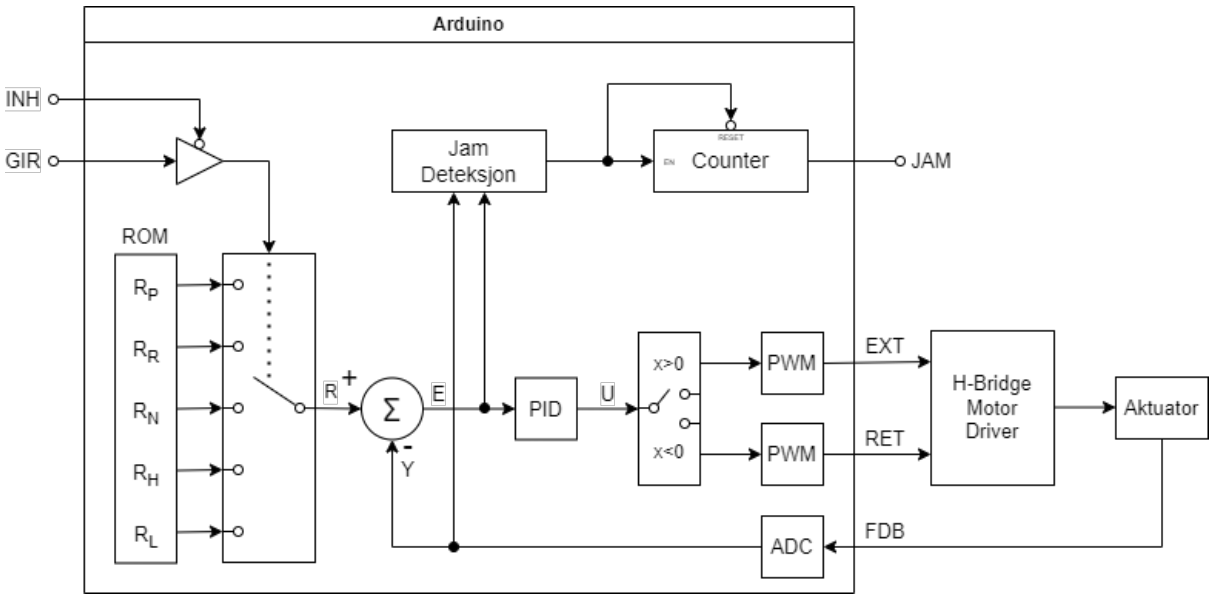
Noen ikke-kritiske detaljer er utelatt i fig. 6.4. Før pådraget U går til diff. PWM-stadiet blir den også jevnet ut med en satt tidskonstant, funnet ved prøving og feiling. Når pådraget er tilnærmet lik 0 blir også et stopp signal sendt til motordriveren, men dette har ingen kritisk funksjon annet enn å sikre at driveren gir 0 pådrag i tilfelle det skulle skje en feil med differensielle PWM-pådraget.

6.2.1 Tidligere utkast

I tidligere utgaver av kontrollsløyfen ble de digitale pinnene på Arduino'en brukt til å velge gir, og programmet hadde et innebygd system for å detektere om giret hadde satt seg fast (jam-deteksjon) via software. To utkast er vist i fig. 6.5 og fig. 6.6.



Figur 6.5: Tidlig utkast av kontrollsløyfen med digital-pin styrt gir og enkel jam-deteksjon som kun ser på amplituden av avviket E for å avgjøre om giret sitter fast. På dette punktet var en klasse D driver vurdert for å drive aktuatoren.



Figur 6.6: Senere utkast av kontrollsløyfen med digital-pin styrt gir og en jam-deteksjon som sammenligner reell fart med estimert fart ut ifra avvik E og tilbakekobling Y. Her er differensiell-PWM-stadiet illustrert med en bryter i steden for to halvbølgelikerettere, men konseptet er fortsatt det samme.

Ved bruk av motordriver EM208 var ikke lenger denne jam-deteksjonen nødvendig, siden motordriveren hadde en innebygd strømgrense som ga et signal ved utslag. Nå som motordriver L298N blir vurdert fremfor EM208, er dette en motordriver uten en slik funksjonalitet. Dermed er denne typen software jam-deteksjon nå én av flere aktuelle løsninger.

6.3 Valg av motordriver

Utover prosjektet har er det blitt forsøkt å bruke 4 forskjellige motordrivere for å drive aktuatoren, med varierende resultater hvorav én ble forkastet under design-stadiet og én ikke fungerte i det hele tatt. Til slutt har vi endt opp med å bruke L298N som driver i vårt system.

På Uno-kretskortet er driveren en separat modul, mens med Portenta-kretskortet er driveren montert rett på kretskortet.

En sammenligning mellom spesifikasjonene til custom H-bro driver, EM208 og L298N ligger i vedlegg A10.

6.3.1 Custom klasse D motor driver (eget design)

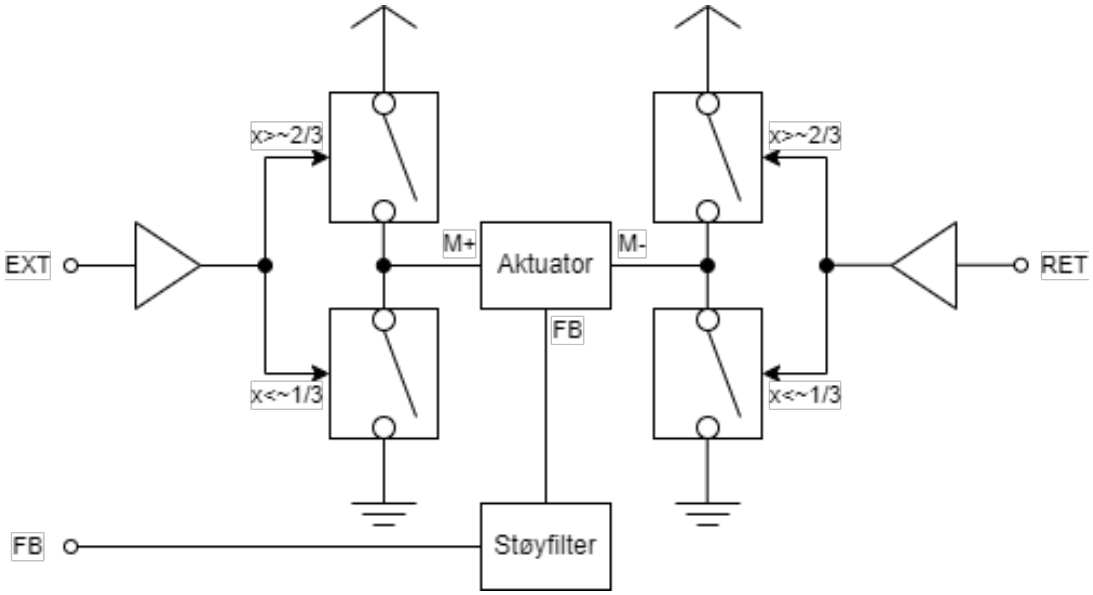
Helt i startfasen av prosjektet ble det først designet en klasse D motordriver. Denne motordriveren kunne styres med ett enkelt PWM-signal. Kretstegningen er ikke lenger å finne.

Denne ble tidlig byttet ut i favør av H-bridge driveren beskrevet i kapittel 6.3.2, ettersom H-bro driveren var mer lineær, hadde et større arbeidsområde, samt at den ikke trengte en spole. Denne motordriveren ble dermed aldri prøvd ut i praksis, og kom aldri lenger enn til tegnebrettet.

6.3.2 Custom H-bro driver (eget design)

Et par uker inn i prosjektet ble konseptet for motordriveren omgjort fullstendig, etter at det ble etablert at en H-bro type driver er bedre egnet for aktuatoren. Dette designet var det første som ble koblet opp og testet med aktuatoren.

Konseptet er illustrert i fig. 6.7. Her er en vanlig H-bro, hvor hver bryter er styrt av et PWM-signal. For å sikre at kun én bryter på hver ende av motoren er lukket om gangen er det nødvendig at det er et dødbånd for spenningen som styrer bryterene. Forsterkere er plassert på hver PWM-inngang for å generere et signal på 12V fra et på 5V.

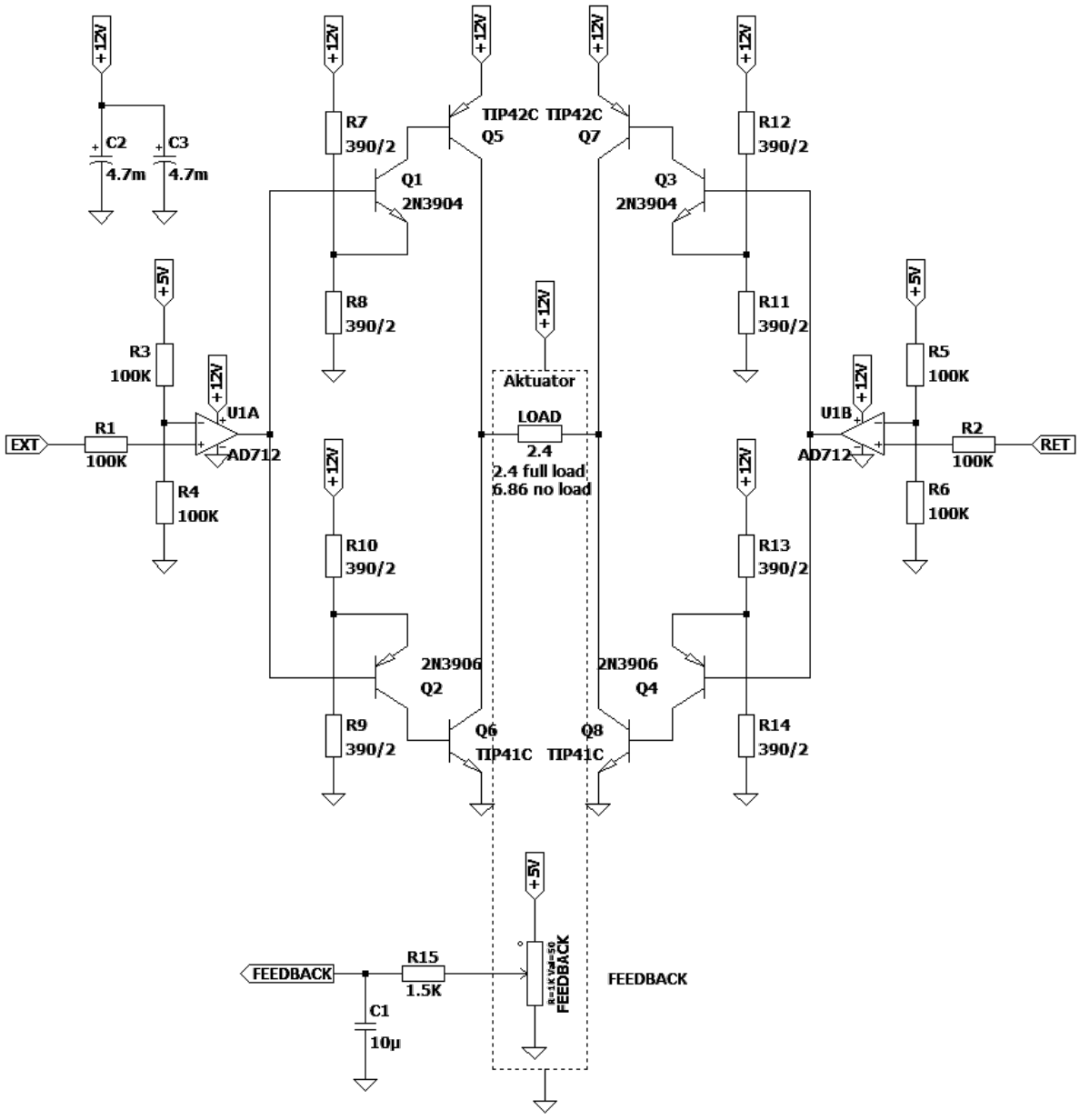


Figur 6.7: Blokkdiagrammet illustrerer konseptet for H-bro driveren. Tilbakekoblingen fra aktuatoren er også med på tegningen.

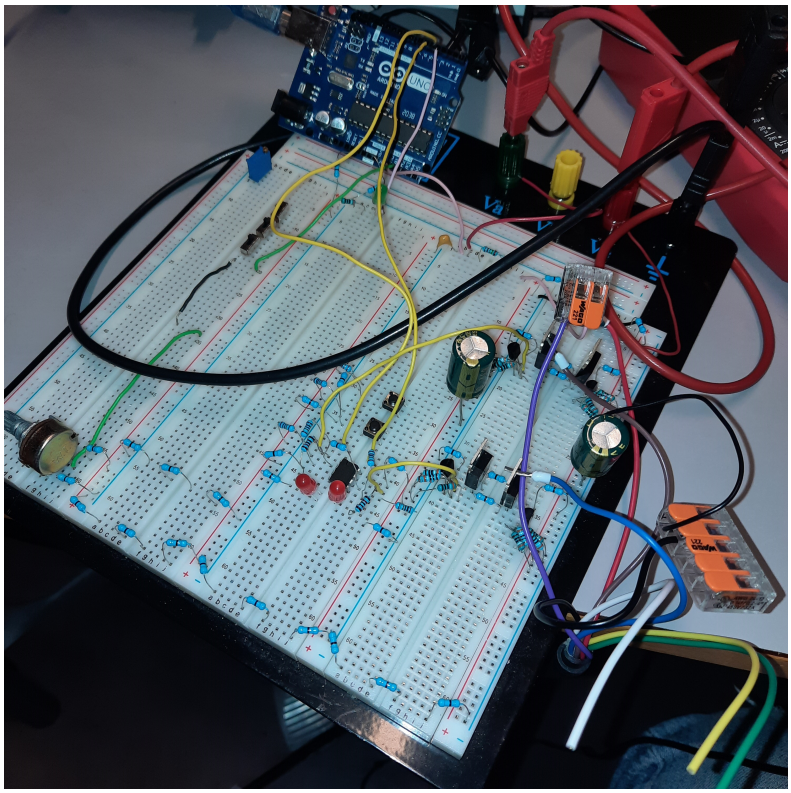
Siste revisjon av kretstegningen er vist i fig. 6.8. Som inngangsforsterker (U1) ble op-ampen AD712 (variant AD712KN) brukt. Denne op-ampen har suboptimal slew-rate, men var lett tilgjengelig. Et par andre op-amper med høyere slew-rate ble prøvd ut: NE5532 og JRC4558, men disse viste seg å ikke fungere når kretsen ble koblet opp. En resistor på 100 kΩ ble satt på inngangen for å unngå at spenningen faller utenfor op-ampens arbeidsområde og var nødvendig ved bruk av AD712. Det er mulig at denne inngangsresistoren kan reduseres for å få bedre frekvensrespons for kontrollsignalene. For H-broen ble det brukt BJT'er for å kunne skru av og på strømmen gjennom hver grein. BJT-løsningen ble ikke valgt fordi det er den beste løsningen, men simpelthen fordi gruppen ikke hadde MOSFETs med høy nok effekt-rating tilgjengelig. Disse BJT-stadiene er koblet opp i noe som ligner på Sziklai-konfigurasjon, men hvor de første transistorene (Q1-Q4) har emitter koblet til en spenningsdeler i stedet for de andre effekttransistorens (Q5-Q8) collector. Dette ble gjort for å sette en terskel slik at begge greienene på hver sin side av H-broen ikke leder strøm på samme tidspunkt, som ville ført til kortslutning. For spenningsdelerene på Q1-Q4's emitter ble det brukt to 390 Ω 1/8 W resistorer i paralell, fordi det ikke var 180 Ω 1/4 W resistorer tilgjengelig. Den lave resistansen på Q1-Q4's emitter er nødvendig for å oppnå høy nok forsterkning, men dette kommer med den ulempe at det går betydelig strøm gjennom spenningsdeleren selv når pådrag er 0.

Utglattingskondensatorene C2-C3 ble lagt til etter behov siden det oppsto for høy

rippel ved testing av kretsen på ATV'ens batterispenning. Før dette punktet ble to utglattingskondensatorer på 470 µF brukt, men ved å øke hver av disse til 4.7 mF ble problemet løst.



Figur 6.8: Kretstegning for custom H-bro driver, med tilbakekoblingssløyfe for aktuatoren.



Figur 6.9: Bilde av custom H-bro driver koblet opp på breadboard. Det er tydelig at måten kretsen er koblet opp på kunne vært mer robust. For eksempel er M+ (brun) og M- (blå) koblet til H-broen ved å legge kablene inni skruehullet på heteskjoldet til effekttransistorene, og kan lett dette ut. Kretser på breadboard skal kun brukes under prototyping og vil aldri være en del av vårt ferdige produkt.

Kretsen har blitt testet grundig med aktuator uten last, drevet på 12 V fra variabel spenningsforsyning EA-3003S. Den ble også testet to ganger på batteriet til ATV, uten last. Den har ikke blitt testet med last (altså med aktuator festet til girstag). Den måtte da revideres etter første test med batteri på ATV, på grunn av for høy forsyningsrippel. Kretsen har bestått alle tester hittil og er fortsatt et konsept som kan vurderes for bruk i det endelige produktet. Mer informasjon om prototypingsprosessen for denne driveren står i vedlegg A9. Kretsen er vist slik den var under bruk, koblet opp på breadboard, i fig. 6.9.

Av like vel, lener gruppen mer mot å bruke ferdig produserte motordrivere som kan kjøpes og bestilles av en fabrikant. Dette gjør det lettere for fremtidige LoneWolf grupper å erstatte driveren om nødvendig. Bruk av et eget design vil også kreve ytterlig dokumentasjon og testing.

Under testing var det et problem at driveren var skjørt satt sammen med løse ledninger, ettersom kretsen var koblet opp på breadboard. Av denne grunn ble ikke driveren testet

med aktuator festet til girstag.

6.3.3 EM288

Denne driveren var kjøpt inn av oppdragsgiver på forhånd for bruk med LA14 og er produsert av LINAK A/S. Den har svært bred funksjonalitet. Driveren er ikke spesifisert for bruk med LA14 eksplisitt (datablad for driver inneholder en liste over støttede aktuatorer), men dens spesifikasjoner er av like vel passende for aktuatoren[10]. Den kan konfigureres med konfigurasjonsmodul EM236A Interface Unit som gir en lang rekke med innstillinger. Det er usikkert om denne driveren kan styres med PWM-signaler.

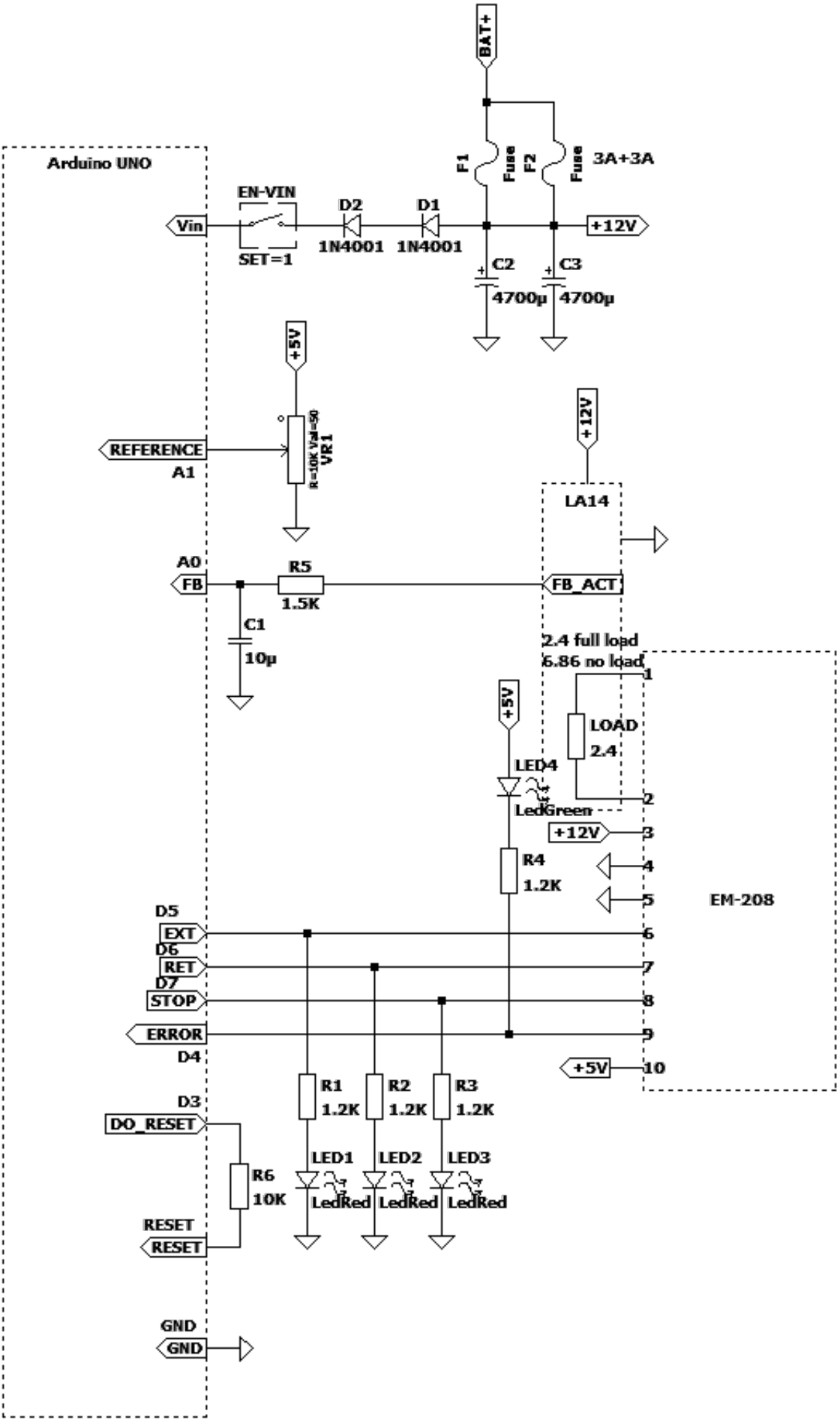
Dessverre viste det seg at denne driveren var ødelagt, og en annen driver, EM208, ble kjøpt inn i stedet. Feilsøkingsprosessen av EM288-driveren er beskrevet i detalj i vedlegg A9.

6.3.4 EM208

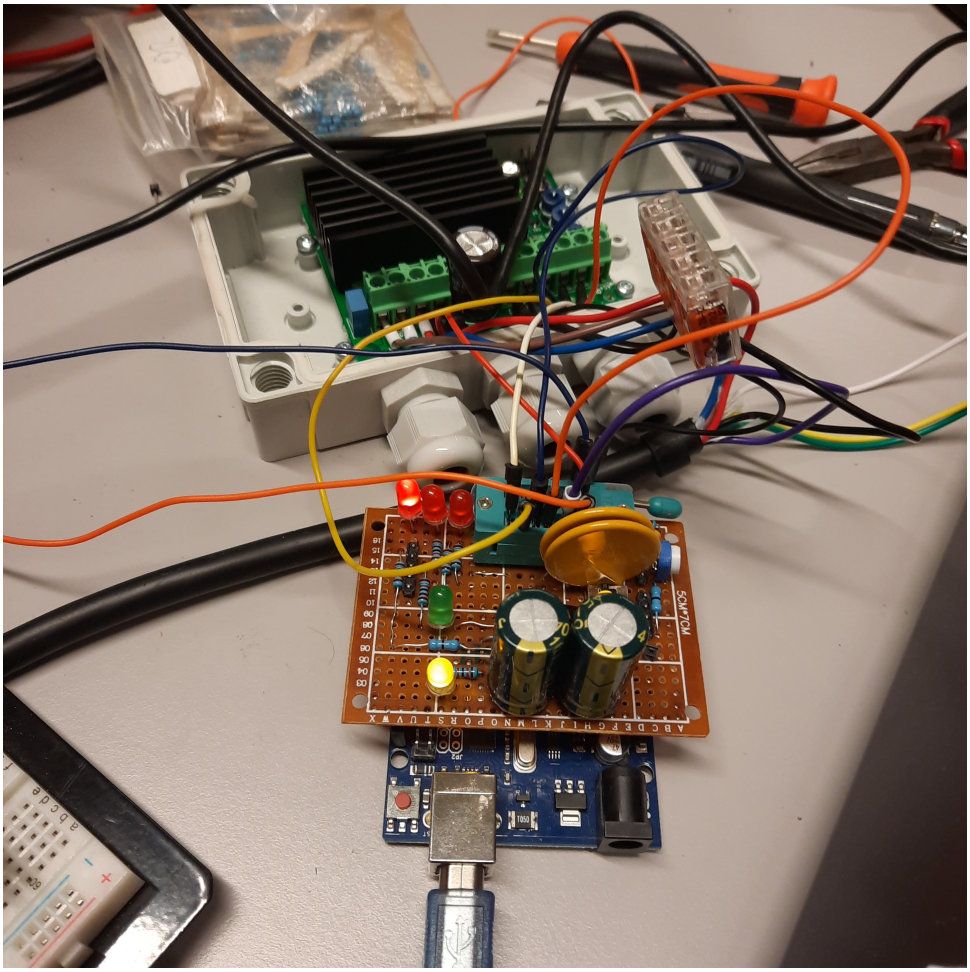
EM208 ble kjøpt inn etter at EM288 viste seg å være i ustand. Denne ble valgt etter forespørsel om en driver med noe redusert antall funksjonalitet, ettersom EM288 hadde mange unødvendige funksjoner for prosjektet.

EM208 er spesifisert for bruk med LA14, kjører på 12 V—35 V og kan levere opp til 12 A. Driveren har en innebygd strømstopp, som kan stilles inn i et område på 1 A—25 A. Når strømstoppen slår ut gir driveren fra seg et aktiv-lav ”error”-signal og stopper alt pådrag til kontrollsignalet møter et satt kriterium (enten at det bytter retning eller at det nullstilles). Driveren kan brukes med kontrollsignaler på 5 V eller 10 V og kan stilles inn til å være aktiv-lav eller aktiv-høy. EM208 har også innstillbar start-tid og stopp-tid som velges med potmeter. Informasjonen i dette avsnittet er hentet fra [11].

For vår bruk av EM208 har gruppen valgt å stille stopp- og start-tid til 0 %, logikksignaler er satt til aktiv-høy 5 V, og strømstopp er satt til ca. 1.2 A. Driveren ble koblet opp som vist i fig. 6.10. Strømstopp-funksjonen gjorde at den da eksisterende jam-deteksjonsalgoritmen ikke lenger var nødvendig, og strømstoppen viste seg å være en mer pålitelig måte å detektere om giret hadde satt seg fast.



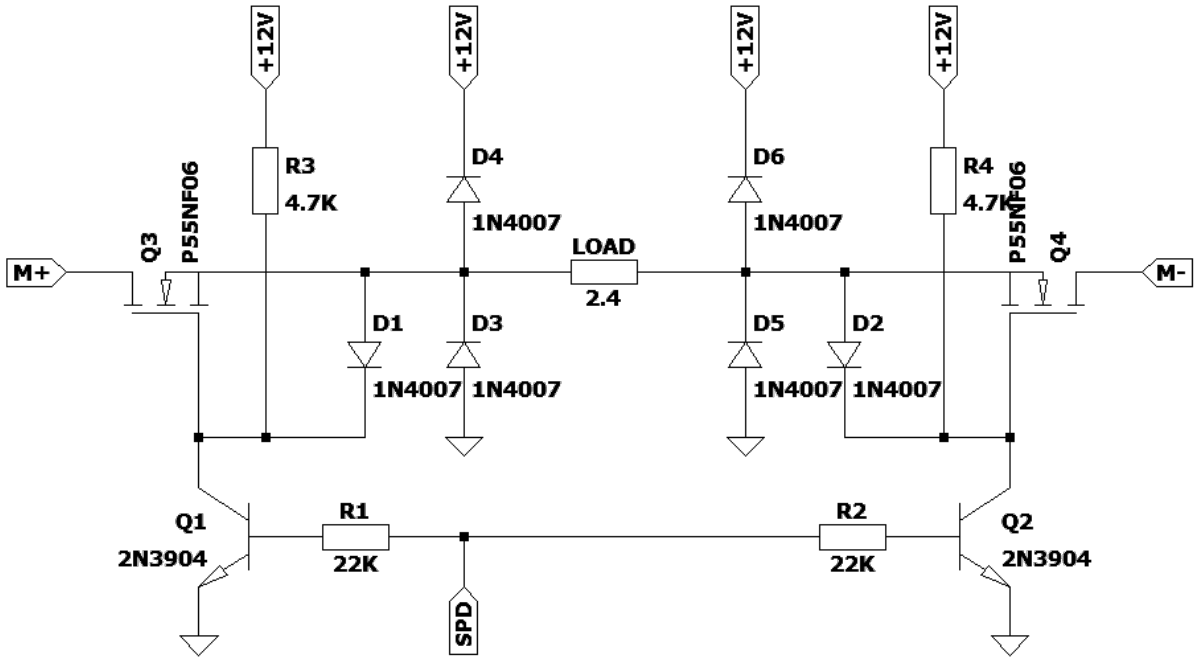
Figur 6.10: Koblingsskjema mellom Arduino Uno, EM208 og LA14.



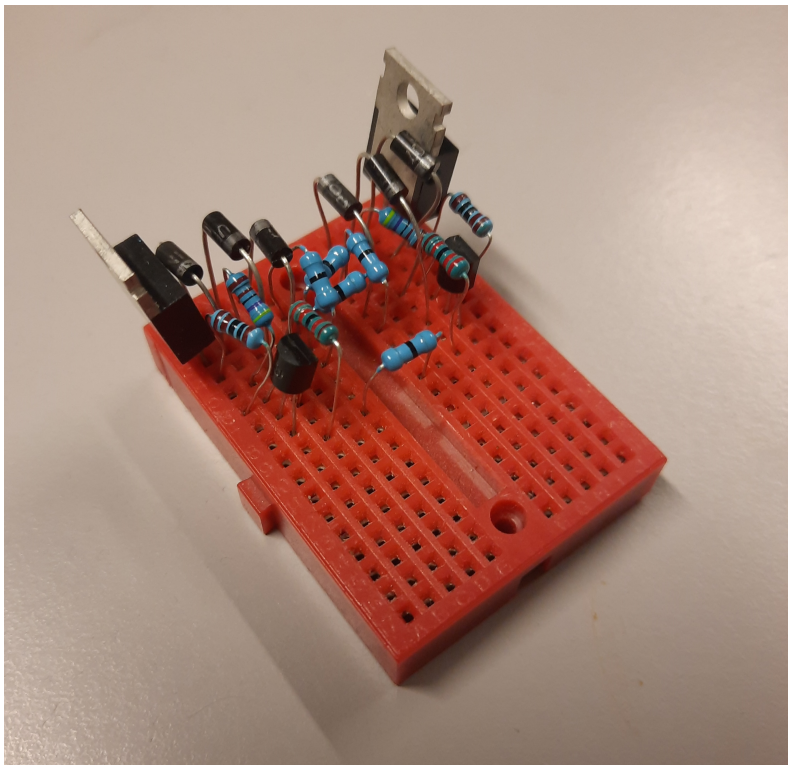
Figur 6.11: Bilde av EM208 koblet til mikrokontroller utstyrt med prototype for LW-GCA1-kretskort, etter kretstegning vist i fig. 6.10.

Ved bruk av EM208 var det stadig vekk et problem at aktuatoren fikk oversving, undersving og hakkete bevegelser. Årsaken viste seg å være at EM208 ikke er laget for å styres med PWM. Dette var ikke oppgitt i driverens dokumentasjon[11][12][13][14], men ble bekreftet gjennom direkte kontakt med LINAK's kundeservice. Det var grenser for hvor godt kontrollparameterne kunne stilles inn for å få bedre styring. Aktuatoren fikk ingen reaksjon på PWM signaler med duty-cycle under ca. 40 %, og bevegde seg med full hastighet dersom duty-cycle var over ca. 40 %. Resultatet var at finkontroll av hastighet på aktuator var umulig.

Det ble gjort en rekke forsøk på å oppnå finkontroll på tross av EM208's mangel på PWM-styring. Det ble forsøkt å bruke PWM kun på stop-signalet, men dette ga samme resultat. Ved å redusere PWM-frekvensen fikk man styrt hastigheten på aktuatoren til en viss grad, men denne kontrollen viste seg å være svært ulineær. Det ble laget en solid-state PWM-styrt relee vist i fig. 6.12 og fig. som ble koblet til utgang på driver, men dette skapte store impulser som stadig vekk trigget driverens strømstopp på grunn av plutselig "switching" av lasten.



Figur 6.12: Solid-state PWM-styrt relee for kontroll av aktuatorens hastighet ved bruk av EM208. Kretsen ble koblet direkte på driverens utganger M+ og M-. Her sendes et aktiv-lav PWM signal på SPD.



Figur 6.13: Solid-state PWM-styrt relee koblet opp på breadboard etter kretstegning i fig. 6.12. Her er ledningene som går fra driver, fra kontroller og til aktuator utelatt fra bildet.

Det ble også gjort et forsøk på å oppnå god *nok* styring uten å prøve å finjustere hastigheten på aktuator. Dette ble gjort ved å bare sende stabil høy eller lav i stedet for PWM. Dette gjorde oversving verre og førte til større unøyaktighet.

EM208 ble etterhvert byttet ut med L298N, som har mulighet for PWM-styring.

6.3.5 L298N

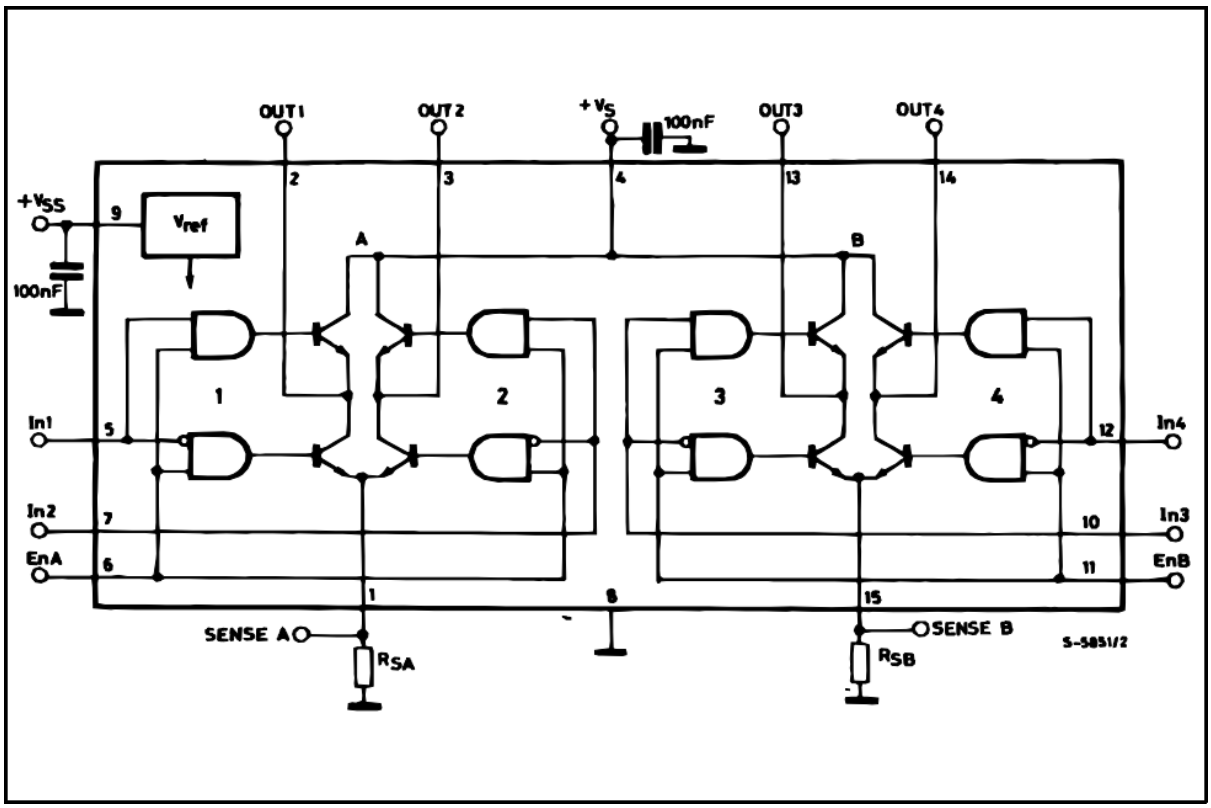
Etter mange forsøk på å få gode resultater med EM208, ble L298N vurdert som motordriver. Denne driveren ble testet både med og uten last, av og på ATV, og ga tilfredstillende resultater.

L298N er en av de mest utbredte og tilgjengelige motordriverne på markedet[15]. Det finnes god dokumentasjon på driveren, og mange eksempler på hvordan denne blir styrt med PWM. Denne er svært populær til hobby-prosjekter ettersom den har en lav pris og er enkel å bruke.

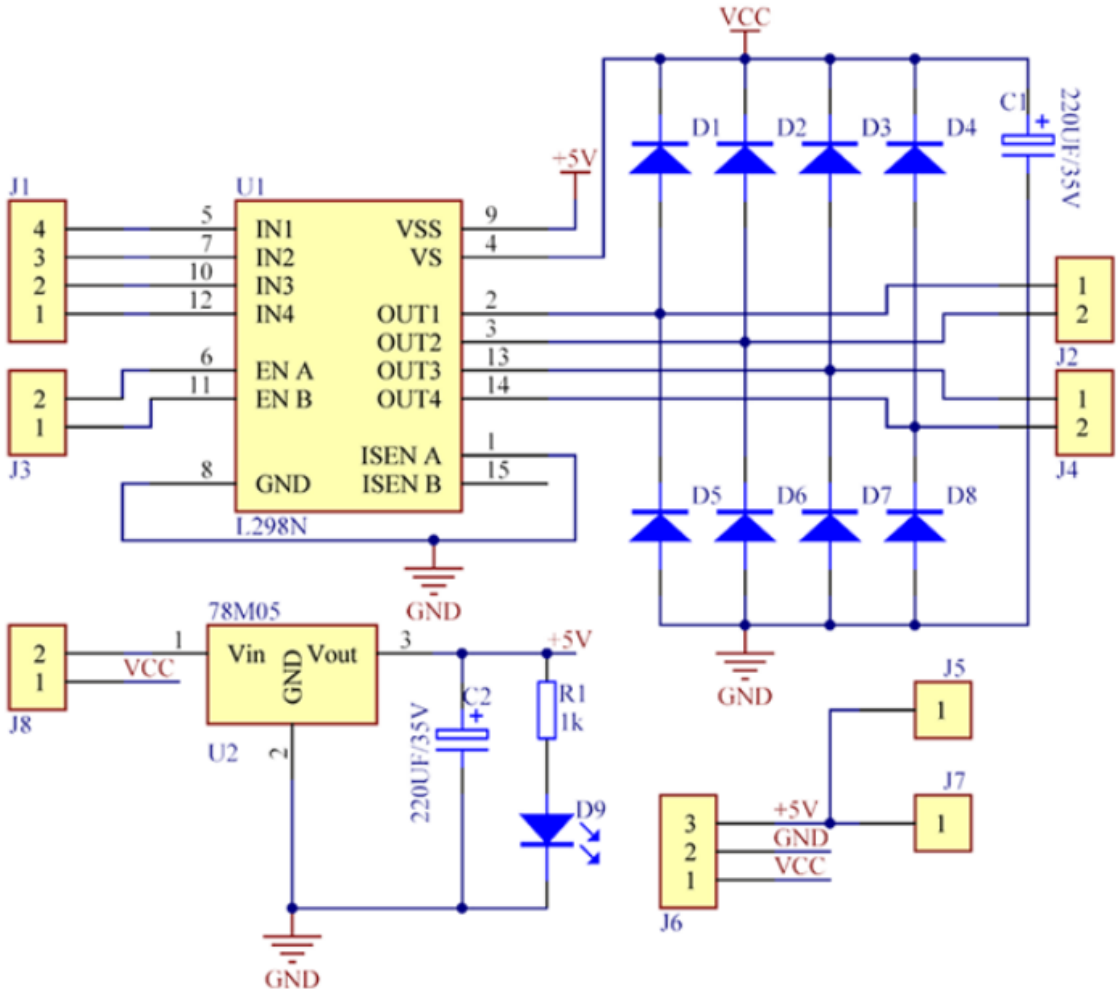
En ulempe med L298N er at IC-chippen ikke er laget for å tåle mer enn 2 A per kanal.

IC-chippen har to kanaler som kan kobles i serie, og dermed levere opp til 4 A. Flere drivere kan kobles i paralell for å gi enda mer strøm om nødvendig.

Et forenklet kretsskjema av det interne systemet på chippen er vist i fig. 6.14. L298 har to BJT H-broer, hver av dem styrt av differensiell PWM, med separate stop-signal som også kan styres med PWM.



Figur 6.14: Blokkdiagram/forenklet kretstegning for L298 IC-chip[16].



Figur 6.15: Kretstegning for L298N driver[15].

Fig. 6.15 viser kretstegning for det komplette driverkretskortet som er utstyrt med L298N. Dette ble brukt i den første iterasjonen av girkontroller-PCB'en for Arduino Uno. I 2. iterasjon ble driveren plassert på kretskortet sammen med kontrolleren.

6.4 Jam-detektor

Siden giret kan henge seg opp, trenger vi å kunne detektere om aktuatorens bevegelse hindres, slik at den kan stoppes for å unngå å påføre skade på girstag eller girkasse.

6.4.1 Programmerte konsepter

Det ble vurdert flere programmerte løsninger for å detektere jam. Dette ble utviklet uten hensyn til maksimal kraft på girstag, men kun for å kunne detektere hindring av

aktuatorbevegelse. Det første konseptet gikk ut på å se om avviket (differansen mellom posisjon og referanse) overstiger en satt grense i mer en angitt tidsperiode. Det neste, mer sofistikerte konseptet brukte en simulert hastighet for aktuatoren, og utførte den samme operasjonen med avviket mellom den egnetlige hastigheten og den simulerte. Disse ble senere vraket til fordel for en analog overstrømsdetektor, fordi de viste seg å være unøyaktige, uforutsigbare og treige.

6.4.2 Analoge konsepter

Tre forskjellige kandidater ble vurdert som konsept for jam-detektoren. Alle de analoge konseptene går alle ut på å detektere overstrøm gjennom motordriveren.

Når aktuatoren er i bevegelse vil dette føre til EMS som induseres over motorspolen. Dette reduserer strømmen gjennom motoren. Det vil si at dersom aktuatoren blir holdt igjen av en hindring, vil ikke denne EMS'en bli indusert og strømmen gjennom motoren blir høyere enn normalt. Dermed kan vi anta at det går an å detektere om aktuatoren har møtt en hindring ved å måle strømmen gjennom motoren, som igjen går gjennom driveren.

Med EM208, kunne vi benytte oss av den innebøygde strømgrensen i kontrolleren, som da slo ut når aktuatoren møtte en hindring. Ettersom L298N ikke har en slik innebygd funksjonalitet, så trenges det å legge til en ekstra krets.

Vi har kommet frem til at maksimal kraft som kan påføres girstaget er 74 N, før vi risikerer skade ved langvarig bruk (se kap. 5.2.9). Vi kan approksimere strømmen som må til i motorpådraget for å føre til en slik kraft til å være ca. 1.2 A, hvis vi antar at kraften er proporsjonell med pådraget, og at maksimal strøm tilsvarer maksimal kraft som gitt i datablad for LA14. Dette er en antagelse som sannsynligvis ikke er helt riktig, men vi kan uansett anta at en økt strøm fører til en økt kraft på girstag, og at strømgrensen vi ønsker å sette er omtrent i dette området.

Det første konseptet baserer seg kun på en enkel selvnullstillende sikring, mens de to andre bruker en aktiv krets for å måle strøm som deretter sammenlignes med en satt terskel. Vi endte opp med å bruke det siste konseptet.

6.4.2.1 PPTC på VCC/jord

Dersom en PPTC[7] benyttes kan det måles om PPTC'en har slått ut. Deretter vil PPTC'en nullstille seg etter noen sekunder. PPTC kan kobles til VCC, til jord, til SENSE-pinnene eller i serie med motor og vil bryte strømmen når den går over den valgte terskelen. Et problem med denne metoden er at PPTC'er bruker et par sekunder på å slå ut, som er tregere enn ønsket. En fordel er at kretsen ved bruk av et slikt komponent blir særdeles enklere i design.

6.4.2.2 Strømmåler på VCC/jord

Dersom VCC eller jord brukes for strømstopp-kretsen vil det ikke lenger være behov for å modifisere driverens ferdiglagde kretskort, men vi vil da også måle strømmen som går til de andre komponentene på kortet, som for eksempel den 5 V spenningsregulatoren som bestemmer nivået for kontrollsignal til driveren. Ved simulasjon har det blitt estimert at denne spenningsregulatoren trekker rundt 5 mA, som vil gi et neglisjerbart avvik i målingen.

Dette er konseptet vi bruker for iterasjon 1 av kretskortet, ettersom vi der bruker et ferdig, eksternt kretskort for driveren L298N.

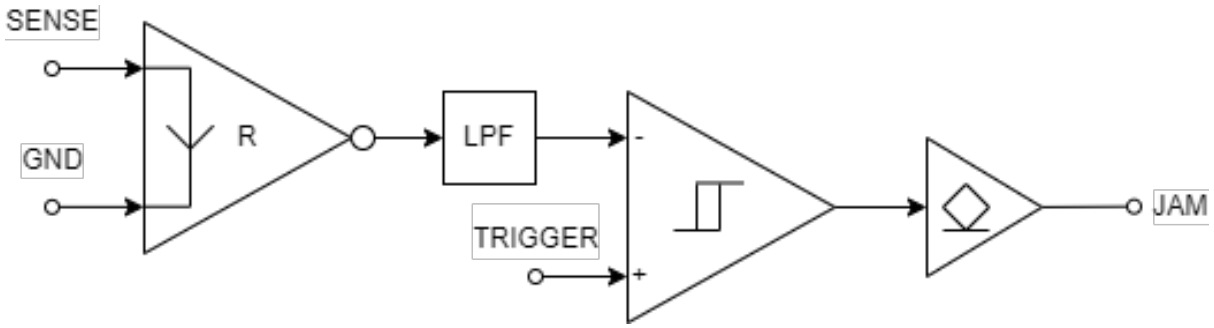
6.4.2.3 Strømmåler på SENSE-pinnene

Som vist i fig. 6.15 er pin 1 og 15 (SENSE A og SENSE B), vist i fig. 6.14, koblet direkte til jord på driverens kretskort. Å kunne måle strømmen via disse pinnene er ønskelig, og må dermed foreta noen modifikasjoner på driver-kretskort slik at strømmen ikke kan overstige 4 A og at strøm over en satt grense (som må være under 1.27 A) kan detekteres, dersom man ønsker å bruke SENSE-pinnene for strømstopp-kretsen.

Dette konseptet har blitt brukt i det endelige kretskortet for Portenta H7, fordi vi der har driveren montert på et kretskort av eget design, og har dermed SENSE-pinnene tilgjengelige.

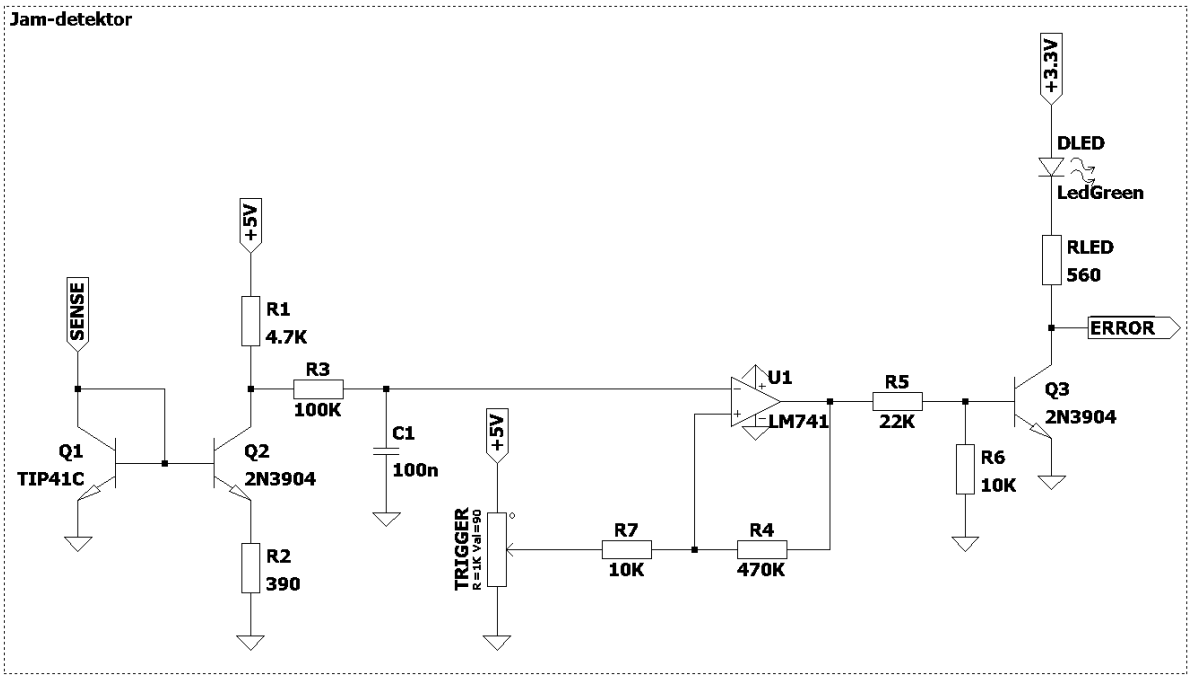
6.4.3 Design

[7] "Polymeric Positive Temperature Coefficient Device", en form for selvnullstillende sikring



Figur 6.16: Blokkdiagram for design av jam-detektor.

Det konseptuelle designet er illustrert i fig. 6.16. Figuren viser en strømmåler markert med R, som gir en analog spenning deretter sendt gjennom et lavpassfilter, og deretter sammenlignet med en satt terskel, hvor resultatet til slutt blir sendt ut til mikrokontrolleren via en open-collector buffer på samme måte som med EM208-driveren (se kap. 6.3.4). Mikrokontrolleren har da som ansvar å stoppe pådraget dersom en strøm over den satte grensen er detektert. Komparatoren må ha en schmitt-trigger, slik at ikke rippel på målingen fører til jitter i 'jam'-signalet.



Figur 6.17: Kretstegning for overstrømsdetektor for å oppdage "jam".

Den ferdige kretstegningen er vist i fig. 6.17.

Design av strømmåleren tar utgangspunkt i et BJT strømspeil, men hvor da strømmen skaleres ned i utgangstrinnet. Resultatet er en spenning som er negativt porporsjonal med

strømmen tilført aktuatoren.

Ettersom denne strømmen har en puls-form må den utgjevnes med et lavpassfilter før sammenligning. Her ble et 1. ordens RC-filter med knekkfrekvens på 31 Hz brukt.

For schmitt-trigger komparatoren, ble en op-amp med positiv tilbakekobling benyttet. Terskelen er satt med en trimpot som kan stilles inn med skrujern på PCB.

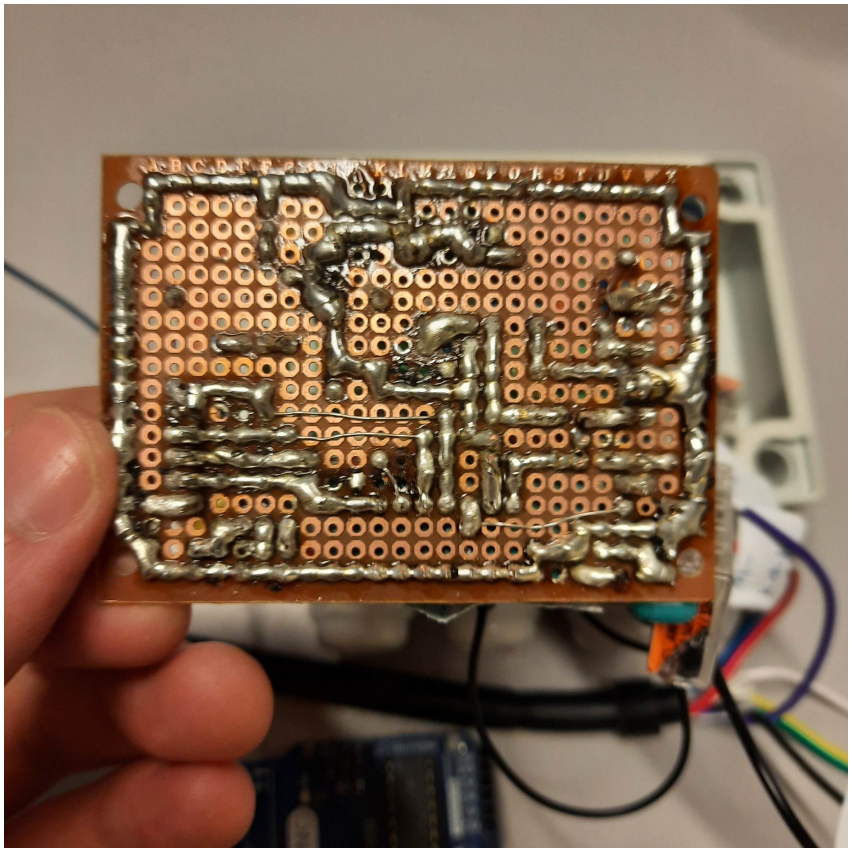
Systemet har blitt testet med fiskevekt festet til aktuatorstempel og terskel har blitt innstilt slik at maksimal kraft ikke overstiger 74 N. I etterkant av dette har det blitt fininstillt for å respondere riktig ved giring med motor på på ATV, uten å overstige kraftgrensen.

6.5 Kretskort

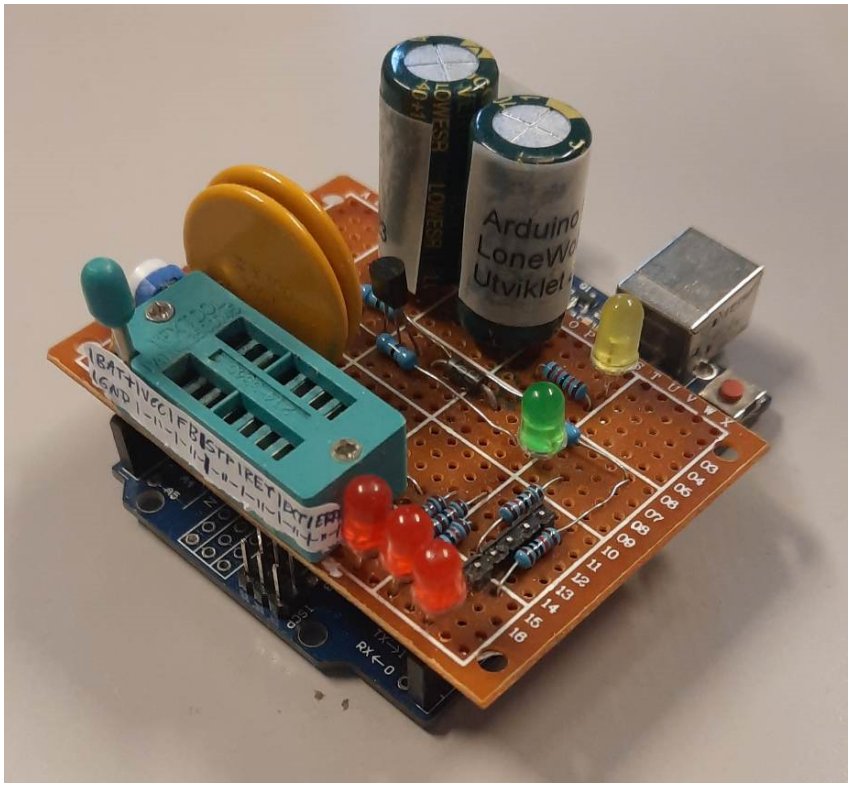
Tre forskjellige kretskort har blitt designet og fabrikkert gjennom prosjektet. Kretskortene for Arduino Uno er navngitt LW-GCA1, og kretskortet for Portenta H7 er navngitt LW-GCA2. Alle kretskortene er designet for "through-hole"-komponenter, slik at vedlikehold og reparasjon er enklest mulig å utføre.

6.5.1 Prototype for LW-GCA1 håndloddet perfboard

Den første prototypen ble loddet og rutet for hånd, på perfboard, og var uten innebygd jam-detektor. Jam-detektoren ble utviklet i etterkant av prototypen, og ble inntil videre koblet opp på breadboard. Dette kortet ble laget for bruk med EM208, før vi gikk over til L298N, og er koblet opp som vist i kretstegning i fig. 6.10.



Figur 6.18: Undersiden av protype-kortet for LW-GCA1.

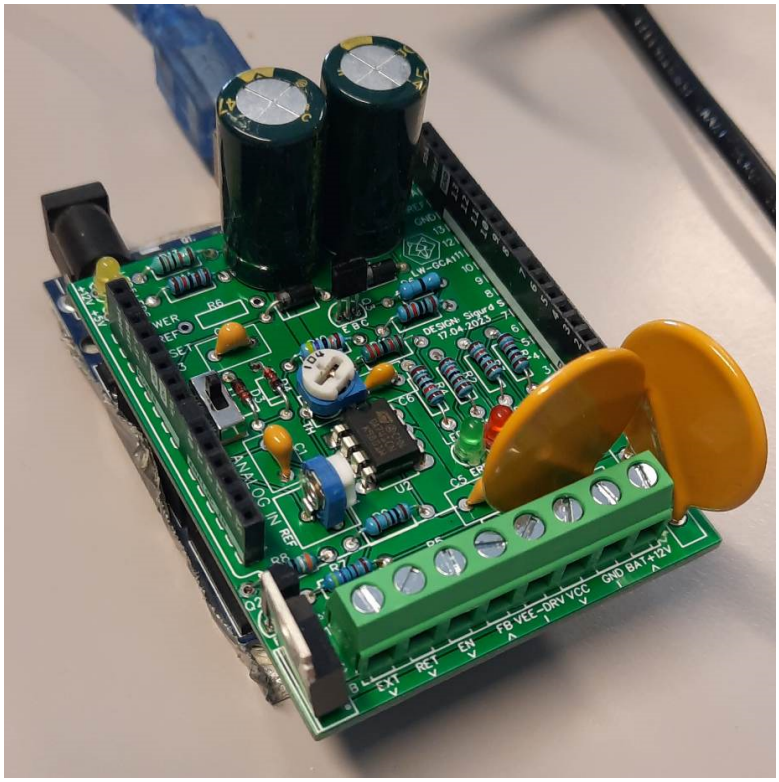


Figur 6.19: Protype-kortet for LW-GCA1 sett ovenifra.

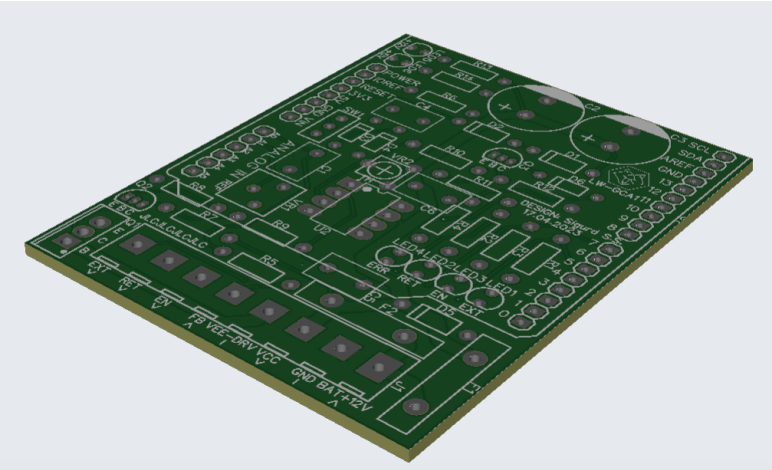
Fig. 6.18 viser prototypen sett nedenifra, mens fig. 6.19 viser kortet ovenifra montert på Arduino Uno. Kretskortet inneholder alle elementer som er på det ferdige LW-GCA1-kortet, unntatt jam-detektoren. For å koble på ledninger ble det brukt en rekkeklemme med spak, noe som viste seg å være et dårlig designvalg, men var den eneste typen rekkeklemme som var tilgjengelig på dette punktet. Ledninger hadde en tendens til å sitte løst, og dette ut av sporet i rekkeklemmen. Dette var uansett en forbedring fra breadboard-implementasjonen, hvor da ledningene hang løst og færre uhell oppsto som resultat.

6.5.2 LW-GCA1 printet kretskort for Arduino Uno

Etter at prototypen ble laget, jam-detektoren ble ferdig utviklet og L298N ble fastslått som den endelige motordriveren for prosjektet, ble det designet og bestilt et printet kretskort. Rutingen ble da gjort av kretskortfabrikanten og loddearbeidet ble betydelig enklere. Dessuten var dette kretskortet utstyrt med rekkeklemmer med skruer, og innebygd overstrømsdetektor beskrevet i kap. 6.4.

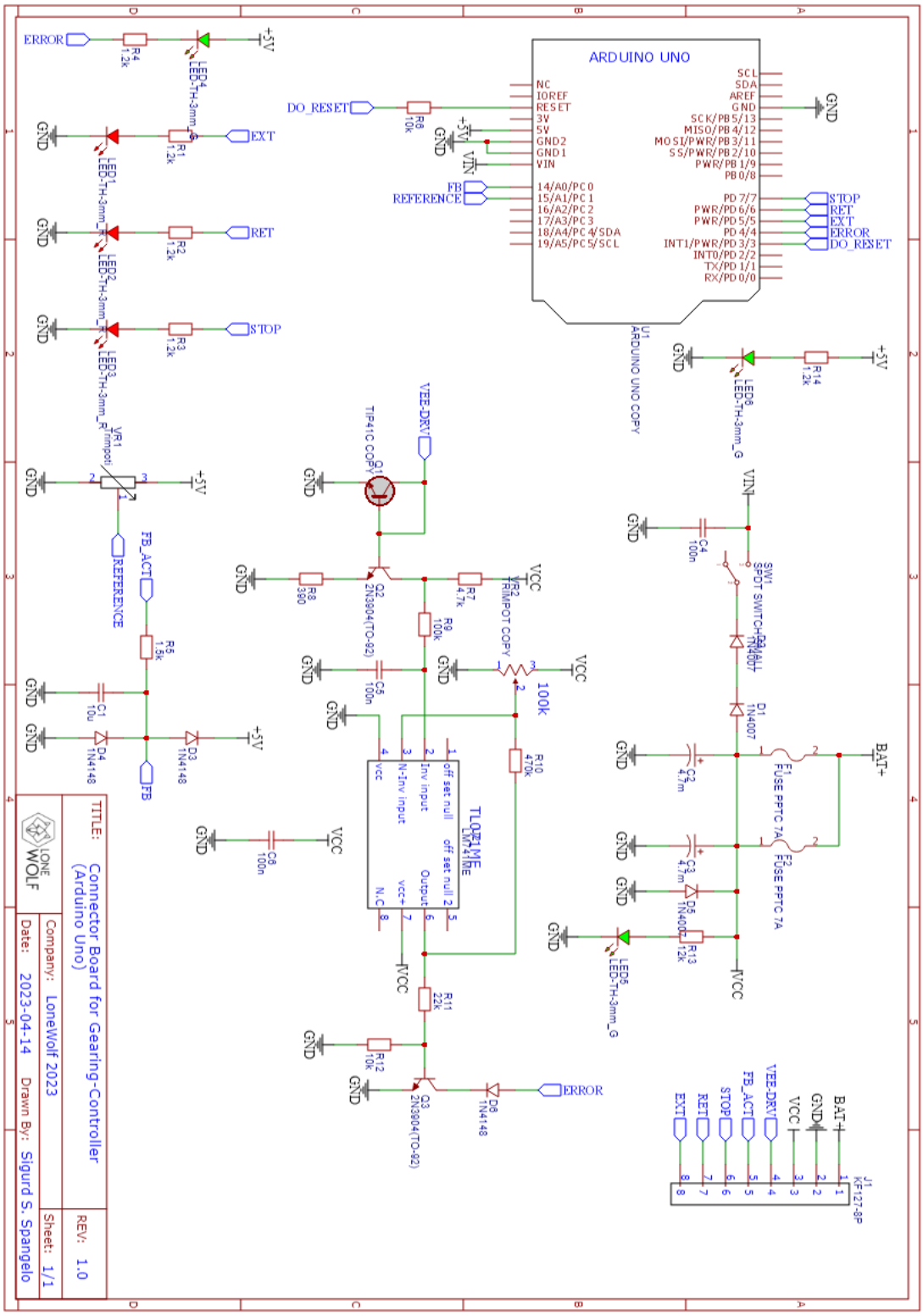


Figur 6.20: LW-GCA1 sett ovenifra montert på Arduino Uno.



Figur 6.21: 3D-render av kretskortet for LW-GCA1.

Det endelige resultatet med alle komponentene loddet på kortet er vist i fig. 6.20, montert på Arduino Uno. En 3D-render av kortet er vist i fig. 6.21.



Figur 6.22: Komplett kretstegning for LW-GCA1.

Kretsen for LW-GCA1 som vist i fig. 6.22 er basert på den tidligere utgaven vist i fig. 6.10, men har nå en innebygd overstrømsmåler. De røde LED'ene indikerer signalene som blir sendt til driveren. Den midterste lyser når det blir sendt et "enable"-signal til driveren, mens de to andre lyser ved fremover- og bakover-signal. Den grønne LED'en vil lyse når jam-detektoren slår ut. De gule LED'ene indikerer at forsyningsspenning er koblet til, med én LED hver for 5 V fra USB-tilkobling og 12 V fra batteriet.

Kretskortet viste seg å være en enorm forbedring fra den tidligere prototypen, spesielt ettersom at jam-detektor er inkludert og ledninger er trygt festet i rekkeklemme og dermed også lettere å montere.

En ulempe med dette kortet er at det ikke får plass inni de vanntette IP66-boksene som LINAK-driverne kommer i, og må dermed puttes inne i skap på ATV for å tilfredsstille krav om IP44-sikret elektronikkmoduler.

Vi ønsket også å gå over til Portenta H7 med Micro-ROS fremfor Uno med Rosserial, så denne utgaven av systemet er simpelthen noe å falle tilbake på som en midlertidig løsning. Senere i et fremtidig LoneWolf-prosjekt er det ønskelig å migrere totalt over til Portenta H7 for å unngå å bruke ROS-bridge.

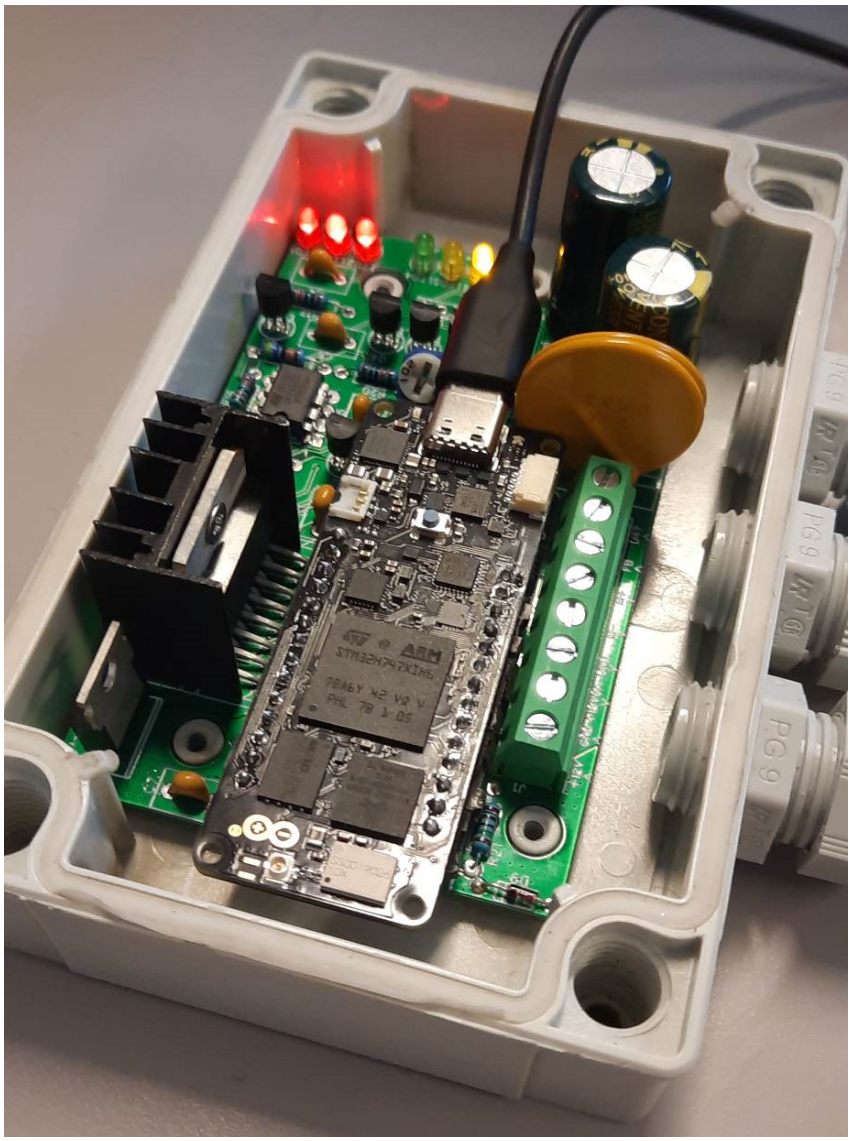
6.5.3 LW-GCA2 printet kretskort for Portenta H7

Et nytt kretskort ble til slutt designet for bruk med Portenta H7. På kretsen har det blitt gjort noen få justeringer.

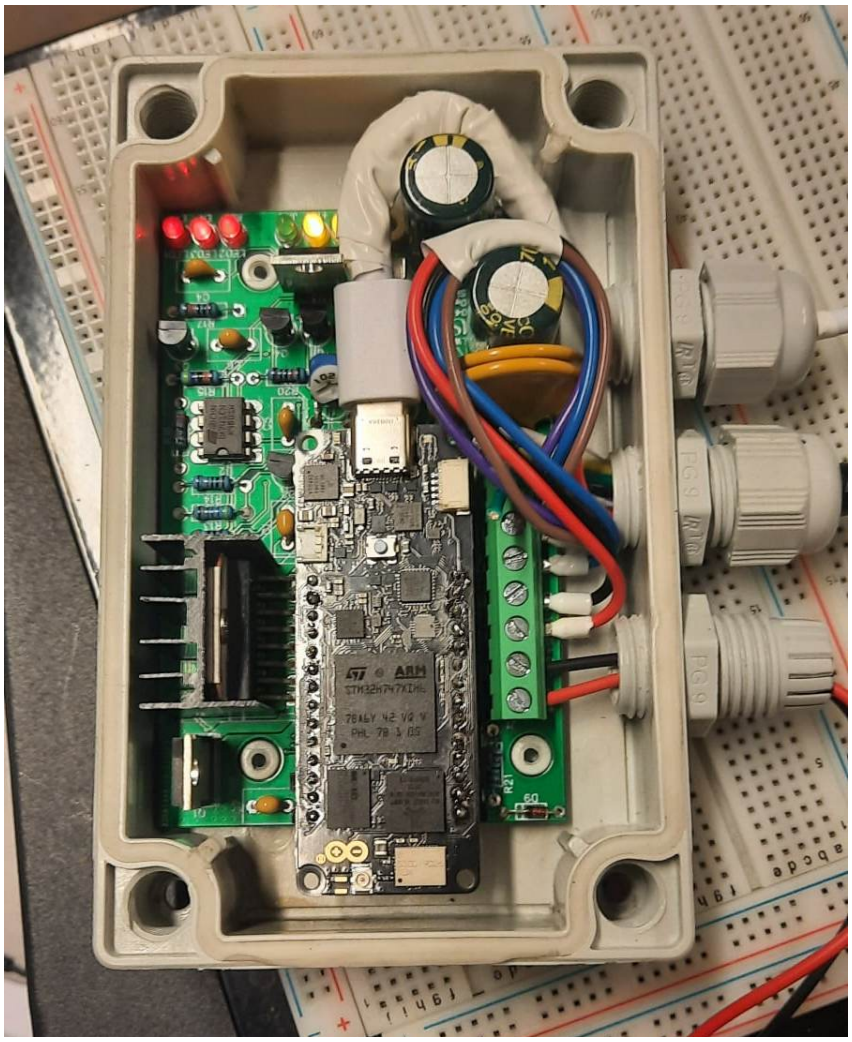
Portenta H7 opererer med logikkspenning på 3.3 V. Dette krever ekstra komponenter som kan forsterke logikksignalene opp til 5 V slik at de kan brukes med L298N. Analog tilbakekobling fra LA14 må dessuten attenueres ned fra 5 V til 3.3 V.

Trimpotten for analog styring av referanse har nå blitt tatt vekk, ettersom det ikke lenger er behov for den.

I stedet for å montere kortet på mikrokontrolleren er nå mikrokontrolleren montert på kortet i en DIP-28 sokket.



Figur 6.23: LW-GCA2 loddet ferdig, og demonstrert koblet til PC med USB. Hele kretsen får plass inni den vanntette boksen.



Figur 6.24: LW-GCA2 koblet opp og under testing.

Det ferdige kretskortet er vist i fig. 6.23-6.24, hvor den i fig. 6.24 også er koblet opp til bruk med LA14 og med ekstern 12 V strømforsyning.

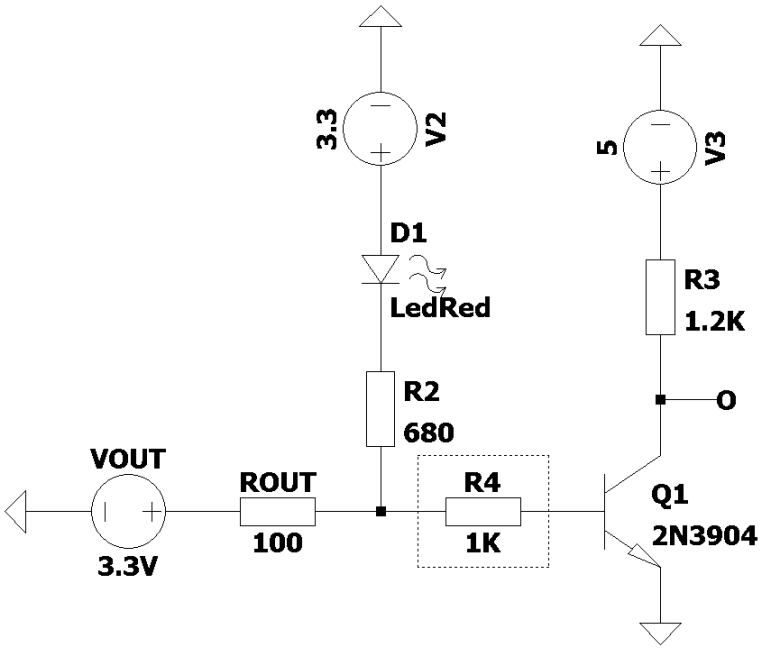
Å få kretskortet til å passe inni boksen viste seg å være en stor utfordring. På grunn av høyden på utglattingskondensatorene og L298N, måtte kortet ligge så tett innpå bunnen av boksen som mulig for å kunne få igjen lokket, og unngå kontakt mellom heteskjold og lokk. For å få plass til alle komponentene ble mesteparten av motstandene i kretsen plassert under mikrokontroller-kortet. Dette førte til suboptimal routing for en del av banene på kretskortet, som må dermed ta en del omveier.

Å ha driveren på samme kort som mikrokontrolleren ga også utfordringer, med da mindre plass til overs, og behov for tjukkere baner som kan levere nok strøm til motor.

Kretskortet måtte designes på kort varsel, ettersom det var lite tid igjen på prosjektet når

Portenta H7 ble levert. Selv om kretsen fungerer slik den skal, så er det rom for forbedring.

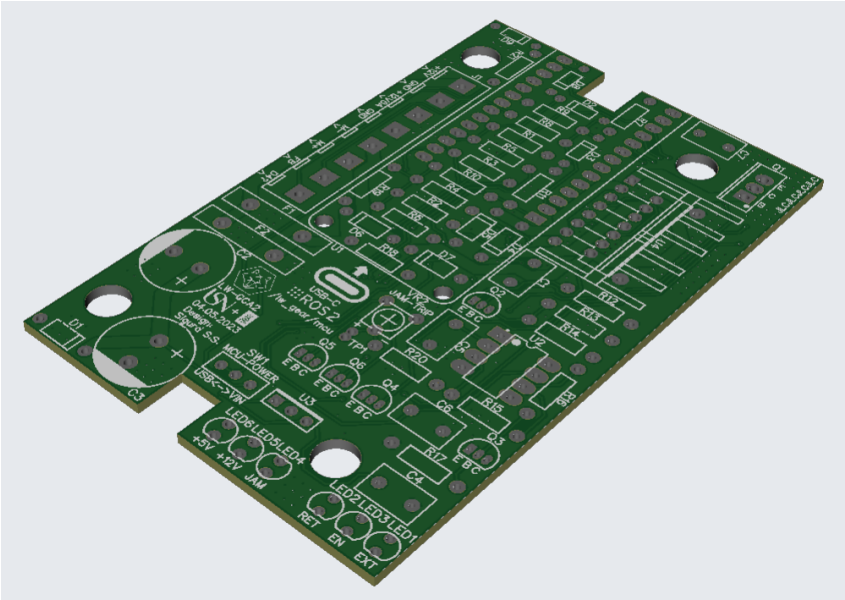
Det ble i etterkant av produksjon, oppdaget en designfeil med dette kretskortet. BJT-forsterkningsleddene for kontrollsignalene kunne vært designet bedre. Slik de er nå, er utgangspinnene koblet direkte til base på BJT, og emitter koblet direkte til jord, noe som gjør at utgangsspenningen blir begrenset av fremoverspenningen til transistoren, som ikke er optimalt. Det kan hende at dette trekker mer strøm enn nødvendig fra utgangspinnene på mikrokontrolleren. Dette førter også til at de røde indikator-LED'ene aldri skrur seg helt av, men bare lyser litt svakere når utgangssignalet (aktiv-lav) er satt til høy. Det viste seg også at frekvensresponsen fra motor-signalene til motordriveren er uhyre redusert i forhold til LW-GCA1. PWM-frekvensen på "extend"- og "retract"-signalene har blitt skrudd ned til 10 Hz for å kunne fungere slik de skal. Det kan være at disse to problemene er forårsaket av samme feil på kretsen. For å løse dette burde det plasseres en motstand mellom digitalutgang og base på transistoren som vist i fig. 6.25. Kretskortet er ikke tilrettelagt for dette, så videre revisjon av kretskortet er ønskelig.



Figur 6.25: Forbedring av utgangstrinn for Portenta H7 på LW-GCA2, for fremtidig revisjon. Den anbefalte modifikasjonen er markert med en striplet boks (R4).

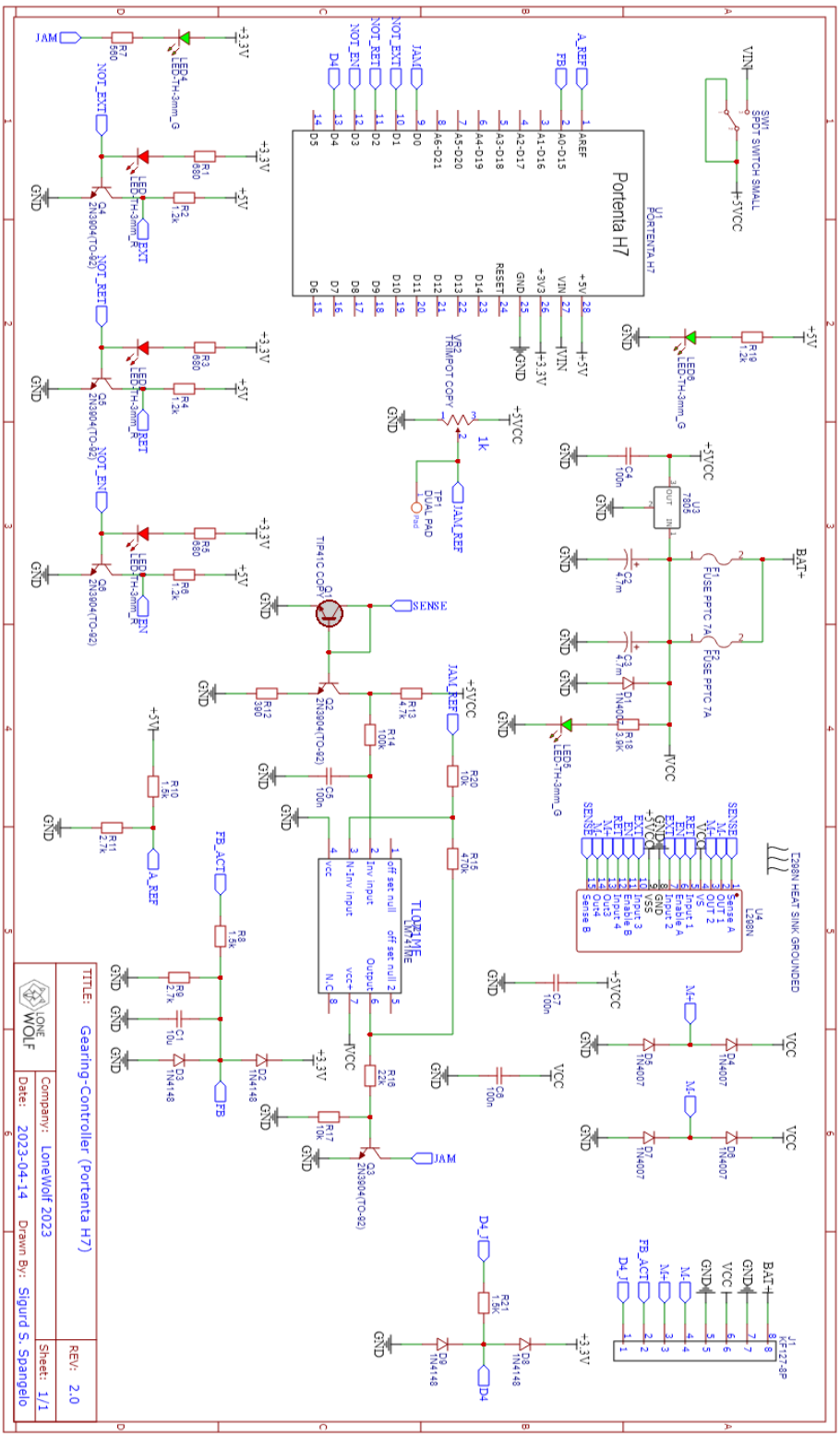
Det er også mulig at banene på kretskortet er for lange, og fører til unødvendig høy kapasitans på vei bort til driveren. Dette burde undersøkes videre. Det er mulig at et kretskort med flere etasjer kunne gjort det enklere å rute kretsen med kortere baner over

et mindre areal.



Figur 6.26: 3D-render av PCB for LW-GCA2.

Fig. 6.26 viser kretskortet for LW-GCA2 uten komponenter. Mesteparten av motstander og dioder i kretsen er plassert under mikrokontrolleren for å spare plass. Kretskortet har hull og utklipp for å kunne få plass inni boksen og gi rom til skruefester.



Figur 6.27: Komplett kretstegning for LW-GCA2.

Hele kretstegningen for LW-GCA2 er illustrert i fig. 6.27. Her er referansen på overstrømdetektoren nå stabilisert med en spenningsregulator, noe som gjør jam-detektorterskelen mye enklere å stille inn riktig. Uten spenningsregulator dipper terskelspenningen ned samtidig som målespenningen. BJT-forskerkningsledd er satt på de digitale utgangene på Portenta H7, for å kunne interagere med L298N med 5 V logikkspenning. Ellers er kretsen stort sett lik den for LW-GCA1, men med noen resistor-verdier justert. Utgangssignalene er her aktiv-lav i stedet for aktiv-høy som med det tidligere kretskortet.

6.5.4 Gerber-filer

Gerber filer for PCB for LW-GCA1 og LW-GCA2 ligger i vedlegg A47 og A48. JSON-filer til bruk med EasyEDA er også med i vedlegg A49 og A50.

6.6 Måling av gir-punkter/referanser

Gir-punktene har blitt målt både i form av referanseverdier og i utslag (mm), vist i tabell 6.2. Utslagslengde har blitt målt med skyvelære fra kropp på aktuator til ytterste tupp på stempel.

Gir	r	**Utslag**
P	0.04778	53.8 mm
R	0.25635	75.77 mm
N	0.43100	87.1 mm
H	0.54911	107.6 mm
L	0.72500	130 mm

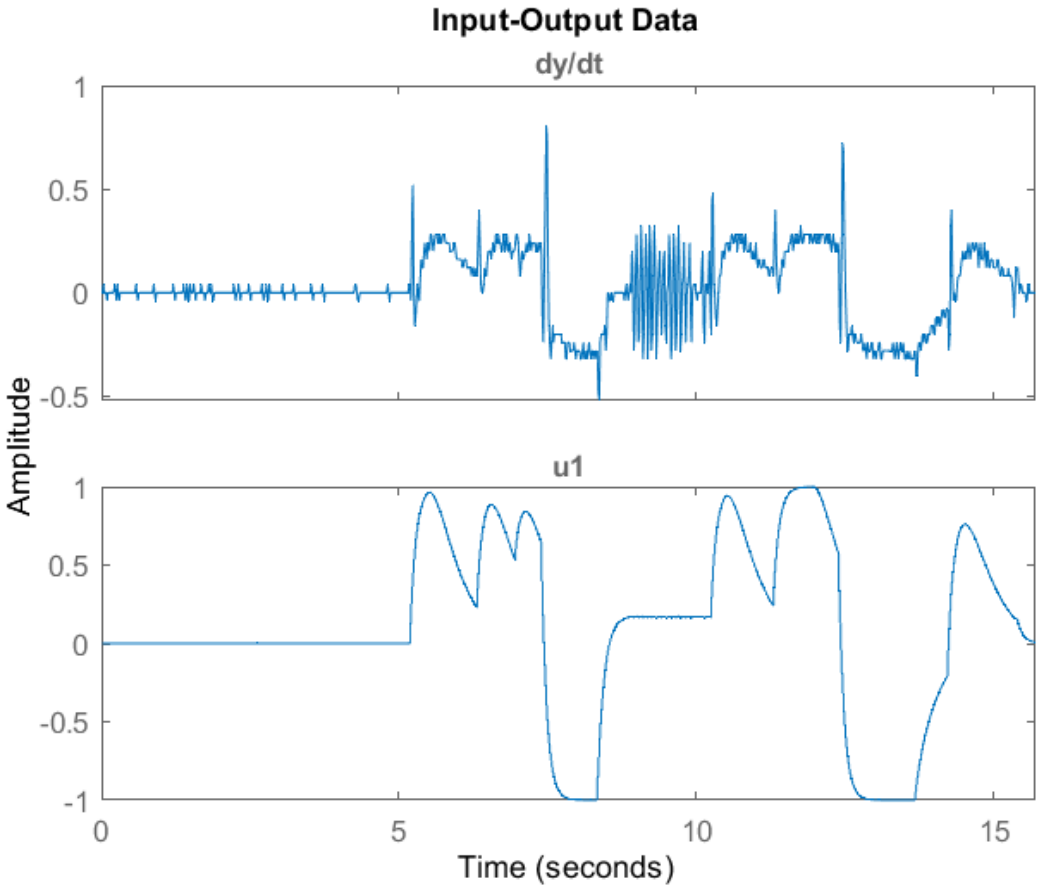
Tabell 6.2: Målte gir-punkter i referanse r og utslag i mm

6.7 Modell av aktuator

Aktuatorens frekvensrespons har blitt målt for å kalibrere PID-kontroller og for å tilrettelegge seinere modellering av aktuator for simulering med Gazebo. Transferfunksjonen har blitt estimert ved hjelp "tfest"-funksjonen fra Matlab's System Identification Toolbox.

Posisjonsdata er målt ved å skru på ENABLE_MEASURE_IR i libserial-branch for

gir-Arduino-koden, byttet referanse gjentatte ganger, og deretter kopiert måledata fra seriellportvindu til Matlab. Denne dataen består av pådrag og posisjon målt i et fast tidsintervall, og en gjennomsnittlig sample-frekvens. Posisjonsmåling blir deretter derivert før den mates inn i "tfest". Et måleresultat oppnådd med denne metoden er vist i fig. 6.28.

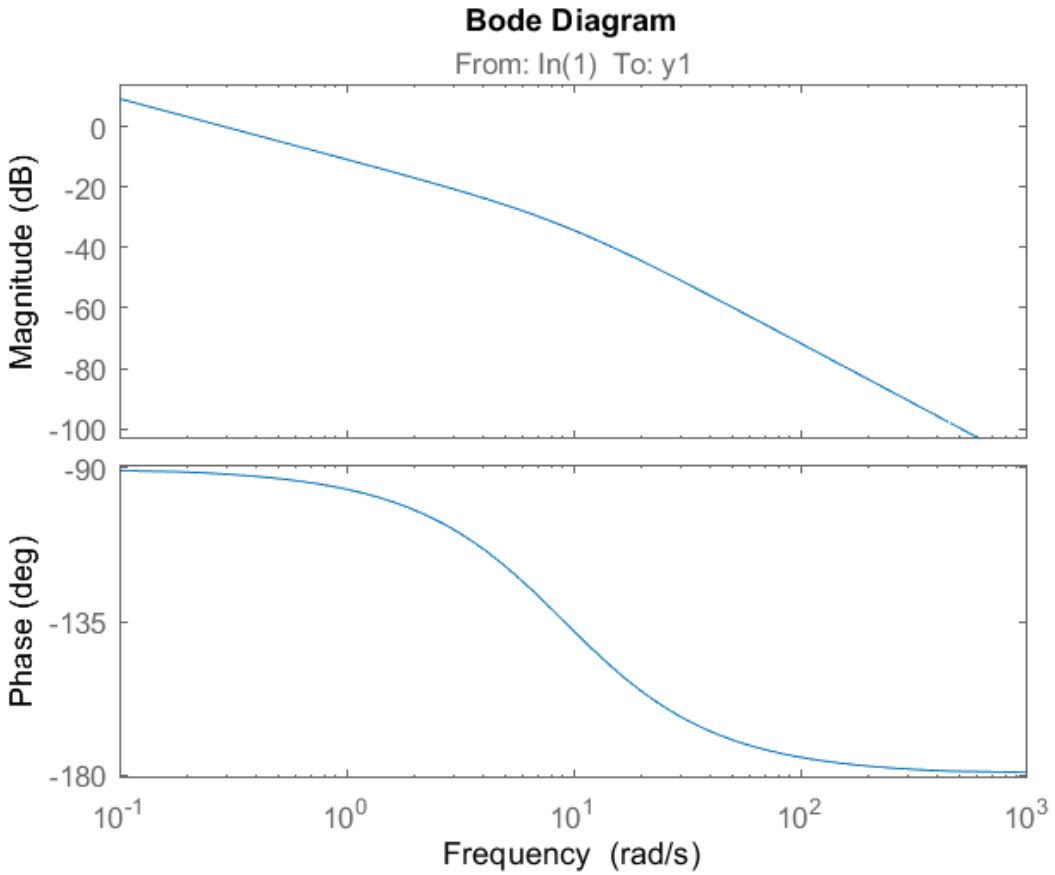


Figur 6.28: Måledata ved sporadisk bytting av gir. Øverste kurve viser den deriverte $\frac{dy}{dt}$ av utslaget og nederste kurve viser pådrag u. Målingen er tatt i et tidsrom på ca. 16 s. Det er en betydelig mengde med støy på den deriverte utslagsmålingen.

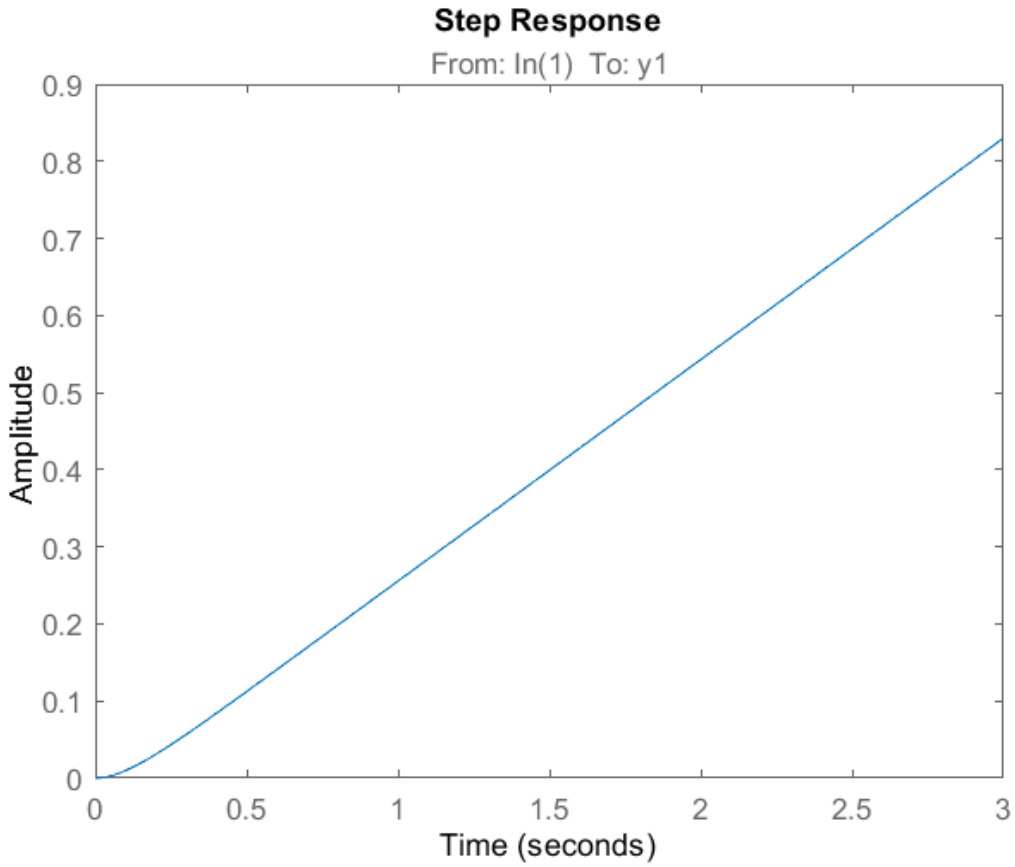
Ved "tfest" har gruppen fått følgende estimat på aktuatorens transferfunksjon:

$$G(s) \cong \frac{2.612}{s^2 + 9.094s}, \tag{6.6}$$

hvilket gir en frekvensrespons vist i fig. 6.29 og en stegrespons vist i fig. 6.30.



Figur 6.29: Bode-plot $G(j\omega)$ av den estimerte transferfunksjonen til aktuatoren $G(s)$.

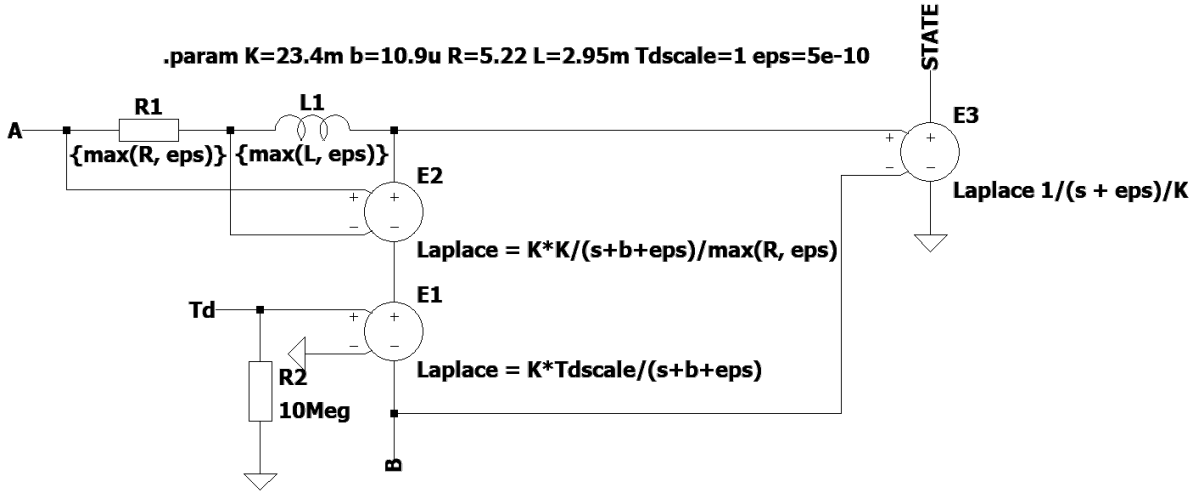


Figur 6.30: Stegrespons $u * g$ av den estimerte transferfunksjonen til aktuatoren $G(s)$. Dette er nesten en rett linje, som tilsvarer en integrator. Det kan sees at ved steg-pådrag bruker aktuatoren rundt 0.1 s på å komme opp i fart, før den etterhvert har en integrator-aktig respons.

Ved å implementere likning 6.6 som et diskret-tid sanntidsfilter (Discrete Time Real-Time Filter), med metning ved posisjon 0 og 1, vil man kunne simulere aktuatorens adferd.

6.7.1 LTSpice-modell av aktuator

En generell DC-motormodell har blitt konstruert i LTSpice. Denne kan brukes til å lage en LTSpice modell av LA14 for å kunne simulere aktuatoren. Dersom det er ønsket å gå videre med en motordriver av eget design, er dette noe som også burde gåes videre med for å kunne simulere driveren med aktuator koblet til. Kretstegning til DC-motormodellen er vist i fig. 6.31.



Figur 6.31: Generell LTSpice-modell av en DC-motor. A og B er terminalene på motoren. Td er forstyrrelsesmoment på motoraksling. STATE er rotasjon på motoraksling relativ til startposisjon i radianer.

DC-motormodellen er laget utifra formler for DC-motor i [17], som vist i ligning 6.7 (ligning for indusert spenning) og 6.8 (ligning for kraftmoment):

$$\varepsilon = k_e \omega, \tag{6.7}$$

hvor ε er indusert/elektromotorisk spenning (ems), k_e er konstant for elektromotorisk spenning (ems) og ω er turtall (rotasjonshastighet), og

$$\tau = k_i i - b\omega + \tau_c + \tau_d, \tag{6.8}$$

hvor τ er kraftmoment/dreiemoment, k_i er konstant for kraftmoment/dreiemoment,

i er strøm, b er dynamisk friksjonskonstant, τ_c er statisk friksjon, ω er turtall (rotasjonshastighet) og τ_d er forstyrrelse.

6.8 Uhell

Gjennom prosjektet har det oppstått et par uforutsette elektriske uhell.

6.8.1 Kortslutning

Under en test av gir-system på ATV hadde ledning fra batteri til breadboard løsnet og fikk kontakt med en jordet kobling på breadboardet. Dette førte til at et par jumpere/0 Ω resistorer tok fyr. Kabelen ble kjapt nappet ut og nødstopp på ATV ble trukket inn. Tilstrekkelig reservedeler var tilgjengelig på stedet, og problemet ble fort diagnostisert og løst.

6.8.2 Overbelastning av spenningsregulator på Arduino Uno

Under finjustering av gir-kontroller ble EA-3003S spenningsforsyning skrudd på. Ved kontakt av USB-port på Arduino fikk Sigurd et støt, og kort tid etterpå begynte det å ryke fra Arduino Uno-brettet. Strømmen ble skrudd av og deretter ble Arduino-brett inspisert. I dette tilfellet hadde 5 V spenningsregulator på Arduino-brettet smeltet og var ute av drift. Brettet fungerte fortsatt, men kunne ikke lenger forsynes av 12 V spenning fra EA-3003S via "Vin"-port. Etter granskning av datablad for Arduino Uno[18] ble det funnet ut at 12 V er helt på grensen av anbefalt spenning til spenningsregulator. En lavere spenning på min. 7 V er å foretrekke for å øke levetiden til spenningsregulatoren[18]. Kobling til "Vin" på Arduino ble dermed modifisert med to dioder for å oppnå et spenningsfall.

6.8.3 Smeltede ledninger og driver

Ledningene vi brukte mellom kretskort for Uno-iterasjonen av girkontrolleren til motordriveren L298N, var for tynne til å tåle det aktuatoren maksimalt kan trekke (5 A). Under test på ATV viste det seg at referanseverdien for lavgiret var feilinstilt, og aktuatoren prøvde å trenge seg for langt fremover. Naturligvis slo jam-detektoren ut, og girforsøket ble avbrutt. Jam-detektoren ble midlertidig skrudd av for å feilsøke problemet, og dette førte til at aktuatoren fortsatte å dytte, og endte opp med å trekke mer strøm

enn ledningene var laget for å tåle. Ledningene til driveren gikk opp i røyk, og nødstopp ble øyeblikkelig slått inn. Dette førte også til at L298N ble satt ute av drift og måtte byttes ut med en reserve. Etter dette ble det brukt tykkere ledninger for driveren.

7 Software

Som dataingeniør er det viktig å ha evnen til å videreutvikle på eksisterende systemer. Følgende steg kan være med på en systematisk og effektiv gjennomføring av videreutvikling.

1. Analyser det eksisterende systemet: For å utvikle videre på et eksisterende systemet, er det nødvendig å forstå hvordan systemet fungerer, hvilke begrensninger det har og hvilke problemer som kan påvirke videreutviklingen?

2. Identifiser kravene til det nye systemet: Nye krav må defineres for hva som er nødvendig å oppnå med det nye systemet. Hvilke funksjonalitet trengs, og hvordan skal det samhandle med det eksisterende systemet.

3. Planlegg utviklingen på eksisterende system: En plan for implementasjon for de nye funksjonene eller eventuelle endringer på det underliggende systemet må gjøres rede for. Hvilke teknologier og verktøy skal benyttes?

4. Utvikle og teste: Koden må utvikles for det nye systemet, og det må testes grundig i form av enhetstesting og integrasjonstesting for å sikre at systemet oppfører seg som forventet og integreres godt med det underliggende systemet.

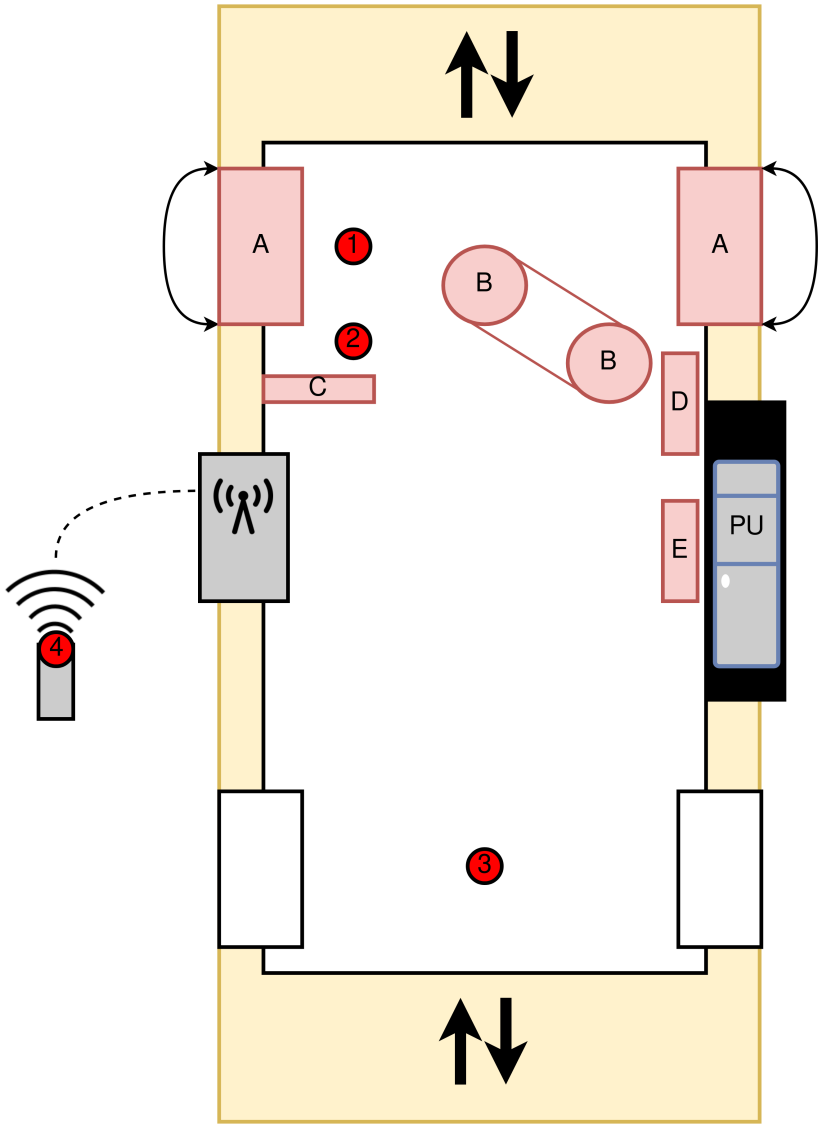
5. Ferdigstill det nye systemet: Når det nye systemet er ferdig utviklet, implementert og testet, kan det ferdigstilles. Det kan også være lurt å gjøre til rede for mulig fremtidig utvikling. Hvilke tiltak må til for å sørge for at systemet har en langvarig levetid, fungerer stabilt og effektivt over tid.

7.1 Analyse av tidligere system

Lone Wolf er et pågående prosjekt med oppstart i 2018, og gjennomføres av studenter for Kongsberg Defence & Aerospace (KDA). Prosjektet har tatt for seg simulering av ATV i simuleringsmiljøet Gazebo, fjernstyring av ATV fra en operatør stasjon via en UM600 UHF radio, se Appendiks A51, kontroll av gasspådrag gjøres ved hjelp av en Digital-to-analog Converter (DAC), brems aktueres av lineær aktuator og et system for sving styres ved hjelp av en stepper motor. Tidlig i prosjektet ble et tankekart laget som beskriver ATV og dets funksjonalitet, se Appendiks A46. Underkapitlene beskriver dataingeniør-studentenes analyse av tidligere system.

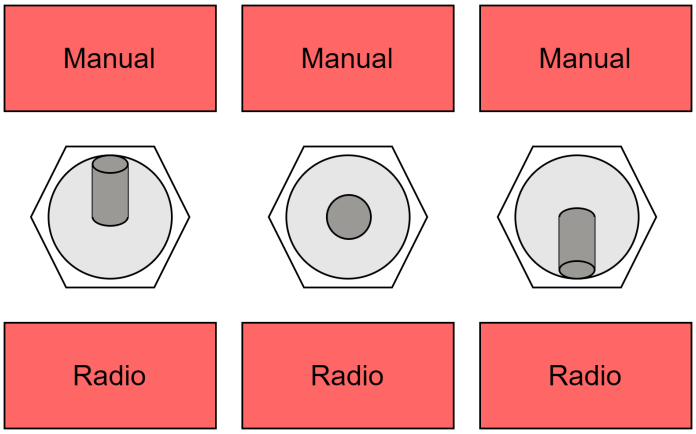
7.1.1 Tidligere funksjonalitet

Det eksisterende systemet er bygget opp slik at gasspådrag, brems, og sving mottar signaler fra en operatør stasjon via en UM600 radio eller fra en radiokontroller via en radiomottaker. Signalene behandles av to separate mikrokontrollere av typen Arduino Mega; en for brems- og gasspådrag og en for sving. Begge mikrokontrollerne er montert i et grått sikringsskap markert med en antenne på venstre side på ATV, se Figur 7.1.



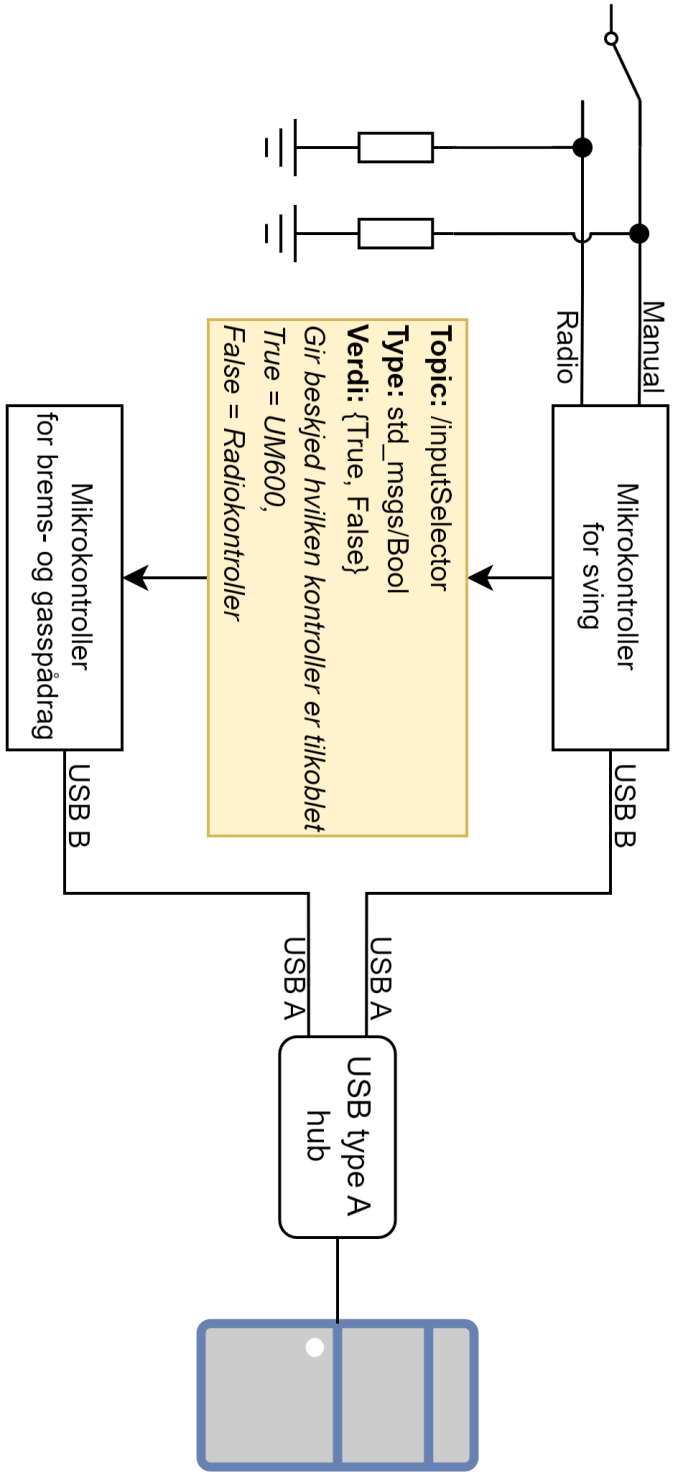
Figur 7.1: Plassering av ulike komponenter

Valg av mottaker styres av en tre-veis bryter tilkoblet mikrokontrollerne. Bryter kan posisjoneres mot markert posisjon Manual eller Radio, se Figur 7.2. Bryter kan også posisjoneres mellom Manual og Radio, men det er ingen ledning tilkoblet midt-posisjonen.



Figur 7.2: Tre-veis bryter for valg av mottaker

Når bryter peker mot Manual får mikrokontrolleren for styring spenning på pinne nr. 9 og når bryter peker mot Radio får den spenning på pinne nr. 10. Når bryter peker mot Radio, og pinne nr. 10 mottar spenning, går mikrokontroller for styring i en tilstand hvor den publiserer til en ROS topic med navn inputSelector, se Figur 7.3. Mikrokontroller for brems- og gasspådrag lytter på topic-en inputSelector og avgjør hvilke mottaker den skal lese data fra. Mikrokontroller for styring vil også i denne tilstanden abonnere på en ROS topic med navn um600Steering. Når en heltallsverdi av type Int64 publiseres til um600Steering topic-en, vil den sende et signal til stepper motoren som svinger rattet til ønsket posisjon. Tallverdien er definert i området 0 til 90, og beskriver styrevinkel.



Figur 7.3: Kobling mellom mikrokontroller og PU

7.1.2 Introduksjon til ROS

Robot Operating System, også kjent som ROS, er et rammeverk med formålet om å forenkle utviklingen av robotikk systemer. ROS støtter kommunikasjon mellom maskinvare-

og programvareenheter på et system ved hjelp av bibliotekene rclc [19], rclcpp [20], rosserial [21] og micro-ROS [22].

Et system basert på ROS er bygget opp av pakker, noder, topics, actions og services. Når en programvareutvikler utvikler et system hvor kommunikasjon foregår mellom forskjellige mikrokontrollere via en datamaskin, er ROS et nyttig verktøy. Som utvikler er det nødvendig å utforme en programvare arkitektur, beskrevet i kapittel 7.3.3. For å utvikle denne programvare arkitekturen oppstod et behov for å forstå hvordan kommunikasjon foregår i ROS.

En pakke kan bestå av en eller flere noder. Pakker hjelper brukerne av systemet til å sortere nodene for ulike formål; for eksempel kan en pakke med navn lager bestå av flere noder med navn finn_ledig_plass og algoritme_for_stifinning. Ved å kjøre de to nodene via ROS vil den ene noden holde oversikt over ledig plasser på et lager. Den vil så videreformidle informasjonen over en topic med navn ledige_lagerområder. De to nodene kan bygges, installeres og kjøres på terminal på en datamaskin slik:

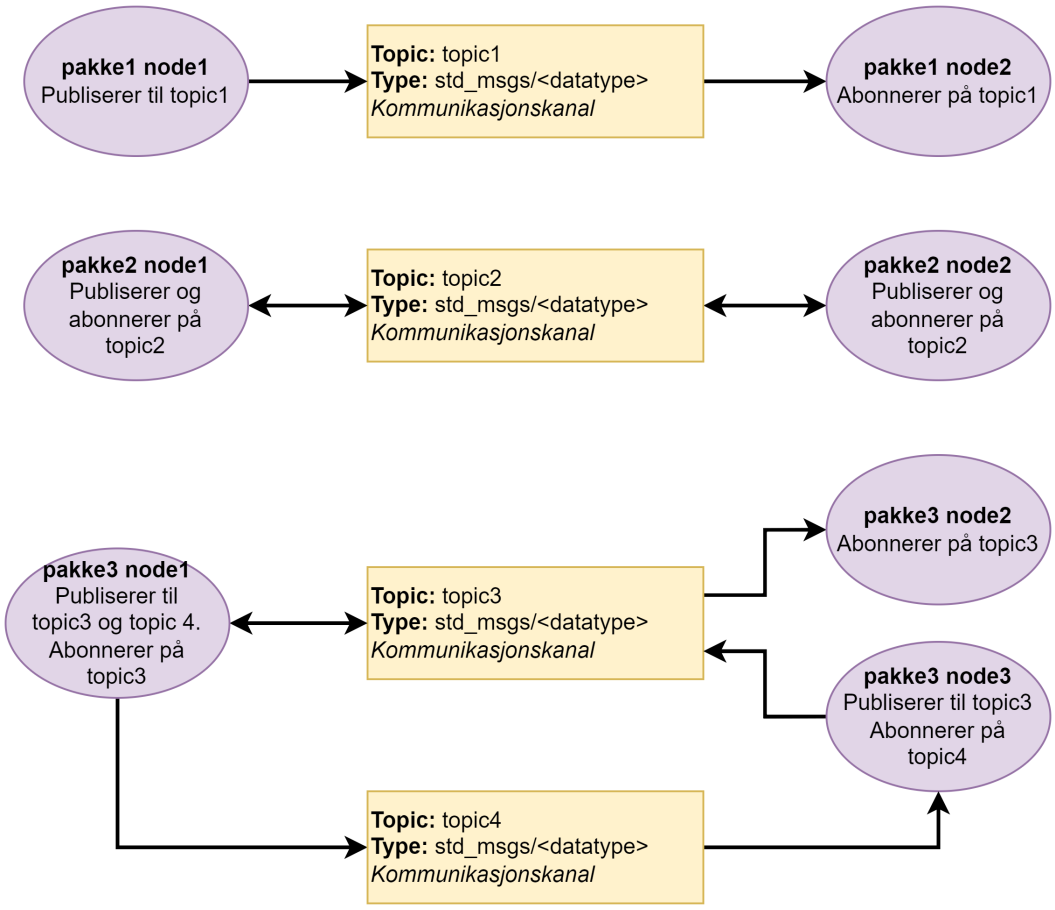
```
cd ros2_workspace
source /opt/ros/foxy/setup.bash
colcon build --packages-select lager
source install/setup.bash

# Terminal 1
ros2 run lager finn_ledig_plass

# Terminal 2
ros2 run lager algoritme_for_stifinning
```

Et ROS system kan bestå av flere pakker, noder og topics. Topics er en kommunikasjonskanal mellom noder. Noder kan publisere og abonnere på Topics for sende og motta data. [23] En beskrivelse av kommunikasjonen for et forenklet system med noder og topics beskrives i Figur 7.4.

ROS benytter også services og actions for å behandle kommunikasjon: Services er laget for to eller flere noder, hvor en node har rollen som tjener og de andre nodene er klienter. Først når klienten forespør informasjon fra tjeneren, vil tjeneren sende tilbake informasjonen [23]. En service kan sammenlignes med et funksjonkall.



Figur 7.4: Kommunikasjon mellom noder på datamaskin

En action er bygget opp av topics og services. I stedet for å publisere data kontinuerlig til en topic eller å publisere data en gang som en service. Vil en action vente på forespørsel fra en klient før den begynner å publisere data over en topic [23].

7.1.3 Analyse av Arduino kildekode

Som nevnt i 7.1 og sett i sammenheng med 7.4.2, virker det svært sannsynlig at kildekoden til hhv. Throttle-Arduino og Steering-Arduino må revideres for å tillate å styres fra et tredje grensesnitt, enten dette er et TUI, GUI eller andre autonome systemer. Løsningsforslaget for å omgå låsing av gir **??** innebærer at systemet påfører brems og gass i gitte tilstander for å påføre rotasjon i girkassen. Det er ikke nødvendig at systemet hverken abonnerer eller publiserer Steering-Arduino om å påføre styrestammen kraft for å svinge hjulene. Det betyr at analysen av Steering-Arduino i denne omgang kun sees i sammenheng med hva som direkte påvirker Throttle-Arduino.

7.1.3.1 Analyse av Steering-Arduino og dens betydning for girsystemet

Det er kun én funksjonalitet i denne kildekoden som er av interesse for girsystemet og det er at det leses av tilstanden på en bryter nevnt i 7.1.1 der besrives det også at mikrokontrolleren publiserer sann eller usann verdi på topic: /inputSelector. Grunnet at det kun er denne publiseringen som er av interesse for girsystemets side, er det ingen behov for endringer i kildekoden til denne kontrolleren.

For å legge til muligheten for at Throttle-Arduino kan styres via et 3. grensesnitt kan det gjøres endringer funksjonaliteten til bryteren koblet til Steering-Arduino slik at alle tre brytertilstander 7.2 benyttes.

Et eksempel på en slik endringer kunne vært noe liknende, der X representerer en ny pinne definert i Steering-Arduino koden.

1. Pin 9 og X er begge sanne: UI/TUI/GUI har kontroll
2. Pin 9 er sann, men pin X er usanne: Ekstern stasjon via UM600 har kontroll
3. Pin 9 og X er begge usanne: Radiokontroller har kontroll
4. Datatypen som publiseres på /inputSelector kan da endres fra Bool til Int, der det publiseres 0, 1 eller 2 istedenfor sann eller usann verdi.

Figur 7.5 og 7.6 dokumenterer at Steering-Arduino lytter til brytertilstand og publiserer til ROS. Kodesnuttene nedenfor er vist slik de er kodet fra tidligere prosjekter uten endringer fra LoneWolf 2023 sin side.

```
37 ros::Publisher inputSelector("inputSelector", &inputSelector_);
```

Figur 7.5: Initalisering og deklarasjon av ROS-publisher på /inputSelector

```
53 //--------------------- State machine ------------------
54 /*Funksjon for state machine, state bestemmes av hvilken pin som er høy fra switch*/
55
56 void State() {
57   if (digitalRead(localPin) == true) {
58     state = local;
59     inputSelector_.data = false;
60   }
61   else if (digitalRead(radioPin) == true) {
62     state = radio;
63     inputSelector_.data = true;
64   }
65 }
```

Figur 7.6: Logikk som lytter til pinne og publiserer sann eller usann verdi

Konklusjon på analysen er at girsystemet må ta hensyn til Steering-Arduino sin publisering på topic /inputSelector.

7.1.3.2 Analyse av Throttle-Arduino og dens betydning for girsystemet

Del 1 av girsystemet omhandler ikke gassrespons for å påføre rotasjon på girkassen til ATV. Det er derimot påkrevd ihht. sw krav 5 7.2.5 å sørge for at bremsen er aktivert under girskift. Del 2 av girsystemet må påføre rotasjon på girkassen for å lette prosessen med å flytte girspaken i ønsket gir dersom en låsesituasjon inntreffer. Måten prosjektet har valgt å påføre rotasjon på girkassen er å gi litt gass når giret befinner seg i en posisjon der det vil gi effekt, altså R, H eller L. Les mer om denne prosessen 7.4.9.5.

Logisk tankegang tilsier at det ikke samtidig bremses og gasses på et kjøretøy. Denne logiske slutningen har tidligere LoneWolf prosjekter tatt hensyn til ved utvikling av den opprinnelige kildekoden til mikrokontrolleren 7.8. Instruksjoner sendt til kontrolleren fra UM600 nettverket publiserer både brems og gass på samme topic /um600Throttle. Throttle-Arduino leser verdi sendt fra radiokontroller på pin D5 der mottakeren er tilkoblet dersom meldingen mottatt på topic /inputSelector er usann. I kildekoden til Throttle-Arduino tas det hensyn til hvilken verdi Steering-Arduino publiserer på /inputSelector. Er verdien usann kan man kun styre gassrespons på ATV med radiokontrolleren og dersom verdien er sann vil den kun ta i mot instrukser sendt via UM600 nettverket.

Figur 7.7 viser at den eksisterende kildekoden til Throttle-Arduino abbonnerer på topic /inputSelector (fra Steering Arduino) og /um600Throttle (fra UM600 nettverket). Figur 7.8 viser innenfor hvilket område gassrespons verdi ønskes mottatt og til hvilken verdi det

mappes til i callback funksjonen 7.9. Kodesnuttene nedenfor er vist slik de er kodet fra tidligere prosjekter uten endringer fra LoneWolf 2023 sin side.

```
66 ros::Subscriber<std_msgs::Int64> um600ThrottleSub("um600Throttle", um600ThrottleCb);
67 ros::Subscriber<std_msgs::Bool> emergencyStopSub("um600EmergencyStop", emergencyStopCb);
68 ros::Subscriber<std_msgs::Bool> inputSelectorSub("inputSelector", inputSelectorCb);
69 ros::Publisher emergencyStopPub("emergencyStop", &emergencyStop);
70 ros::Publisher rcThrottlePub("rcThrottle", &rcThrottle);
```

Figur 7.7: Initalisering og deklarasjon av ROS-subscribers og -publishers

```
12 const int minInputThrottle = -100;
13 const int maxInputThrottle = 100;
14 const int minPWM = 1200;
15 const int maxPWM = 1800;
```

Figur 7.8: Initalisering og deklarasjon av min og max verdi for gassrespons og PWM verdi

```
51 void um600ThrottleCb(const std_msgs::Int64& pwmThrottle_) {
52   if (inputSelector) {
53     int pwmThrottle1 = pwmThrottle_.data;
54     pwmThrottle = map(pwmThrottle1, minInputThrottle, maxInputThrottle, minPWM, maxPWM);
55   }
56 }
```

Figur 7.9: Callback funksjon som tar imot verdi fra -100 til 100 mapper dette dersom verdien mottatt på /inputSelector er sann

Det er åpenbart at kildekoden til Throttle-Arduino må endres dersom et tredje alternativ for å kunne ta i mot instrukser skal implementeres som en del av girsystemet. I tillegg må det analyseres hvordan gassrespons-verdi kan sendes fra girsystemet og hvordan det kan mottas og behandles på mikrokontrolleren på en måte som gir ønsket respons og driftsikkerhet. Det er også svært viktig at tidligere funksjonalitet ikke forstyrres eller forringes på noe vis. Endring av kildekoden må på ingen måte gjøre det vanskelig eller umulig å styre ATV slik det var før LoneWolf 2023 startet sitt arbeid.

Det er i hovedsak to grunner til at det ikke er ønskelig at girsystemet publiserer instruks på /um600Throttle topic som er i bruk fra før. Det er at konflikt kan oppstå dersom ekstern stasjon via UM600 nettverket og girsystemet er aktivt samtidig, samt at det virker uryddig og uoversiktlig om girssystemet publiserer data til en topic som har missvisende navn.

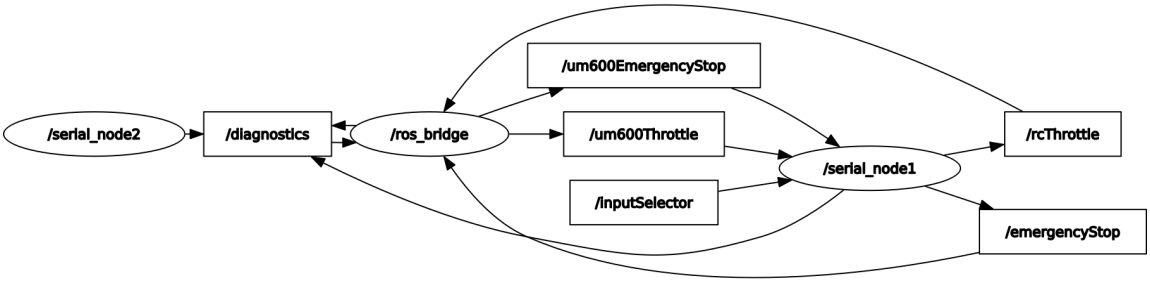
Etter denne analysen er det oppdaget minst fire ting som må endres for å kunne benytte eksisterende Throttle-Arduino sammen med girsystemet:

1. Det må opprettes en topic kontrolleren kan subscribe på i tillegg til /inputSelector
2. Det må opprettes en topic kontrolleren kan subscribe på for å motta gassrespons-verdi fra girsystemet
3. Det må opprettes en topic kontrolleren kan publisere til for å informere girsystemet om nåværende gassrespons-verdi
4. Kildekodens logikk må generelt revideres for å tillate girsystemet kan kontrollere gassrepons-verdi hvis og bare hvis det er ønskelig fra brukerens side

Og det er oppdaget minst 3 ting som i tillegg er ønskelig å endre:

1. Det bør opprettes en topic kontrolleren kan subscribe på for å motta nødstopp signal fra girsystemet
2. Det bør opprettes en topic for å publisere verdien til throttle1 variabelen (feilsøkingshensyn)
3. Kildekoden bør generelt ryddes opp i og splittes i flere filer

Figur 7.10 som her er representert ved serial node 1 viser relevante topics det må tas hensyn til ved revisjon av kildekoden. Figuren dokumenterer at serial node 1 blant annet abonnerer på /inputSelector og /um600Throttle. Publisering på topic /rcThrottle er kun relatert til bevegelse på stikken på radiokontrolleren og er ikke relevant for girsystemet i denne omgang. Publisering på /emergencyStop er interessant å ta med i systemet, da det videresender signal mottatt på pin D4 fra nødstopp på ATV.



Figur 7.10: Minimal visning av relevante topics for Throttle-Arduino

Konklusjon på analysen er at girsystemet bør ta hensyn til publisering på topic /emergencyStop. Det bør legges til en subscriber på ny topic som eksempelvis kan navngis

som /setThrottle der det sendes ønsket verdi mellom -100 til 100 fra girsystemet. Det må legges til en subscriber på en ny topic som eksempelvis kan navngis som /uiSelector der dataypen er sann eller usann. Sann verdi publisert fra girsystemet på topic /uiThrottle og sann verdi publisert på topic /inputSelector kan eksempelvis medføre at kun girsystemet kan endre gassrepons-verdi på Throttle-Arduino koblet til ATV.

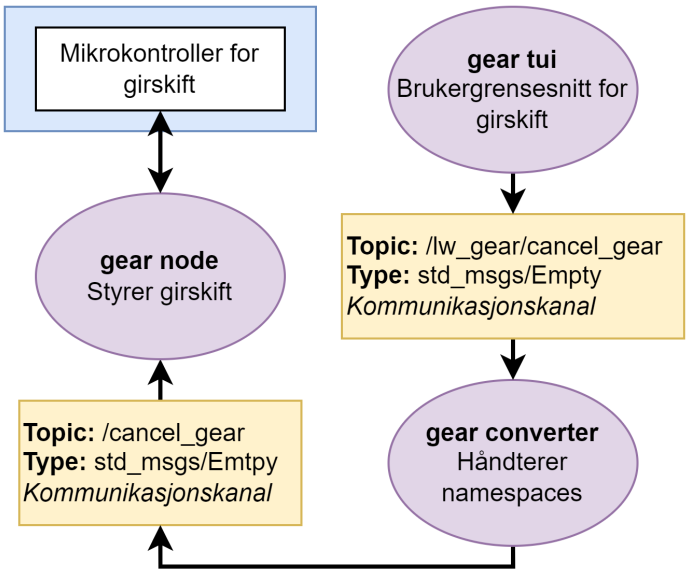
7.1.4 Begrensninger

Under planleggingsfasen og gjennom utviklingsfasen ble flere kompatabilitetsproblemer, mangler og svakheter avdekket. De mest merkverdige kommer frem i underkapitlene.

7.1.4.1 Rosserial og ROS 1

Som nevnt i kapittel 7.1.1 utviklet tidligere studenter systemer for sving, gass og brems på en Arduino Mega. Arduino Mega har støtte for ROS 1 funksjonalitet gjennom rosserial biblioteket, men dette har sine begrensninger. ROS 1 og ROS 2 er ikke direkte kompatible med hverandre, men ved hjelp av ROS2 pakken ros1_bridge er det mulig å opprette to-veis kommunikasjon mellom ROS 1 og ROS 2 [24].

Services og actions ble diskutert som en mulighet for å redusere ressursforbruk ettersom en service kun sender data på forespørsel, men det ble senere avdekket at rosserial ikke har støtte for hverken services eller actions. Prosjektet ble begrenset til noder og topics. Rosserial har heller ikke støtte for å abonnere på topics med namespaces. Mangel på støtte for namespaces er ikke funksjonelt problematisk, men med namespaces vil det være lettere for fremtidig utvikling ved å skape en separasjon mellom de forskjellige systemene. Det var ønskelig å ha et namespace for gir-systemet (lw_gear), ett for gass- og bremsesystemet og et for styresystemet. En mulig løsning for namespaces er en node som sammenkobler topics med og uten namespaces, denne noden kan derimot forårsake forsinkelser for tidskritisk funksjonalitet. Eksempelet for topic /cancel_gear på Figur 7.11 har som funksjon å avbryte aktivt girskift, og det er viktig å nå frem til mikrokontroller for girskift uten forsinkelser. Dersom en feil forekommer og bruker publiserer til topic /cancel_gear kan girskift prosessen stanses og redusere et eventuelt skadeomfang.



Figur 7.11: Namespace håndteringsnode

7.1.4.2 Analyse av muligheten for å benytte ROS-tjenester (services) på Rosserial Arduino-enheter

Etter undersøkelser vises det til følgende grunnlag for at det satses videre på topics og ikke services ved bruk at Arduino mikrokontrollere som ikke støtter ROS 2 (micro-ros), men som må benytte Rosserial biblioteteket for kommunikasjon med ROS 1, nærmere bestemt Noetic i dette tilfelle.

1. Begrensede ressurser: Arduino-enheter har begrensede ressurser som begrenser funksjonaliteten de kan støtte, inkludert støtte for komplekse kommunikasjonsprotokoller som ROS-tjenester.

2. Protokollkompatibilitet: ROS-tjenester er primært utviklet for ROS-rammeverket og er basert på spesifikke protokoller og meldingsformater. Å tilpasse disse protokollene for å fungere med Arduino-baserte systemer kan være komplisert og kreve tilpasning og håndtering av begrensninger.

3. Biblioteksstøtte: Det kan være begrenset biblioteksstøtte og dokumentasjon for å implementere ROS-tjenester på Arduino-plattformen. ROS Serial-biblioteket, som brukes til å kommunisere med Arduino via ROS, har visse begrensninger og kan kreve spesifikk konfigurasjon og tilpasning.

4. Begrensninger i ROS-økosystemet: ROS-økosystemet er i hovedsak rettet mot større maskinvareplattformer og operativsystemer, og støtte for Arduino og lignende enheter kan være begrenset. Dette kan føre til manglende dokumentasjon, fellesskapstøtte og verktøy for å forenkle oppsettet av ROS-tjenester på Arduino.

På grunn av disse utfordringene kan implementeringen av ROS-tjenester på Arduino kreve betydelig tilpasning, feilsøking og eksperimentering.

Konklusjonen av analysen er at girsystemet vil fungere tilfredstillende ved bruk av topics. Tjenester kunne muligens løst det på en bedre måte, men tatt i betrakning tidsbruk for konfigurasjon, er det ikke hensiktsmessig å gå videre med det.

7.1.4.3 Operativsystem og programvare

En stasjonær datamaskin er montert på høyre side av ATV, navngitt PU[8], se Figur 7.1. På denne datamaskinen er operativsystemet Ubuntu 20.04 LTS installert. Det er også installert programvare i form av Robot Operating System (ROS), ROS 1 Noetic Ninjemys og ROS 2 Foxy Fitzroy.

ROS distribusjonene ROS 1 Noetic Ninjemys og ROS 2 Foxy Fitzroy er ikke støttet på Ubuntu 22.04 LTS og ROS 2 Humble Hawksbill er ikke støttet på Ubuntu 20.04 LTS. ROS 2 Foxy Fitzroy har end-of-life (EOL) mai 2023 og ROS 2 Humble Hawksbill har EOL mai 2027 [25], [26]. Et mål var å oppgradere til Ubuntu 22.04 LTS og fortsette utviklingen i ROS 2 Humble, men tidligere avhengigheter på ROS 1 Noetic Ninjemys førte til kompatabilitetsproblemer. For å forlenge levetiden og fjerne avhengigheter til ROS 1 måtte det undersøkes hvorvidt micro-ROS på Arduino Portenta H7 kunne erstatte rosserial på Arduino Mega.

7.2 Krav for systemet

Fra tidligere prosjekter har ordet 'bruker' blitt brukt om personer som samhandler med ATV, men ettersom KDA har et mål for fremtidig utvikling av autonom programvare for Lone Wolf, var det nødvendig å redefinere ordet bruker. Bruker defineres for denne oppgaven som mennesker og/eller autonom programvare som samhandler med ATV og dets funksjonaliteter.

[8]Processing Unit

Som nevnt i kapittel 4.1.5 konkluderte gruppen med at et motorstyrt girskift-system var mest kosteffektivt og ville spare Lone Wolf ATV for kompliserte systemer. Dataingeniør-studentene gjenkjente muligheten for å utvikle et system som tar i mot instruksjoner fra bruker, kontrollerer tilstand for girskift og forsøker å utføre girskift. Dersom gir henger fast under girskift, vil en gyngealgoritme forsøke å bevege ATV i en posisjon som tillater girskift. Dette systemet blir nærmere forklart i kapittel 7.4.9.5.

7.2.1 Software Krav 1

Prosjekt:	1651		
Krav ID:	SW1	Dato:	19.05.2023
Kravklasse:	A	Status, krav:	V
Test ID:	T_SW1	Status, test:	G
Kravspesifikasjon	PU skal kunne styre mikrokontroller for girskift gjennom et grensesnitt.		
Beskrivelse av krav	PU skal ha toveis kommunikasjon med mikrokontrolleren. Mikrokontrolleren skal sende signaler til motordriver samt lese respons fra aktuatoren.		

Figur 7.12: Software Krav 1

Dataingeniør-studentene gjenkjente behovet for å kontrollere mikrokontroller for girskift gjennom et grensesnitt.

7.2.2 Software Krav 2

Prosjekt:	1651		
Krav ID:	SW2	Dato:	18.04.2023
Kravklasse:	A	Status, krav:	V
Test ID:	T_SW2	Status, test:	G
Kravspesifikasjon	Bruker skal ha toveiskommunikasjon med PU.		
Beskrivelse av krav	Bruker skal sende signal om girskift til PU gjennom et brukergrensesnitt (f.eks. terminal-vindu, trådløs kontroller, autonom programvare, GUI, e.l.). PU skal sende informasjon tilbake til bruker om tilstanden (akselerasjon, brems, girposisjon, GPS) til ATV.		

Figur 7.13: Software Krav 2

For videreutvikling av et system for girskift, gjenkjente dataingeniør-studentene et behov for et brukergrensesnitt slik at bruker kan kontrollere girskift fra PU[9].

7.2.3 Software Krav 3

Prosjekt:	1651		
Krav ID:	SW3	Dato:	18.04.2023
Kravklasse:	A	Status, krav:	V
Test ID:	N/A	Status, test:	N/A

Kravspesifikasjon	PU skal benytte en ROS2 node for å kommunisere med mikrokontroller.
Beskrivelse av krav	PU skal benytte ROS2 (Robot Operating System) node til å kommunisere ved å sende og motta data gjennom emner (publish-subscribe-topics) og tjenester (request-response-services).

Figur 7.14: Software Krav 3

For å unngå å skape flere avhengigheter til ROS 1, gjenkjente dataingeniør-studentene er behov for å utvikle i ROS 2.

7.2.4 Software Krav 4

INformasjon

Prosjekt:	1651		
Krav ID:	SW4	Dato:	18.04.2023
Kravklasse:	A	Status, krav:	V
Test ID:	T_SW4	Status, test:	G

Kravspesifikasjon	PU skal være kjent med absolutt girposisjon.
Beskrivelse av krav	PU skal være kjent med posisjonen til girspaken for hvilket gir den er i.

Figur 7.15: Software Krav 4

Dataingeniør-studentene gjenkjente et behov for at bruker skal være kjent med hvilket gir ATV befinner seg i.

[9]Processing Unit

7.2.5 Software Krav 5

Prosjekt:	1661		
Krav ID:	SW5	Dato:	19.05.2023
Kravklasse:	A	Status, krav:	V
Test ID:		Status, test:	
	T_SW 5.1	T_SW 5.1	G
	T_SW 5.2	T_SW 5.2	G
	T_SW5.3	T_SW5.3	G

Kravspesifikasjon	Det skal kun sendes instruksjon til mikrokontroller om girskift når ATV er stillestående, bremser er aktivert og det ikke gis gasspådrag.
Beskrivelse av krav	Girsystemet til ATV er laget slik at girskifte kun kan forekomme uten gasspådrag. ATV er utstyrt med en CVT girkasse. Denne fungerer slik at clutch begynner å gripe ved et bestemt turtall. Etter clutch har begynt å gripe burde ikke girskifte gjennomføres, ettersom dette kan medføre skade på interne komponenter i girkasse. Det er derfor viktig at det ikke er noe gasspådrag på ATV ved girskifte. Dersom bremser ikke er aktivert under girskifte, er det en fare for at ATV kan begynne å rulle. Det skal derfor ikke være mulig for ATV å gire før hjulene er stanset og bremsene er aktivert. Kontrollsløyfen skal lese data for å sikre at girskift ikke tar sted med mindre tilstanden tilsier at dette er tillatt.

Figur 7.16: Software Krav 5

Dataingeniør-studentene tilegnet seg kunnskap som tilsier at girskift under bevegelse kan påføre skade på girsystemet. Et behov for å sørge for at det kun sendes instruksjon til mikrokontroller for girskift når ATV er stillestående, bremser er aktivert og gasspådrag er inaktivt ble gjenkjent.

7.2.6 Software Krav 6

Prosjekt:	1868		
Krav ID:	SW6	Dato:	18.04.2023
Kravklasse:	B	Status, krav:	V
Test ID:	T_SW6	Status, test:	G

Kravspesifikasjon	Et script skal utvikles for installasjon av utviklermiljø på PU.
Beskrivelse av krav	For å utvikle programvare for girsystemet er det nødvendig å ha tilgang til et utviklermiljø som tillater dette. Dersom harddisk på PU blir ødelagt er det nyttig å ha et script som installerer og laster ned alt det nødvendige for å fort erstatte harddisk og fortsette utvikling.

Figur 7.17: Software Krav 6

Dersom Solid State Drive (SSD) på PU[10], eller PU påtar seg skade, når nye utviklere tar over prosjektet og har behov for flere installasjonsinstanser av ROS 1 Noetic Ninjemys og/eller ROS 2 Foxy Fitzroy, gjenkjente dataingeniør-studentene et behov for å utvikle et installasjonsskript som oppretter et utviklermiljø.

7.2.7 Software Krav 7

Prosjekt:	1680		
Krav ID:	SW7	Dato:	11.05.2023
Kravklasse:	B	Status, krav:	V
Test ID:	T_SW7	Status, test:	G

Kravspesifikasjon	Bruker skal kunne lese status for girskift på en UI.
Beskrivelse av krav	For å feilsøke, er det nyttig at bruker kan hente informasjon ved hjelp av en (Graphical / Text-based) User Interface. Her kan man få informasjon for hvilket gir ATV er i, lese klarsignal til giring, feil-/ tilbakemeldinger, el.

Figur 7.18: Software Krav 7

For å feilsøke girsystemet gjenkjente dataingeniør-studentene et behov for å lese informasjon på et brukergrensesnitt i form av posisjon på giret, klarsignal for giring, tilbakemeldinger fra mikrokontroller for girskift.

7.2.8 Software Krav 8

Prosjekt:	1680		
Krav ID:	SW8	Dato:	18.04.2023
Kravklasse:	B	Status, krav:	V
Test ID:	N/A	Status, test:	N/A

Kravspesifikasjon	PU må ha fungerende operativ system for ROS2 & Gazebo installasjon.
Beskrivelse av krav	Fra tidligere prosjekt har PU installert Ubuntu 20.04 med Gazebo og ROS2 Foxy. ROS2 Foxy nærmer end-of-life (EOL) og bør oppgraderes til ROS2 Humble (EOL 2027). Testing og verifisering av kode og kontrollsløyfe for girmekanisme kan gjøres via ROS2 og Gazebo.

Figur 7.19: Software Krav 8

[10]Processing Unit

Ved oppstart av prosjektet realiserte dataingeniør-studentene at ROS 2 Foxy Fitzroy mister støtte, mai 2023. Et behov for å forlenge levetiden på ROS 2 ved oppgradering av operativsystem ble gjenkjent som en mulighet.

7.3 Planlegging for utvikling

Etter forståelsen for systemet var etablert og kravene for videreutviklingen var definert, begynte dataingeniør-studentene planlegging for videreutvikling på systemet. Underkapitlene beskriver prosessen.

7.3.1 Teknologier og verktøy

Som beskrevet i kapittel 7.1.1 har tidligere prosjekter benyttet seg av teknologier i form av ROS 1, ROS 2, C++, Python og Bourne Again SHell (bash). For videreutviklingen av det automatiske girsystem på Lone Wolf ATV, valgte dataingeniør-studentene etter anbefaling fra arbeidsgiver å planlegge prosjektutviklingen basert på eksisterende teknologi.

Som nevnt i kapittel 1.4.7 har tidligere studenter benyttet git repositories på Azure DevOps som et verktøy for versjonskontroll. Arbeidsgiver gav i den sammenheng tilgang til Azure DevOps platformen for videreutvikling. Git tillater flere utviklere å arbeide på samme filer uten overskriving. Kommandoer som 'git clone', 'git add', 'git commit', 'git pull' og 'git push' dekker det grunnleggende behovet for samarbeid i et felles repository. Det kan oppstå konflikter dersom endringer på samme fil har tatt sted. Visual Studio Code er nyttig verktøy i form av dets funksjonalitet som en kode redigeringsprogram og som en 'Integrated Development Environment' (IDE), men det er også nyttig for konfliktløsning for versjonskontrollsystemer som git.

Etter tilgang på Azure DevOps gikk dataingeniør-studentene til verks med å etablere en oversikt over hvordan tidligere prosjekter har blitt kompilert og testet. For å få en bedre forståelse for hvordan systemet skulle struktureres ble blant annet programmet rqt_graph benyttet for å få en oversikt over ROS nodene på systemet. Syntaksen i programmet rqt_graph ble også en inspirasjon for utviklingen av diagrammet som beskriver grensesnittene i systemet, se figur 7.20.

I kapittel 1.4.18 beskrives VMware Workstation 17 Player som et verktøy for oppretting og

kjøring av ulike operativsystemer. PU[11] har som nevnt installert operativsystemet Ubuntu 20.04 LTS. For å starte med utviklingen forsøkte dataingeniør-studentene å bygge pakkene fra tidligere prosjekter, men etter mange forsøk som resulterte i feil under kompilering oppstod behovet for å bygge et separat ROS 2 workspace. Det var tidsbekostende å både forsøke å kompilere tidligere systemer, samt å installere ROS 2 manuelt. I den sammenheng basert på krav SW6, kom dataingeniør-studentene til enighet om å utvikle et bash installasjonsscript med hensikt for å forenkle installasjonsprosessen.

En nyttig egenskap å ha ved utvikling på et linux basert operativsystem er evnen til å manøvrere filsystemet via en terminal. Kompilering av ROS 2 gjøres via terminal ved hjelp av python3-colcon-common-extensions pakken. Ved å åpne en terminal i filplasseringen til et etablert ROS 2 workspace og kjøre terminal kommandoen 'colcon build', blir filene 'CMakeLists.txt' og 'package.xml' lest. CMake er et verktøy som spesifiserer hvordan et system bygges og installeres. I 'CMakeLists.txt' og 'package.xml' spesifiseres pakkenavnet, nodenavnet, avhengigheter i form av rclcpp og std_msgs (standard datatyper til topics), samt hvor kildekoden er plassert i filsystemet. Etter kompilering opprettes tre mapper med navn build/, install/ og log/. Installasjon av pakken og noden på systemet gjøres ved kommandoen 'source install/setup.bash'. Neste steg er å kjøre ROS 2 noden med kommandoen 'ros2 run pakkenavn nodenavn'.

7.3.2 Arduino mikrokontrollere

Arduino Mega 2560 er en rimelig og funksjonell mikrokontroller som kan benyttes til en lang rekke oppgaver. Det er to slike kontrollere allerede implementert på ATV som styrer henholdsvis gasspådrag/brems og rattutslag. Det er derfor naturlig at mikrokontroller fra samme produsent benyttes til girsystemet dersom det er mulig. Det er fleksibelt å bruke en uavhengig mikrokontroller for å styre funksjoner på en robot, i dette tilfellet en fjernstyrt ATV.

Arduino Zero eller Due ble vurdert for implementasjon av micro-ROS (ROS 2), men landet på Arduino Portenta H7 grunnet at dette var den eneste micro-ROS kompatible Arduino kontrolleren tilgjengelig innen rimelig tid fra Elfa Distrelec.

[11] Processing Unit

7.3.2.1 Arduino UNO

LoneWolf 2023 ønsket å levere to separate mikrokontrollere for håndtering av automatisk girskift. Det første systemet som skulle utvikles benyttet en Arduino Uno og rosserial. Dette systemet fungerer på samme måte som de andre mikrokontontrollerne allerede tilkoblet ved at det benytter ROS Bridge for å kommunisere med ROS 2 Foxy.

7.3.2.2 Arduino Portenta H7

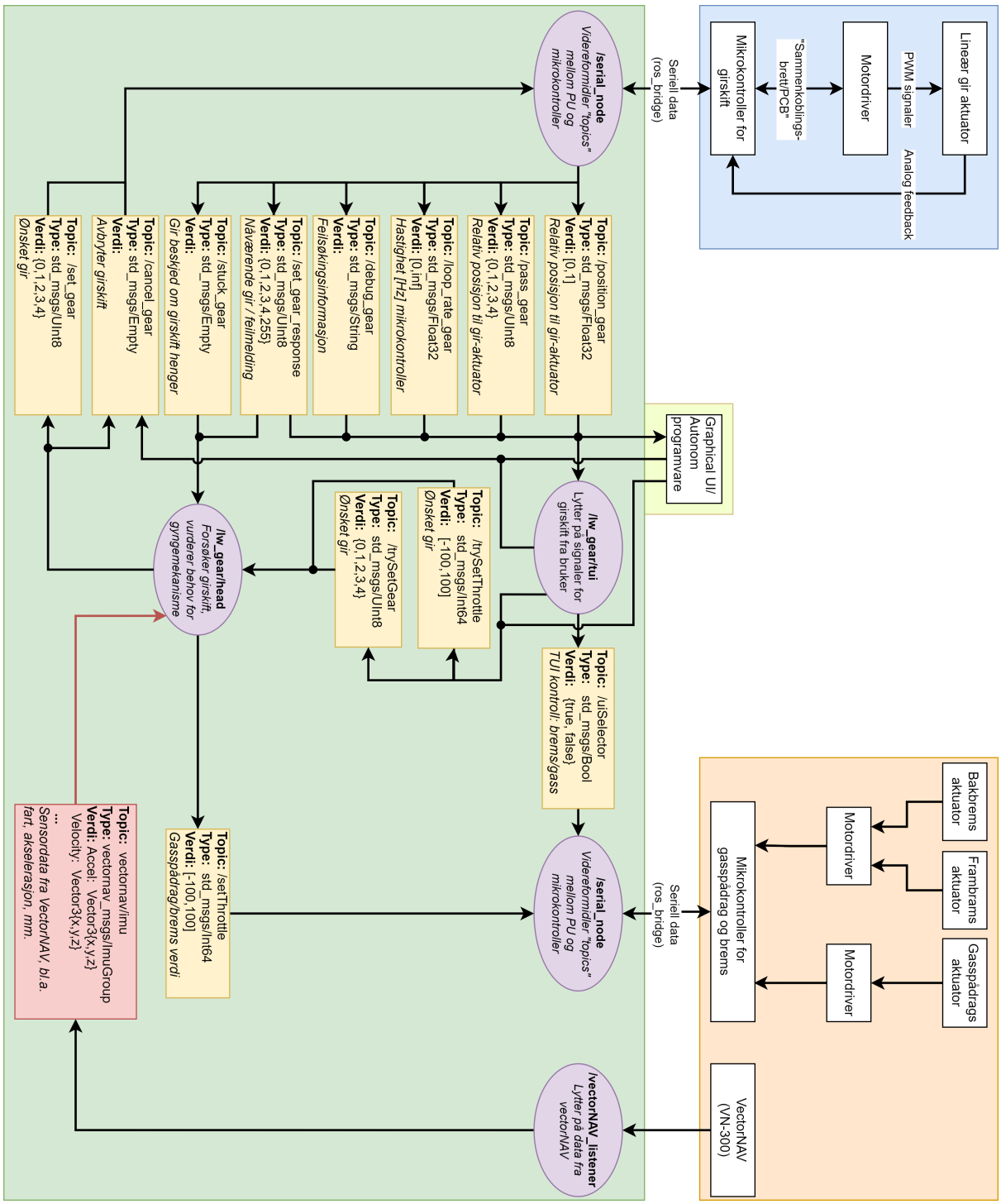
Den andre mikrokontrolleren består av en Arduino Portenta H7 og benytter micro-ROS slik at ROS Bridge ikke trenger å benyttes for direktekommunikasjon med ROS 2 Foxy. Merk at ROS Bridge likevel må benyttes for all kommunikasjon fra ROS 2 til de allerede eksisterende mikrokontrollerne for gasspådrag og sving. Portenta H7 har en STM32H747 ARM-prosessor, som inneholder to kjerner, M7 og M4, som kan flashes med hvert sitt separate program som kjører parallelt. M7 er hoved-kjernen som kjøres alltid ved oppstart. M4-kjernen krever at M7-programmet kaller en bootM4-funksjon før den starter. For å oppnå kommunikasjon mellom kjernene kan RPC[12] brukes, som gjør det mulig å utføre funksjonskall på tvers av kjernene. Den har mer RAM enn Arduino Uno og Mega, og har offisiell støtte for Micro-ROS. I likhet med Arduino Uno og Mega har den programmerbare I/O-pinner, som gjør det mulig å bruke den på et kretskort på omtrent samme måte som vi har gjort tidligere med Arduino Uno.

7.3.3 Programvare Arkitektur

Før videreutvikling av programvare på det eksisterende systemet, utformet dataingeniør-studentene en programvare arkitektur som beskriver funksjonaliteten til systemet. Figur 7.20 og Figur 7.21 har gjennomgått mange iterasjoner, hvor førstnevnte beskriver de planlagte grensesnittene i systemet, og sistnevnte beskriver tilstander ATV kan befinne seg i. Alle iterasjoner av diagrammene for grensesnitt og tilstander, mm. er vedlagt dokumentasjonen, se mappen Vedlegg/Bilder og Diagrammer

[12]Remote Procedure Calls

7.3.3.1 Noder og grensesnitt



Figur 7.20: Diagram for grensenitt

Den oransje boksen i Figur 7.20 beskriver fysiske komponenter som ble benyttet under utviklingen av ATV i tidligere prosjekter, det er planlagt for å bruke data fra komponentene i diagrammet til å utvikle styring av gir-aktuator.

VectorNAV (VN-300) er et navigasjonssystem med funksjonalitet for måling av bevegelse ved hjelp av blant annet akselerometer. Databladet beskriver to varianter av VN-300, Rugged og SMD (Surface Mount Device)[27]. I Appendiks A2, Kapittel 3.4 VectorNAV Casing, Figur 4; Kan VN-300 identifiseres av type Rugged. Datablad for VN-300 Rugged beskriver to protokoller for seriell kommunikasjon: RS-232 og Serial TTL. De eksisterende nodene som leser fra VN-300 er utviklet for ROS1, hvilken protokoll som er brukt mellom PU og VN-300 er et usikkerhetsmoment. Aktuator for frambrems, brems- og gasspådrag styres av Arduino, som kommuniserer med PU i form av seriell data ved hjelp av rosserial biblioteket.

Den lime grønne boksen i Figur 7.20 beskriver eksterne systemer for hvordan bruker kan samhandle med systemet i form av Graphical User Interface (GUI) og for fremtidig utvikling; Autonom Programvare (AP). For å forenkle feilsøking av girsystemet er det planlagt at Text-Based User Interface (TUI) navngitt /lw_gear/tui i diagrammet, GUI og AP leser data fra systemet (pil inn), og skriver data til systemet (pil ut).

Den blå boksen i Figur 7.20 beskriver de nye komponentene for implementasjon av det nye girskift-systemet. PU sender instruksjon til mikrokontroller som sender signal til motordriver som styrer gir-aktuator, mikrokontroller sender tilbake posisjonsdata fra gir-aktuator, samt hvilket gir den passerer, frekvensen på mikrokontroller, etc. til PU.

Den grønne boksen i Figur 7.20 beskriver ROS2 systemet som skal implementeres. Runde lilla bokser beskriver noder, gule bokser; topics og rød boks; ikke implementert topic.

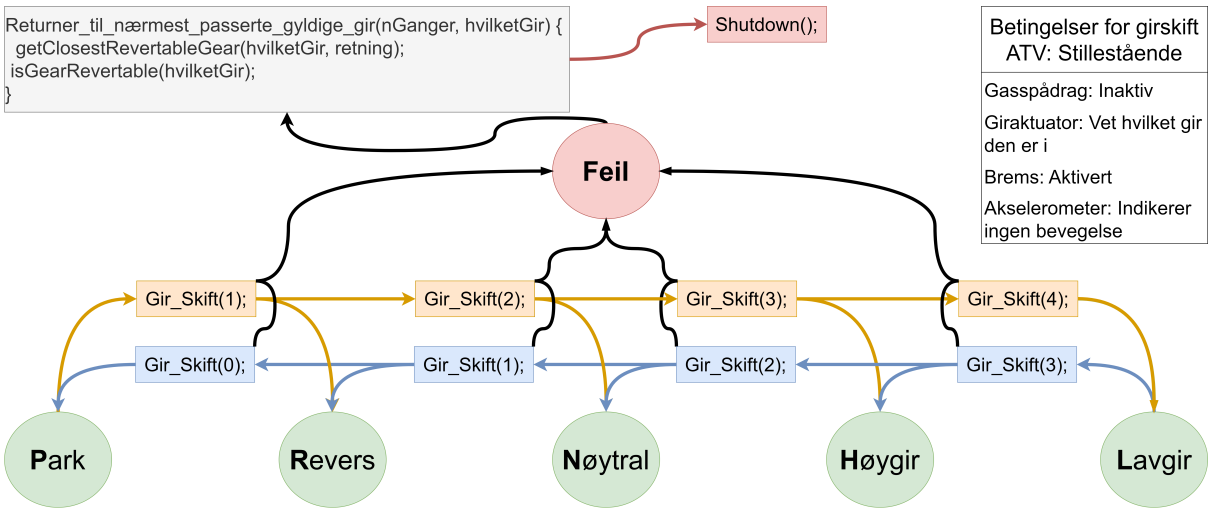
TUI og GUI leser data og presenterer data til bruker fra mikrokontroller for girskift gjennom topics: /position_gear, /pass_gear, /loop_rate_gear, /debug_gear og /set_gear_response. TUI og GUI mottar instruksjoner fra bruker og publiserer til topics: /cancel_gear som avbryter pågående girskift, samt /trySetThrottle og /trySetGear som videreformidler instruksjonene til Head-noden navngitt /lw_gear/head som er hjernen til systemet. Dersom hjernen får instruksjon om girskift fra bruker, kan den bremse ATV før den gjennomfører girskift. Det var også planlagt å lese og kalibrere data fra VectorNAV (VN-300) for en indikasjon på bevegelse i form av akselerasjon og/eller fart.

For en nærmere beskrivelse av de ulike nodene, les kapittel 7.4.9 for Head, kapittel 7.4.13

for GUI og kapittel 7.4.12 for TUI.

/serial_node for brems- og gasspådrag benytter biblioteket rosserial fra ROS 1 Noetic Ninjemys. I kapittel 7.4.6.1 nevnes det at arbeidsgiver har vist interesse for fornyelse av system, da det er ønsket å gå bort fra ROS Bridge. Det er derimot ingen krav om å unngå ROS Bridge. I kapittel 7.4.7 kommer det frem at ROS2 Foxy har EOL mai 2023, det er derfor en mulighet å oppgradere de eksisterende nodene til micro-ROS. Oppgradering fra ROS 2 Foxy til ROS 2 Humble vil blant annet kreve en oppgradering av all eksisterende kildekode på Arduino fra tidligere prosjekter.

Figur 7.21 viser seks tilstander for girskift-systemet; fem tilstander for de absolutte posisjonene for giret og en feiltilstand. Før girskift kan iverksettes må ATV være stillestående: Sensordata i form av akselerometer og GPS data fra Vector NAV, samt verdier fra mikrokontroller for brems- og gasspådrag er parametere som kan være med på å avgjøre hvorvidt ATV er stillestående. Når alle kriterier er møtt blir det utført et forsøk på å gire fra en tilstand til en annen. Et eksempel kan være når gir befinner seg i **L**avgir, og bruker ønsker å gire til **R**evers.



Figur 7.21: Diagram for tillatte tilstander for girskift

I forkant av utviklingen av de tekstbaserte og grafiske brukergrensesnittene ble en kodesnutt utformet og foreslått til arbeidsgiver. Kodesnutten under illustrer løsningen som ble vurdert i forkant av utviklingen av nodene Head, TUI og GUI.

```
1 void permit_gearchange(int set_gear, int current_gear) {
2
3   // Function checks if requirements/criteria are met.
```

```
4   bool requirements_met{ get_requirement_status(set_gear, current_gear)
     };
5
6   if (requirements_met && current_gear != set_gear) {
7     // change gear recursively.
8     recursive_gearchange(set_gear, current_gear);
9   }
10
11  else if(!requirements_met)
12    std::cout << "Requirements not met\n";
13
14  else if (current_gear == set_gear)
15  switch(current_gear){
16    case 0:
17      std::cout << "Gear is allready in Park\n";
18    break;
19    case 1:
20      std::cout << "Gear is allready in Reverse\n";
21    break;
22    case 2:
23      std::cout << "Gear is allready in Neutral\n";
24    break;
25    case 3:
26      std::cout << "Gear is allready in High-gear\n";
27    break;
28    case 4:
29      std::cout << "Gear is allready in Low-gear\n";
30    break;
31  }
32 }
```

```
Gir_Skift(3);
Gir_Skift(2);
Gir_Skift(1);
Gearchange is completed!

[1] + Done                    "/usr/bin/gdb" --interpreter=mi
martin@Legion-Y740-15IRHg:~/Git/cpp/Gir_Logikk$
```

Figur 7.22: Output fra kode

Som nevnt i kapittel 4.1 kan girskift prosessen henge seg opp, spesielt når motor er

avskrudd. Det er ikke planlagt å implementere fjernstyrt oppstart, men det kan være en fremtidig forbedring av ATV. Fra eksempelet der bruker skal gire fra **R**evers til **L**avgir, må Gir_Skift funksjonen kjøre med parameter 3, 2, 1, se figur 7.22. Dersom en feil skjer under forsøk på girskift må girposisjon tilbake i posisjon, før gyngemekanismen aktiveres som sørger for at girskift ikke forekommer når ATV er i bevegelse. For mer informasjon om konseptkode i form av Doxygen rapport, se Appendiks A4.

Logikken for girskift i koden beskriver sekvensielle girskift i retning opp eller ned. Etter progresjonsmøte med arbeidsgiver, onsdag 22. mars 2023, ble det avdekket ved hjelp av eksterne veiledere at utførelsen av sekvensielle girskift vil ta lengre tid enn utførsel av absolutte girskift, som medfører et behov for ny iterasjon av foreslått logikk. Det er verdt å nevne at absolutt girskift er implementert for mikrokontroller og beskrives i kapittel 7.4.4.

7.3.4 planlegging av mulige implementasjon for girsystem del 1

Girstystem del 1 må eller bør ta hensyn til følgende krav: Software krav 1 7.2.1, 2 7.2.2 og 3 7.2.3, som blant annet innebærer:

- Ta imot instruksjon om ønsket gir fra bruker via et grensesnitt
- Begrense kraften som påføres girstaget innenfor verdier anslått her.

7.3.5 planlegging av mulige implementasjon for girsystem del 2

Girstystem del 2 må eller bør ta hensyn til følgende krav: Software krav 4 7.2.4, 5 7.2.5 og 7 7.2.7, som blant annet innebærer:

- Å være kjent med absolutt girposisjon
- Ikke tillate girskift med mindre ATV har aktivert bremsene
- Informasjon om girstatus skal formidles til bruker

Sett i sammenheng med tidligere utredet løsning for girsystem del 2 7.3.6 må det utvikles en presis og driftsikker algoritme for å hjelpe aktuator med å bevege girspak til ønsket gir dersom girkassen er rotert slik at det oppstår en låsing under girskift. Algoritmen trenger ikke å ta hensyn til kraft påført girstag da dette er utredet i 5.2.9 og implementert

i kildekode til gir-arduino 6.4 og 7.4.4.5.

Hovedpoenget med en programvarebasert løsning for å omgå låsing av gir kan oppsummeres slik:

Etter undersøkelser og testing av endring på girspakens posisjon og girkassens begrensninger når motor er av eller på 4.1.3, viser det seg at girspak alltid lar seg flytte tilbake til utgangspunktet dersom bremsene er aktivert under hele girskifteprosessen. Dersom instruksjon om girskifte ikke lar seg utføre, må girspaken enten settes i Reverse, High eller Low, for deretter å påføre et lite gasspådrag på ATV for å rotere girkassen. Etter dette gasspådraget kan det igjen forsøkes å gire til opprinnelig ønsket gir. Grunnen til at det er behov for å rotere girkassen er fordi innretning på drev i girkassen kan kollidere med hverandre om de ikke er synkronisert for ønsket girskift 4.1.

Girsystem del 2 må eller bør ta hensyn til følgende som ikke er nevnt i girsystem del 1 7.3.4:

- Vite hvilket gir som er aktivt på ATV før instruksjon om girskift sendes
- Koordinere brems, gass og tilbakeføring av girspak til opprinnelig posisjon
- Antall ganger det forsøkes å gire frem og tilbake før systemet må gi opp
- Dersom systemet gir opp, skal det være mulig å dytte litt på ATV for deretter å prøve igjen uten å starte systemet på nytt
- Behandle returdata fra Gir-Arduino:
  - Absolutt posisjon til aktuator
  - Feilmelding relatert til at aktuator ikke klarer å flytte girspak til ønsket posisjon

7.3.6 Plan for utvikling av ROS 2 Head pakken

Basert på analyser av det tidligere systemet slik det er beskrevet gjennom kapittel 7.1 gjør det mulig å planlegge konsept for utvikling av en hoved pakke. Denne pakken ble først internt omtalt som en samlenode og hovedtanken er at PU har betraktelig større maskinvareressurser enn det som er tilfelle for Arduino enhetene tilkoblet systemet. Det ble likevel planlagt for at Gir-Arduino skal være mest mulig selvstendig, men del 2 av girstystemet, samt koordinasjon av alle delsystemer skal prosesseres av PU.

Det var et ønske fra oppdragsgiver at prosjektet i hovedsak leverer ROS 2 kompatible pakker der det er mulig. Som nevnt i den tidligere analysen vil første utgave av Gir-Arduino bli programmert med ROS 1 kompatibilitet (Rosserial). Senere versjon av Gir-Arduino ønskes levert med ROS 2 kompatibilitet (micro-ROS). Det er derfor viktig at det planlegges for at Head pakken kan kommunisere med både ROS 1 og ROS 2 versjonen av Gir-Arduino implementasjonen.

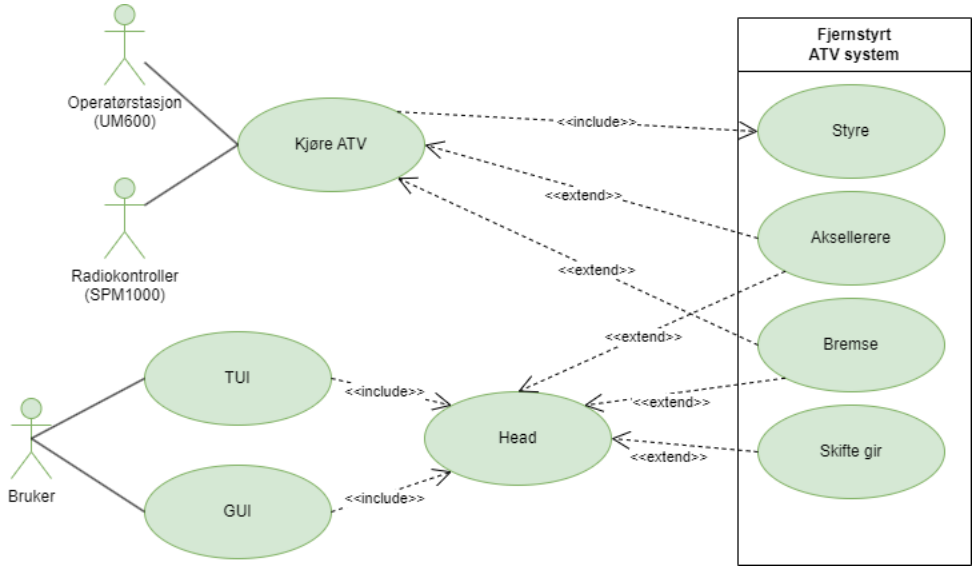
I forbindelse med planleggingen av Head pakken og dens hovedfunksjonalitet; 'samlenode' og 'anti-jam-system' ble det etterhvert klart at samlenodekonseptet var høyst reelt og fornuftig. Det ble også raskt bestemt at anti-jam systemet skal implementeres i Head-pakken.

To muligheter for implementasjon av anti-jam systemet ble vurdert. Les mer om første alternativ her 7.3.3.1 og her 7.21. Andre alternativ med if-else struktur ble raskt avfeid som en ressurskrevende og lite presis løsning. Begge alternativene ser ut til å bli erstattet av en tilstandsmaskin. Argumenter som taler for at det bør benyttes en tilstandsmaskin:

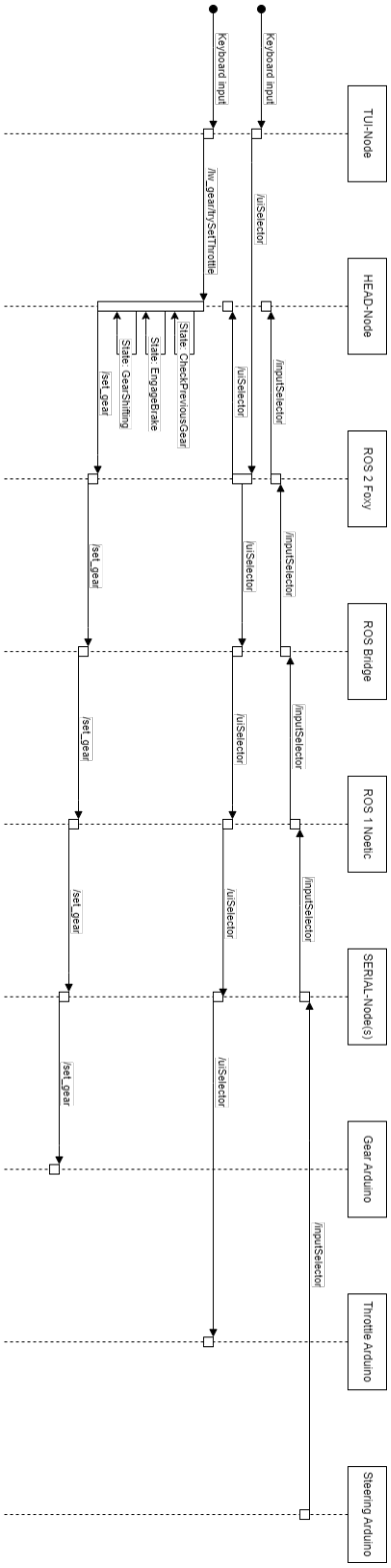
- En tilstandsmaskin er en modell som brukes til å beskrive og styre systemers atferd. Den representerer systemets ulike tilstander og overgangene mellom disse tilstandene og består av en samling tilstander, hendelser og overgangsregler
- Ved å benytte ulike kontrollstrukturer i callbackfunksjoner for ROS abonnering kan systemet reagere ulikt på samme type beskjede avhengig av hvilken tilstand systemet befinner seg i ved mottak av melding
- Antall tilstander kan enkelt modifiseres ved behov for utvidet funksjonalitet
- Tilstandsmaskiner brukes ofte i komplekse systemer som er avhengig av undersystemer og der det er naturlig at systemet befinner seg i ulike faser.

Som et tidlig utkast ble det utarbeidet et sekvensdiagram 7.24 som omfatter kommunikasjonsveien gjennom hele systemet. Diagrammet begrenser seg til kun til tenkt oppstart av systemet og første utkast av en tenkt tilstandsmaskin. Legg merke til hvor oppdelt kommunikasjonen blir ved å benytte ROS 1 og 2 samtidig forbundet med ROS Bridge og seriell-noder som igjen kommuniserer med mikrokontrollere. Kommunikasjonen er likevel meget rask og presis og er langt innenfor det som skal til for å lage et presist system. Det ble også utviklet et USE-CASE diagram i planleggingsfasen 7.23. Se mer

omfattende sekvensdiagram som ble utviklet i løpet av utviklingsprosessen (...).



Figur 7.23: USE-CASE diagram for tenkt oppsett for Head noden



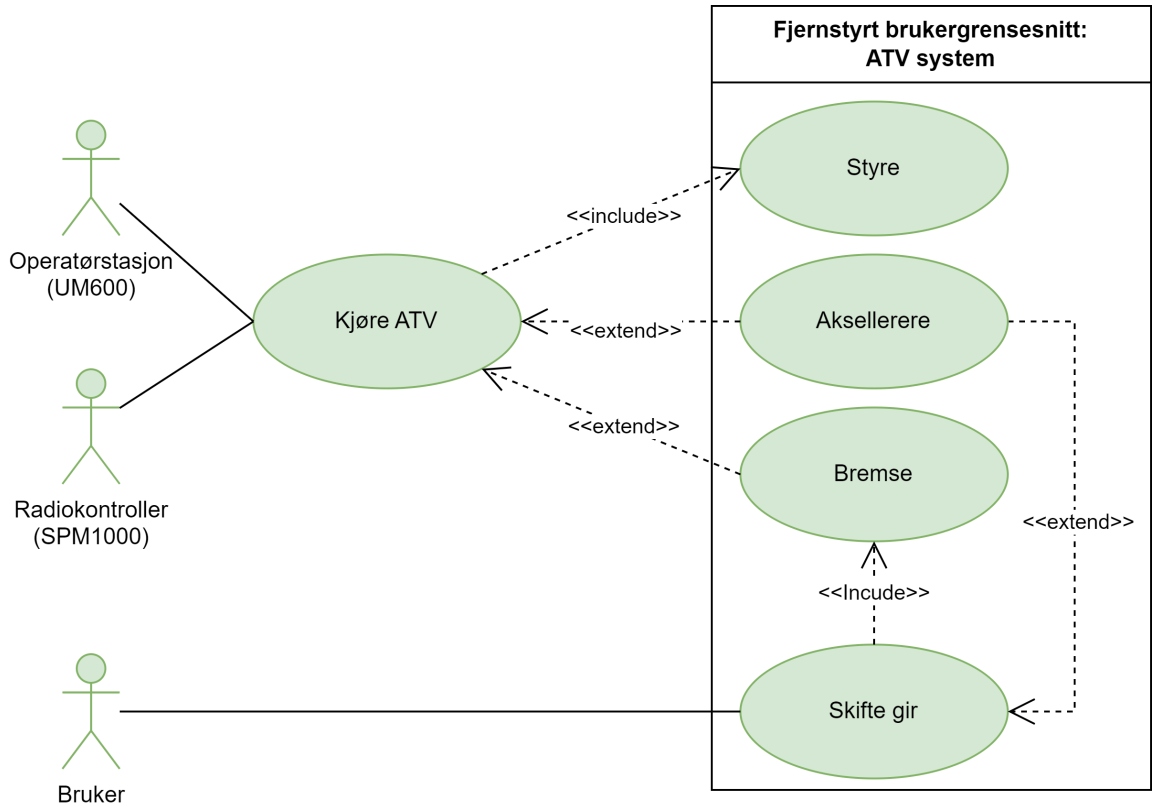
Figur 7.24: Sekvensdiagram av Head pakkens interaksjon med systemet tidlig i planleggingsstadiet

Les mer om utvikling av tilstandsmaskinen her 7.4.9.5 under kapittel Utvikling av ROS 2 Head pakken 7.4.9.

7.4 Utvikling og testing av system

Ved utvikling av nye systemer er det tidseffektivt å fortløpende kunne teste funksjonalitet og at kode fungerer etter hensikt. Ved implementasjon av systemer der utvikling og testing ikke er mulig å utføre fortløpende stilles det andre krav til koding og systemutvikling. I dette prosjektet har LoneWolf 2023 hatt tilgang til ATV én gang pr uke á to til tre timer. Det har dog vært mulig å rekvirere ekstra tid hvis ønskelig, men som hovedregel har nevnte ukentlig frekvens vært benyttet. Det betyr at systemutviklerne i prosjektet har måttet simulere funksjonalitet i større grad enn det man har vært vant med fra tidligere skolefag og private prosjekter. Denne utfordringen er svært lærerik og alle gruppens medlemmer kommer til å møte på denne utfordringen i næringslivet på et eller annet tidspunkt.

Som et ledd i planleggingen og utviklingen av girsystemet ble det utarbeidet et USE-CASE diagram for å vise på en oversiktlig måte hvordan brukeren kan benytte både det eksisterende systemet og det nyutviklede girssystemet. Som vist i figur 7.25 representerer girsystemet den nedereste brukeren som interagerer med skifte av gir. Funksjonaliteten å skifte gir er avhengig av alltid å kunne iverksette brems og i noen tilfeller vil girsystemet også være avhengig av å kunne påføre akselerasjon.



Figur 7.25: USE-CASE diagram for bruk av både eksisterende system og nytt girsystem

7.4.1 Processing Unit og Teststasjon

Lone Wolf 2023 bygger videre på tidligere Lone Wolf prosjekter som hver for seg har lagt til nye funksjoner på kjøretøyet. For hvert nye prosjekt blir det stadig mer materiale tilgjengelig for å få full forståelse for alle delsystemer. Datastudentene har et spesielt behov for å analysere det tidligere systemet for å kunne lansere målrettede løsninger. Det var ønskelig å ha tilgang til ATV og dens Processing Unit langt utover hva som var praktisk mulig for KDA å tilrettelegge for, spesielt siden USN ikke hadde mulighet til å ha ATV utplassert på skolens områder, selv ikke i kortere perioder. Etter godkjenning fra arbeidsgiver ble det bestemt å sette opp en klonet utgave av PU på grupperommet.

Først ble det satt opp ulike Virtual Machines (VMs) 1.4.18 der noen ble satt opp med nøyaktig samme oppsett som PU og andre med OS og ROS 2 versjon det var ønskelig å oppgradere til. Ulempen med VM er at maskinvaren delvis er svak, spesielt på grafikk i forbindelse med Gazebo.

Forskjellige oppsett med VM

1. Ubuntu 22.04 LTS og ROS 2 Humble Hawksbill og Gazebo 11
2. Ubuntu 22.04 LTS og ROS 2 Humble Hawksbill, ROS Bridge og Gazebo 11
3. Ubuntu 22.04 LTS og ROS 2 Foxy
4. Ubuntu 22.04 LTS og ROS 1 Noetic
5. Ubuntu 20.04 LTS og ROS 2 FOXY
6. Ubuntu 20.04 LTS og ROS 1 Noetic og Rosserial

Det var viktig for maskin og elektro studentene å gjøre nødvendige målinger og undersøkelser på ATV i den tidlige delen av prosjektet. Grunnet sikkerhetshensyn kunne ikke data studentene analysere PU eller utføre testing på denne når det samtidig ble gjort andre fysiske undersøkeler på ATV. Strømkretser måtte være brutt når arbeid ble utført i nærheten av elektronisk bevegelige deler. Det ble montert en standard 230V ATX strømforsyning i PU slik at den kunne arbeides med uavhengig av hvorvidt ATV var påskrudd. Likevel er alt av mikrokontrollere og sensorer avhengig av strøm fra ATV for å kunne brukes sammen med PU. Det ble i løpet av de 3 første eksterne arbeidene klart at dersom datastudentene skulle få fortgang i arbeidet med analyseringen og forståelsen av nåværende system, måtte enten PU tas med til grupperom eller den måtte klones.

Etter møte med intern veileder Joakim Bjørk, fikk gruppen tilbud om en stasjonær maskin med kraftig CPU og GPU omtrent likt spesifisert som PU på ATV. Det er derfor en ypperlig kanditat for å montere en klonet disk fra PU. Lånemaskinen hadde fra før en 3TB spinndisk og en 512GB M.2 SSD.

Følgende rekkefølge og prosedyre ble brukt for å sikre i størst mulig grad at data fra PU ikke går tapt i kloningsprosessen.

1. 1TB M.2 SSD ble demontert fra PU
2. 512GB M.2 SSD ble demontert fra lånemaskin og 1TB M.2 SSD fra PU ble montert i stedet
3. For å unngå å endre en eneste sektor på original disk fra PU, ble hele disken (1TB) klonet til 3TB spinndisk på PU og deretter fjernet fysisk fra lånemaskin for å hindre sletting med uhell. Deretter ble 512GB M.2 SSD satt tilbake i lånemaskin.

4. Nå kan 1TB partisjonen som ble klonet over på 3TB disken krympes slik at den kan få plass på 512GB M.2 SSD. Kloningen består av 2 partisjoner (oppstart + OS) og setter ny størrelse på OS partisjon til 100GB.

5. Nå kan begge partisjonen kopieres sekvensielt til 512GB M.2 SSD disk med god plass til overs. OBS: En linux installasjon kan ikke enkelt klones direkte til en mindre disk, da det er for stor risiko for driftsikkerheten ved å redusere partisjonstørrelsen på en aktiv OS-partisjon i forkant av kloning. Derfor ble 3TB brukt som mellomstasjon og ny OS-partisjon kunne nå reduseres uten risiko for innholdet på original disk.

6. Deretter endres UEFI BIOS på lånemaskin til kun å starte opp M.2 disk i tillegg til andre relevante endringer for å starte Ubuntu 20.04.

7. Etter vellykket oppstart av klonet OS, lages det nå 2 nye bildekopier av tidligere nevnte oppsett på 100GB, slik at det er mulig for gruppen å senere gjenopprette systemet uten å måtte gjenta kloning av original disk fra PU

8. I tillegg kan oppstartsrekkefølge endres på lånemaskinen slik at klonet PU både er tilgjengelig på rask M.2 SSD disk og på spinndisk som reserve. Derfor kan arbeidet fortsette etter kun få minutters avbrekk på teststasjonen selv ved total havari av OS eller SSD.

For å skape en tilsvarende teststasjon må maskinvare fra PU dupliseres til testmaskin. Følgende utstyr er koblet til testmaskin for å simulere med størst mulig nøyaktighet:

- En Arduino Mega 2560 ble flashet med opprinnelig Throttle-Arduino kildekode
- En Arduino Mega 2560 ble flashet med opprinnelig Steering-Arduino kildekode
- En Arduino UNO ble til enhver tid flashet med nyeste versjon av Rosserial Gir-Arduino kildekode
- En Arduino Portenta H7 ble til enhver tid flashet med nyeste versjon av micro-Ros Gir-Arduino kildekode
- I en kortere periode ble VectorNav (VN-300) koblet til testmaskin for analyse av data

Etter oppstart av klonet PU på lånemaskinen ble det tatt en utskrift av rqt graph som

ble brukt til å analysere deler av ROS kommunikasjonen. Denne forståelsen ble benyttet ved utvikling av diagram for grensesnitt, se Figur 7.20.

Kunnskapen realtert til eksisterende systemet har økt etter at det ble mulig å programmere mikrokontrollere for testing direkte på VM eller klonet teststasjon.

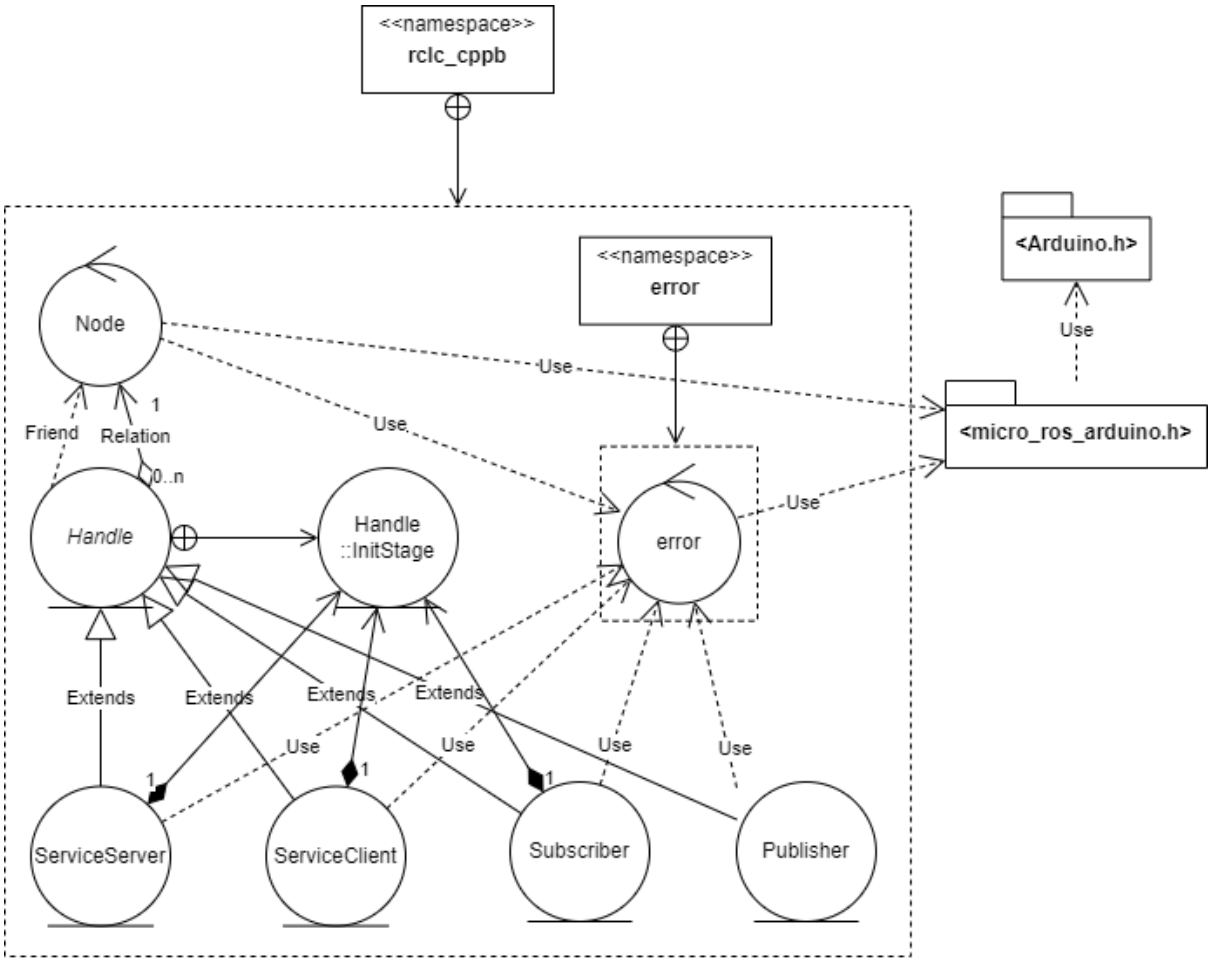
7.4.2 Konseptene bak girstyring via PU

I kravspesifikasjonen er det bestemt at PU skal kunne styre mikrokontroller for girskift gjennom et grensesnitt 7.2.1. Girsystemet som utvikles skal som minimum kunne styres av PU, men det er ingenting i veien for at systemet senere kan utvides til å styre mikrokontroller fra andre kilder enn PU.

Hovedformålet med ROS er at man kan sende instruksjoner raskt og driftsikkert mellom eventuelle sender(e) og mottaker(e). To-veis kommunikasjon er ønskelig for et girskiftsystem for at brukeren skal vite hvilket gir ATV er satt i.

7.4.3 rclc_cppb

For å gjøre programmet på gir-kontrolleren mer utvidbart og følge prinsippene for "clean-code", har det blitt utviklet et "stand-alone" Arduino-bibliotek med C++ "bindings" for rclc slik det er implementert av Micro-ROS-Arduino-biblioteket. Dette biblioteket er ment for generell bruk av Micro-ROS, og er testet grundig på Portenta H7. Det inneholder da fullt-ut objekt-orienterte "bindings" som dekker mesteparten av funksjonene til rclc, men som kan brukes på en måte som ligner mer på rclcpp.



Figur 7.26: Forenklet klassediagram av rclc_cppb. Error er teknisk sett ikke en klasse, men bare en samling med funksjoner i et eget namespace. Hverenkelt objekt som arver fra Handle holder en peker til et Node-objekt. Egenskap-grensesnitt er utelatt fra tegningen.

Klassearkitekturen til biblioteket er vist i fig. 7.26. Her går alle funksjonskall til rclc-funksjoner som potensielt kan feile, gjennom en error-handler, hvor da egen brukerdefinert funksjonalitet kan defineres gjennom makroer ERROR_ONCE og ERROR_LOOP. Fullstendig klassediagram ligger på side 2 på vedlegg A7.

Offisielt støttet maskinvare:

- Portenta H7

Funksjonalitet:

- Noder
- Publishers
- Subscribers

- Service Servers
- Service Clients
- "Message"-egenskap
  - Ferdig implementert for alle std_msgs
  - Makro for enkel implementasjon for brukerdefinerte typer
- "Service"-egenskap
  - Ferdig implementert for alle std_srvs
  - Makro for enkel implementasjon for brukerdefinerte typer

rclc_cppb er publisert på GitHub [**?**], og det er planlagt vedlikehold og videreutvikling av dette biblioteket utover sommeren 2023.

Planlagt funksjonalitet fremover:

- Timers
- Alternativ til rosidl-generering av kode for brukerdefinerte "messages" og "services"
- All resterende funksjonalitet i rclcpp

Hele koden med dokumentasjon ligger som en del av doxygen rapport for mikrokontroller-systemet i vedlegg A5.

7.4.4 Mikrokontrollerprogram (gir-kontroller)

Vi har i siste øyeblikk fått utfordringer med å få Micro-ROS-Agent til å kjøre på PU, så Portenta-implementasjonen er ikke fullstendig testet med hele systemet per dags dato. Men begge mikrokontrollerne klarer å oppnå styring av gir, samt å reagere på en rekke forskjellige instruksjoner fra PU.

Portenta H7 har to kjerner, M7 og M4, i prosessoren sin, og gruppen ønsker dermed å kjøre koden fordelt på de to kjernene i hver sin parallelle prosesss, og med et grensesnitt mellom dem som håndterer globale variabler. Kontrollsløyfen skal kjøre på M4-kjernen, mens PU-grensesnittet kjører på M7. Globale variabler ligger på M4-kjernen slik at kontrollsløyfen

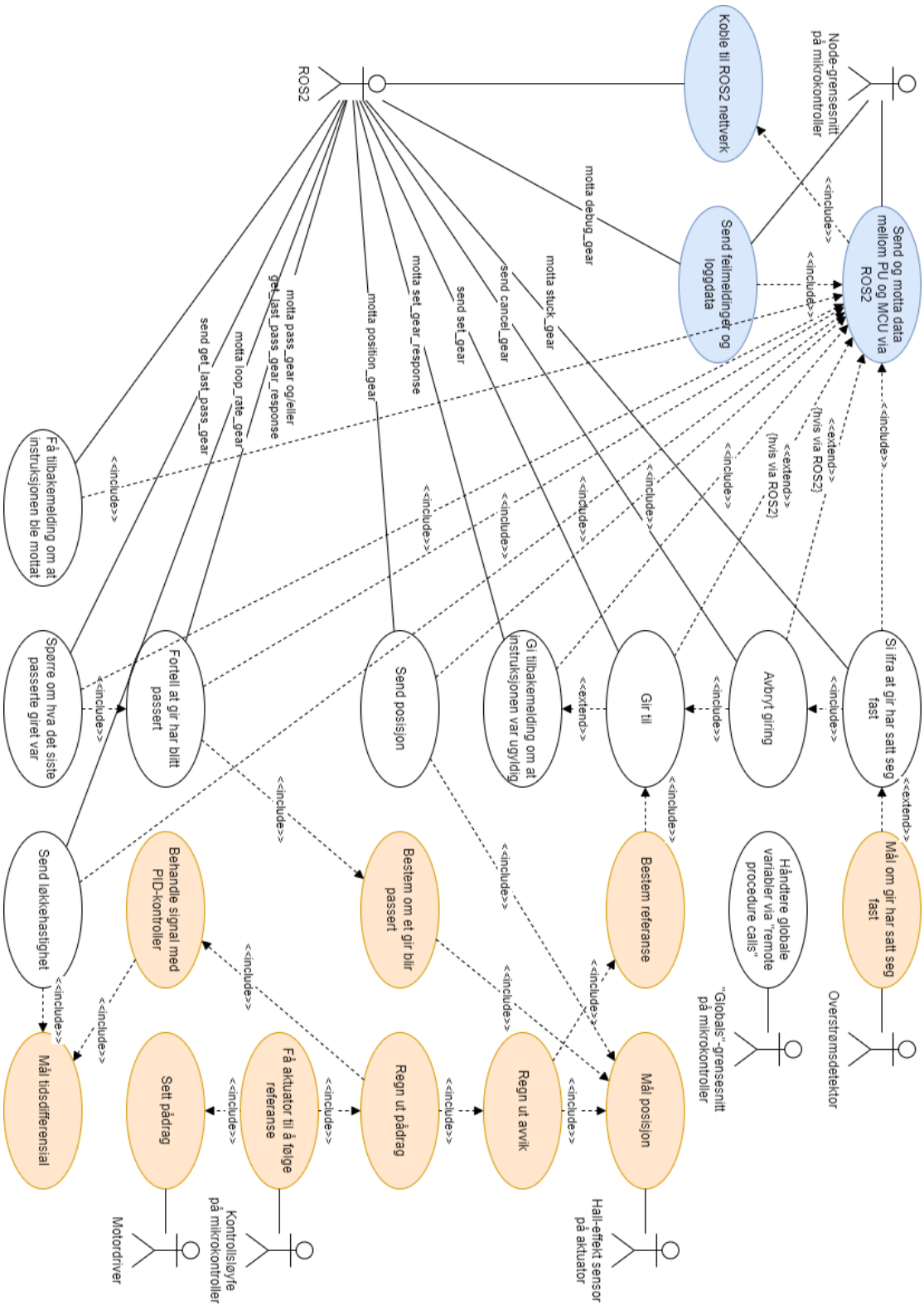
har direkte tilgang til dem, mens prosesseser på M7-kjernen må benytte RPC[13] for å nå dem.

Et tidlig utgangspunkt brukte kun seriellporten direkte for kommunikasjon med PU, men senere ble det besluttet å gå over til å bruke ROS (på Uno) og ROS2 (på Portenta) for å kommunisere med PU.

En fullstendig doxygen-rapport på koden, inkludert biblioteket rclc_cppb, ligger i vedlegg A5.

[13]Remote Procedure Call

7.4.4.1 Use-cases



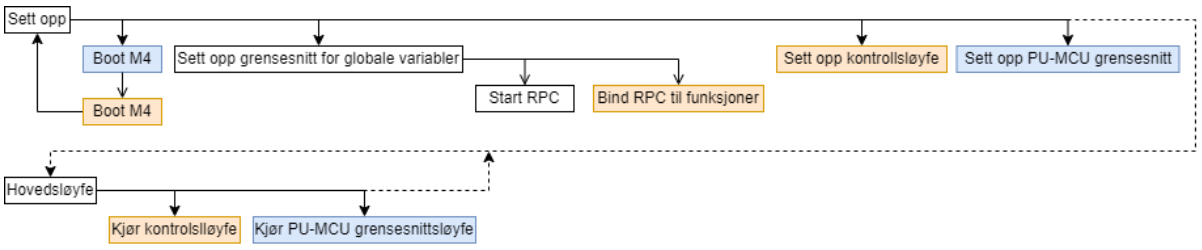
Figur 7.27: Use-case diagram for mikrokontrollerenhet. På Portenta H7 er de blå use-casene gjort på M7-kjernen og de oransje på M4-kjernen, mens de hvite innebærer funksjonalitet på tvers av kjernene.

Som vist i fig. 7.27, er det i grunn 11 use-cases sett fra resten av ROS2-systemet utenfor girkontrolleren, hvilket er:

- Koble til ROS2-nettverk
- Sende feilmeldinger og loggdata
- Sier fra at gir har satt seg fast
- Avbryt giring
- Gir til
- Gi tilbakemelding om at instruksjonen var ugyldig
- Send posisjon
- Fortell at gir har blitt passert
- Send løkkehastighet
- Spørre om hva det siste passerte giret var
- Få tilbakemelding om at instruksjonen ble mottatt

Utenom 'Koble til ROS2-nettverk' er alle disse use-casene nevnt ovenfor håndtert gjennom ROS2 topics. En oversikt over topics er vist i fig. 7.20.

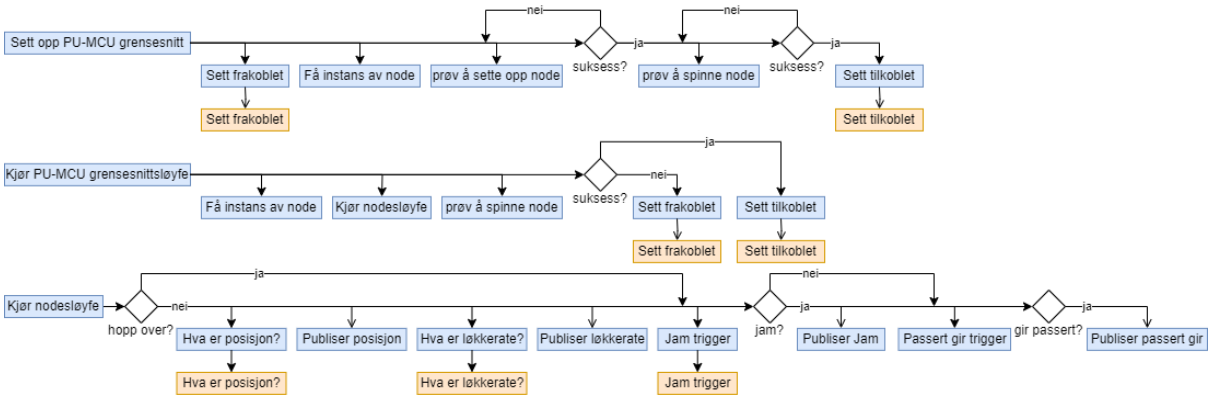
7.4.4.2 Funksjonell beskrivelse av oppstart og hovedløkke



Figur 7.28: Oppstartsprosedyre og hovedløkke til programmet illustrert i et funksjonelt diagram. Funksjoner markert i blått kjøres kun på M7-kjernen, mens de i oransje kjøres kun på M4-kjernen.

Oppstartsprosedyren til programmet er vist i fig. 7.28. Ved oppstart kjøres oppsett for M7-kjernen. Den utfører da "Boot M4" som starter oppsett på M4-kjernen parallelt.

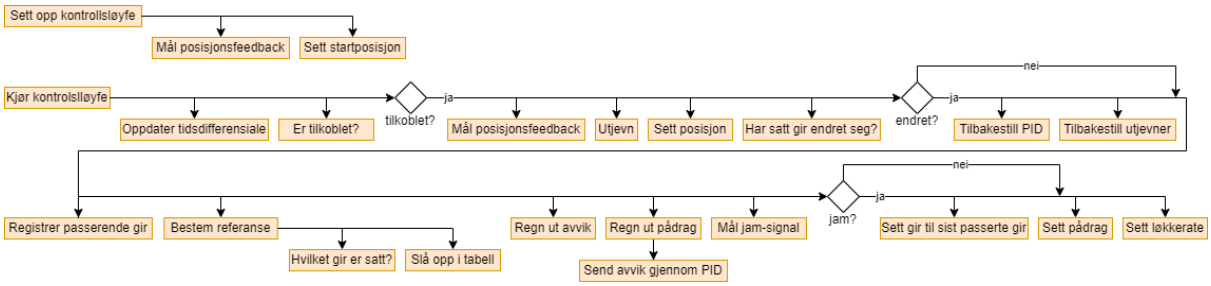
RPC-systemet blir deretter startet ifra begge kjernene, og RPC blir bundet til "cross-core" funksjonalitet på M4-kjernen. Deretter kjøres de respektive oppstartprosedyrene for kontrollsløyfen og PU-MCU grensesnittet på hver sin kjerne.



Figur 7.29: Oppstartsprosedyre og løkke for PU-MCU grensesnitt og nodeløkke. Globale variabler nås fra M7-kjernen via RPC og ender i et funksjonskall i M4-kjernen.

Som vist i fig. 7.29 vil PU-MCU grensesnittet ved oppstart forsøke å koble noden til ROS2-nettverket, helt til den får det til. Deretter setter den opp tilhørende topics.

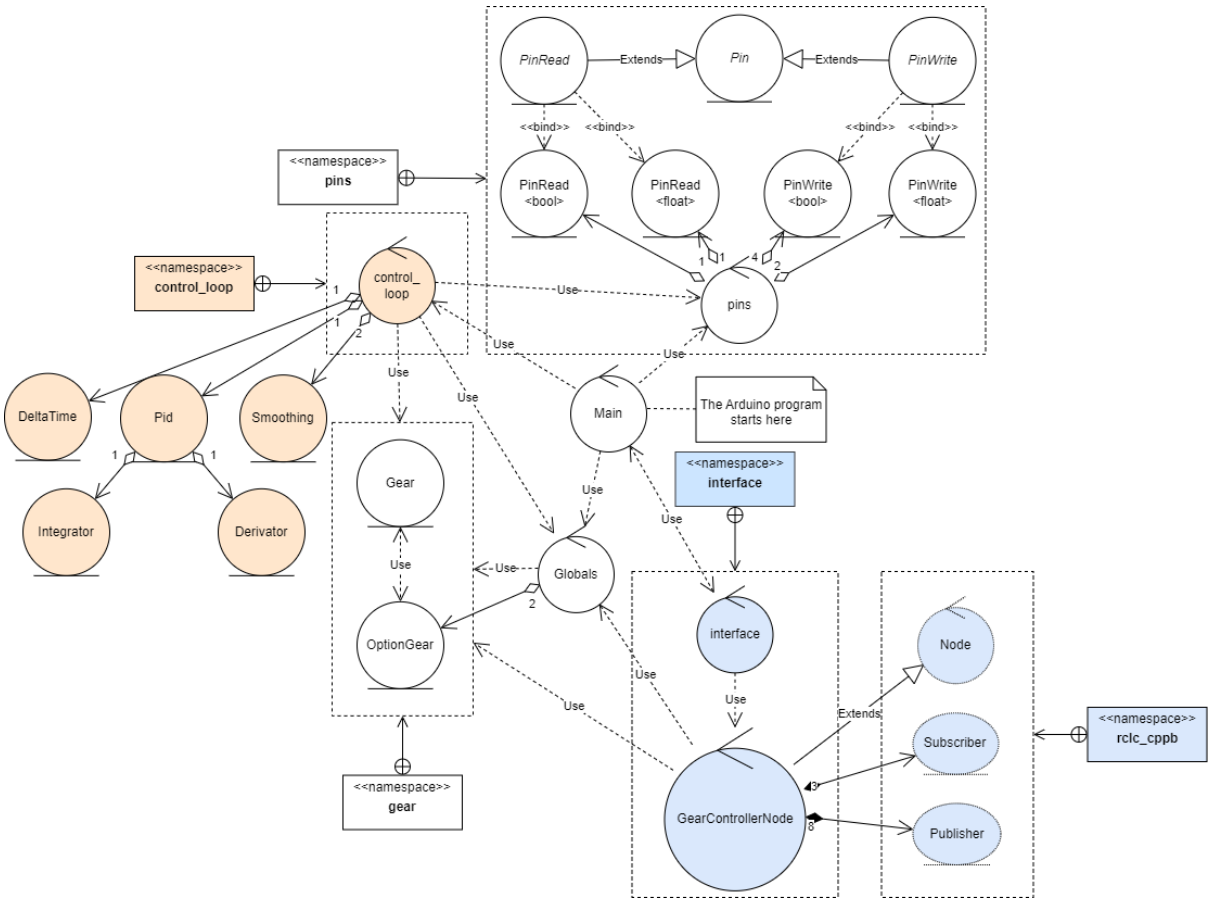
I PU-MCU grensesnittsløyfen vil noden spinnes, nodeløkken kalles, og deretter tilkoblingsstatus oppdateres. I nodeløkken publiseres det kontinuerlig på et par forskjellige topics.



Figur 7.30: Funksjonell fremstilling av kontrollsløyfen. Her foregår hele prosessen på M4-kjernen.

Koden for kontrollsløyfen ble utviklet tidlig i prosjektløpet og er laget etter spesifikasjoner i kap. 6.2. Den endelige versjonen er vist i funksjonelt diagram i fig. 7.30. Dersom noden ikke er tilkoblet vil kontrollsløyfen pauses. Den vil utføre noen ekstra steg dersom satt gir har endret seg siden sist løkkesyklus, eller dersom gir har satt seg fast (jam). Ellers er dette et enkelt kontrollsystem som har som oppgave å få aktuatorposisjon til å følge referanse som er forutbestemt av hvilket gir som er satt.

7.4.4.3 Arkitektur



Figur 7.31: Forenklet klassediagram for hele koden på Portenta H7 girkontroller. Enkelte samlinger med funksjoner og variabler i namespaces har blitt representert som «control»-klasser i dette diagrammet, men er navngitt med liten forbokstav. Main er heller ikke egentlig en klasse, men en samling med funksjoner. På Portenta H7 er de oransje elementene kun på M4-kjernen, og de blå elementene kun på M7-kjernen, mens de hvite er på begge. På Uno er alt på samme prosessorkjerne, og Rosserial er brukt i stedet for rclc_cppb.

Fig. 7.31 viser en forenklet oversikt av klassene brukt i programmet på mikrokontrolleren. Et fullstendig klassediagram ligger i vedlegg A7.

Pins-"namespacet" inneholder "wrapper"-klasser for lesing og skriving av signaler på IO-pins på mikrokontrolleren, og gjør det enklere å oppnå kompatabilitet for forskjellig hardware.

Gear-"namespacet" inneholder to "enums" som representerer gir-posisjon, og har også en "Option"-enum som også har en ekstra udefinert tilstand.

Alle RPC-bundne, globale variabler er håndtert i Globals-klassen, og gjør det mulig for begge kjernene å dele tilstand.

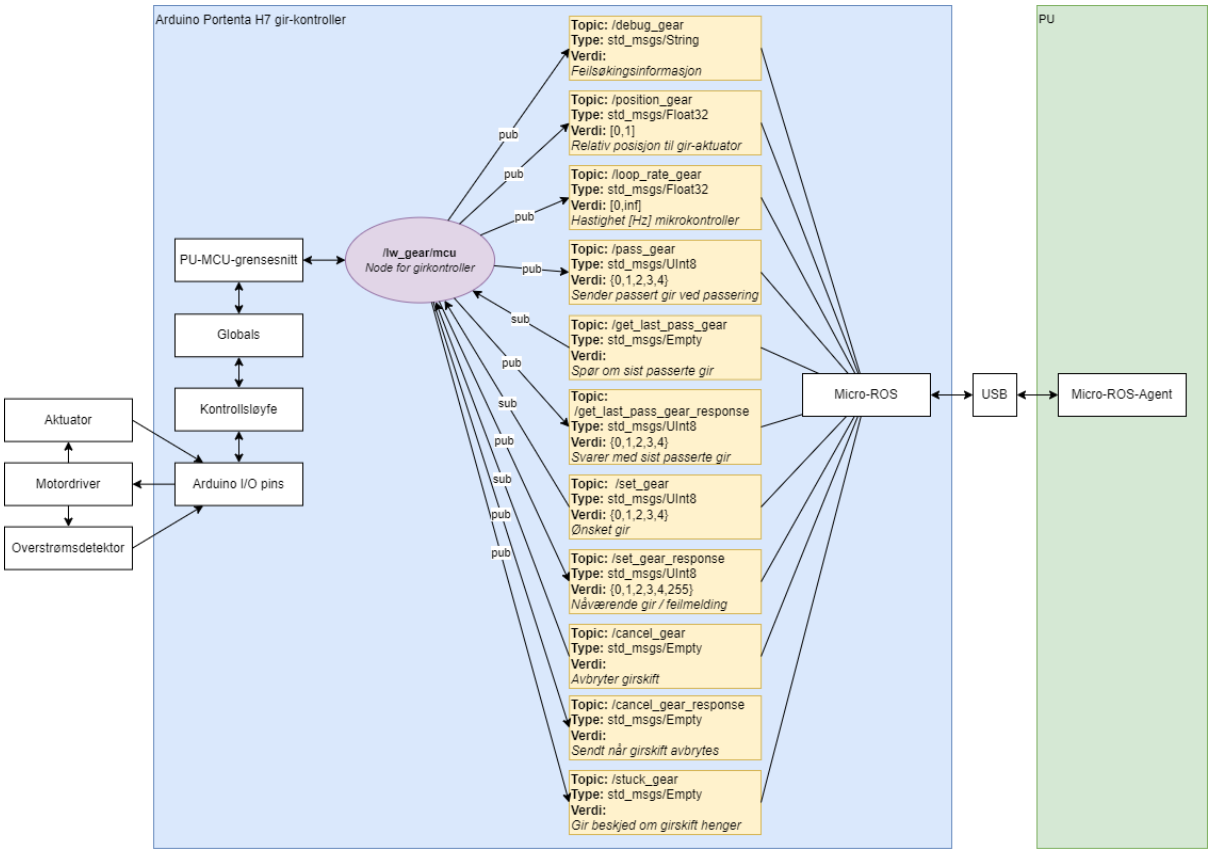
7.4.4.4 Topics

For å dekke alle use-cases som innebærer kommunikasjon med PU, har gruppen satt opp 11 forskjellige ROS2/ROS topics på noden i mikrokontrollerprogrammet, oppsummert i tabell 7.1.

Topic		Pub /sub?	Datatype
Portenta H7, micro-ROS, ROS2	Uno, Rosserial, ROS		
/lw_gear/debug_gear	N/A	Pub	std_msgs/msg/String
/lw_gear/position_gear		Pub	std_msgs/msg/Float32
/lw_gear/loop_rate_gear		Pub	std_msgs/msg/Float32
/lw_gear/pass_gear		Pub	std_msgs/msg/UInt8
/lw_gear/get_last_pass_gear	N/A	Sub	std_msgs/msg/Empty
/lw_gear/get_last_pass_gear_response	N/A	Pub	std_msgs/msg/UInt8
/lw_gear/set_gear	/set_gear	Sub	std_msgs/msg/UInt8
/lw_gear/set_gear_response		Pub	std_msgs/msg/UInt8
/lw_gear/cancel_gear	/cancel_gear	Sub	std_msgs/msg/Empty
/lw_gear/cancel_gear_response		Pub	std_msgs/msg/Empty
/lw_gear/stuck_gear		Pub	std_msgs/msg/Empty

Tabell 7.1: Komplett liste over topics i både Portenta- og Uno-implementasjonen av girkontrolleren.

Rosserial_python-noden som brukes som grensesnitt mellom Rosserial og ROS er ikke kompatibel med rosserial-subscriber'e med namespaces, og det er dermed blitt utelatt namespaces på topic'ene for å sette gir og avbryte giring. Den takler heller ikke tekstmeldinger veldig bra, ettersom dette fører til at grensesnittet sporadisk mister kontakt med noden, så debug-topic'en er også utelatt på Uno-implementasjonen. Topic for å spørre om det sist passerte gir ble lagt til seint i utviklingen, og ble ikke lagt til Uno-implementasjonen på grunn av mangel på tid. Som en midlertidig fiks har Uno-implementasjonen blitt modifisert til å kontinuerlig publisere pass_gear hele tiden, som gjør at lw_gear/head-noden kan utføre jobben sin uansett hvilken mikrokontroller som blir brukt. Nodene på PU har et makro-flagg som kan skru på kompatibilitet med Uno-implementasjonen uten namespace på subscriber-topics.



Figur 7.32: Interface diagram for gir-mikrokontroller, med fokus på ROS2-node og ROS2-topics. Alle topics viderekobles til PU via Micro-ROS-Agent. I Uno-implementasjonen brukes rosserial_python.py i stedet for å videreformidle topics. Noden er illustrert med en lilla boble, og topics med gule rektangler.

Grensesnitt, med fokus på noder og topics er vist i fig. 7.32. Micro-ROS kommuniserer med Micro-ROS-Agent over USB transport, til-og-fra PU. "Globals" er grensesnittet på tvers av kjernene.

7.4.4.5 Ved jam

Kontrollsløyfen er programmert til å sjekke om overstrømsdetektoren slår ut hver eneste løkke-syklus. Når jam inntrer, vil den sette gir til det forrige passerte giret, og sette et globalt trigger-flagg "is_stuck", som igjen leses og tilbakestilles av node-løkken som da reagerer med å publisere en beskjed på topic'en /lw_gear/stuck_gear.

Dette kan forbedres med å bruke hardware-interrupts, men fungerer bra slik det er.

7.4.4.6 Utjevning

For å redusere støy har en utgjevningsfunksjonalitet blitt lagt til, vist i fig. 7.31 som "Smoothing"-klassen. Denne fungerer med å ta en vektlagt sum av nåværende og forrige verdi som beskrevet i følgende uttrykk:

$$y[n] = \begin{cases} e^{-\sigma \Delta t[n]} y[n-1] + \left(1 - e^{-\sigma \Delta t[n]}\right) x[n] & n > 0, \\ x[n] & n = 0 \end{cases}. \tag{7.1}$$

hvor σ er et satt parameter som bestemmer endringsraten til y.

7.4.4.7 PID-kontroller

PID-kontrolleren fungerer som vist i fig. 6.3, hvor den deriverte og integrerte av avviket e blir regnet ut med Eulers metode.

Integrator ved Eulers metode er beskrevet i formel 7.6.

$$Ix[n] = \Delta t[n] x[n] + Ix[n-1] \tag{7.2}$$

Derivator ved Eulers metode er beskrevet i formel 7.3.

$$Dx[n] = \begin{cases} \frac{x[n] - x[n-1]}{\Delta t[n]} & n > 0, \\ 0 & n = 0 \end{cases}. \tag{7.3}$$

PID-kontrolleren er beskrevet i formel 7.4.

$$y[n] = x[n] K_P + Ix[n] K_I + Dx[n] K_D \tag{7.4}$$

Alternativer til Eulers metode for integrasjon og derivasjon Et alternativ er å utlede integrator og derivator ved bilineær-transform[28], som gir:

$$Dx[n] = \begin{cases} \frac{2}{\Delta t[n]} \left(x[n] - x[n-1]\right) - Dx[n-1] & n > 0, \\ 0 & n = 0 \end{cases}. \tag{7.5}$$

og

$$Ix[n] = \frac{\Delta t[n]}{2}\left(x[n] + x[n-1]\right) + Ix[n-1]. \tag{7.6}$$

Dette er en tyngre utregning, men gir et mer nøyaktig resultat. Et annet alternativ er å bruke en Runge-Kutta-metode, som vil kunne gi enda bedre nøyaktighet, men på bekostning av kjøretidshastighet. Eulers-metode viser seg å være tilstrekkelig for vårt bruksområde og har dermed forblitt uendret siden den først ble implementert.

7.4.4.8 Måling av aktuators stegrespons

Et separat arduino-script ble også laget for å måle aktuatorens steg-respons. Dette er ikke lenger i bruk, siden denne funksjonaliteten ble lagt til som en valgfri funksjonalitet på hoved-scriptet seinere, skrudd av/på med flagg ENABLE_MEASURE_IR.

Dette scriptet sendte et enhetssteg på pådraget til aktuator, målte posisjonen kontinuerlig underveis, og deretter printet posisjonsdataen ut på seriellporten formatert slik at det kan kopieres som en matrise direkte inn i Matlab.

7.4.5 Libserial

Dette biblioteket ble brukt på et punkt for å kommunisere mellom mikrokontroller og PU, men ble etterhvert utelatt fra systemet til fordel for Rosserial og Micro-ROS.

Libserial er et lettvekt, åpen-kildekode, general-purpose c/c++ bibliotek for å kommunisere med eksterne enheter via seriellport ifra Linux operativsystemet[29]. Biblioteket fungerer på egenhånd, uavhengig av ROS/ROS2. Med dette biblioteket kan man lese og skrive bytes på en valgt COM-port/seriellport. Hva slags data som sendes over COM-porten er opp til brukeren av biblioteket.

Å bruke et general-purpose bibliotek for seriellkommunikasjon er et hardware-uavhengig alternativ til å bruke et ROS/ROS2 bibliotek installert på mikrokontrolleren. Med denne løsningen stilles det ingen krav til mikrokontrollerens hardware, annet enn at den skal ha en seriellport (hvilket den har, så lenge ATmega-chippen brukes med Arduino-brettet. Om ATmega-chippen benyttes for seg selv vil det i det tilfellet trenges et eget USB-seriell-UART-grensesnitt som for eksempel FT232[30]).

En node på PU'en kan utstyres med dette biblioteket for å kommunisere med

mikrokontroller, og vil dermed fungere som et interface mellom ROS2 og mikrokontrolleren.

En utfordring med denne typen løsning er at COM-portene for seriellkommunikasjon tilegnes dynamisk, og kan variere selv om enhetene er koblet til de samme USB-portene. Dermed er det nødvendig å kunne velge riktig COM-port dynamisk ved å identifisere enhetene på portene.

En mulig løsning er å reservere en hittil ubrukt verdi i forespørselpakken for forespørsel om at mikrokontroller identifiserer seg. Deretter legg til en ny responstype for denne typen forespørsel, som bærer med seg et unikt ID-nummer (unikt for hver mikrokontroller). Slik kan man skille mikrokontrollerne fra hverandre.

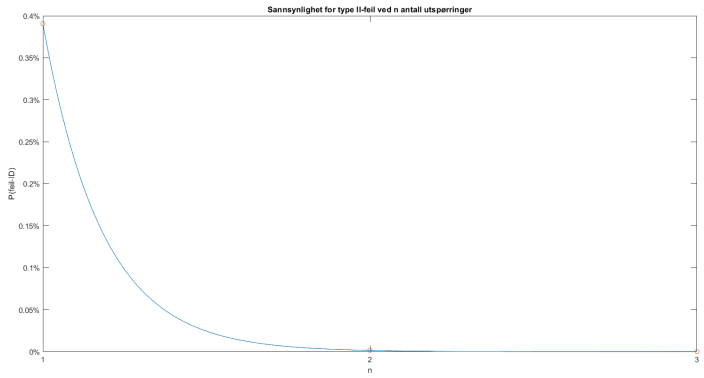
Dette kan derimot gå galt, dersom mikrokontrollere med denne identifiseringsfunksjonaliteten brukes sammen med mikrokontrollere som mangler den, ettersom de som mangler denne funksjonaliteten vil med uhell kunne tilfeldigvis sende en respons som er identisk den etterlyste UID[14]'en og dermed bli feilidentifisert. Dette er et usannsynlig, men mulig tilfelle. En løsning på dette problemet kan være å be om identifikasjon gjentatte ganger, for å redusere sannsynligheten for feilidentifikasjon betraktelig. Gitt en feil enhet med tilfeldig respons har hver utspørring har en 0.39 % sannsynlighet for feilidentifikasjon.

Sannsynlighet for feilidentifikasjon (type II-feil) etter n utspørringer har følgende sannsynlighetsfordeling:

$$P(\text{feil-ID}) = p^n \tag{7.7}$$

hvor $p = \frac{1}{256} = 0.39\,\%$. Sannsynlighetsfordelingen er illustrert i fig. 7.33.

[14]Unik identifikasjon/unik ID



Figur 7.33: Sannsynlighet for feilidentifikasjon $P(\text{feil-ID})$ ved forskjellig antall utspørringer n. Ved 3 utspørringer er sjansen for feilidentifikasjon på 0.000 006 %.

Noden kan ha en liste over tilgjengelige COM-portnavn, og rullere til neste navn på listen når den detekterer en feil UID ved utspørring.

Det er også mulig at dersom feil enhet blir spurt om å identifisere seg, kan dette forstyrre oppførselen til enheten og bli feilaktig tatt imot som en annen forespørsel. Det er derfor viktig at signaturen til ID-forespørselen er valgt for å unngå dette så godt det lar seg gjøre. En alternativ løsning kan være at en utspørring ikke sendes, men at noden kun lytter etter UID med en satt tidsbegrensning, hvor da UID sendes ved jevne mellomrom uten forespørsel fra de enhetene som skal identifiseres. Dersom noden ikke mottar noen UID innen satt tidsfrist, vil enheten anses som feil.

7.4.5.1 Seriellprotokoll

En seriellprotokoll for kommunikasjon mellom PU og gir-Arduino har blitt utarbeidet ved bruk av Libserial. Denne er laget i det tilfellet hvor kommunikasjonen skal foregå direkte over seriellporten uten et ROS/ROS2-bibliotek i mikrokontroller-enden, men har blitt utelatt i det endelige systemet til fordel for micro-ROS.

Gir-kontrolleren (Arduino) har blitt utvidet for å kunne motta og sende pakker, og en ROS2-node som bruker Libserial har blitt implementert for å kunne sende og motta pakker, etter spesifikasjoner gitt i protokollen. ROS2-noden har ikke blitt testet enda, men koden kompilerer. Arduinokoden ligger i vedlegg A5 og ROS2-nodens kode ligger i A8.

Første utkast av seriellprotokollen er beskrevet i detalj i vedlegg A3.

I etterkant av første utkast (som presentert i vedlegg A3), har en sjekksum blitt lagt til

returpakken, slik at pakker med korrupt data kan detekteres.

Pakken som sendes til mikrokontrolleren har en størrelse på 1-byte er på følgende form:

Byte	Representerer	Verdi
0	Gir-posisjon	0 => P 1 => R 2 => N 3 => H 4 => L _ => ugyldig

Tabell 7.2: Form på forespørselspakke som sendes fra PU til mikrokontroller.

og pakken som sendes fra mikrokontrolleren har en størrelse på 3-byte og er på følgende form:

Byte	Representerer	Verdi
0	Tilbakemelding-/status-type (hode)	0 => SetReference 1 => StatusPosition 2 => IsStuck
1 (for SetReference)	Gir-posisjon/feil	0 => P 1 => R 2 => N 3 => H 4 => L 255 => feil (ugyldig forespørsel)
1 (for StatusPosition)	Kontinuerlig posisjon	0..=255 => Aktuatorens posisjon, hvor 0 er maksimalt inntrukket og 255 er maksimalt utstrakt.
1 (for IsStuck)	Giret sitter fast	0
2	Sjekksum (hale)	0..=255 => byte 0 + byte 1 modulo 256

Tabell 7.3: Form på returpakke som sendes fra mikrokontroller til PU. Dersom det blir mottat en uforutsett verdi, eller sjekksum ikke stemmer så har pakken blitt korrupt.

7.4.6 ROS 1

ROS 1 noder kan kommunisere med ROS 2 noder via ROS Bridge. Se info om end-of-life noetic [31].

ROS 1 skiller seg blant annet fra ROS 2 ved at det opprettes en catkin pakke. For å kjøre ferdige ROS 1 pakker må kommandoen: source devel/setup.bash kjøres fra arbeidsmappen til pakken det gjelder. Det kan opprettes pakker med samme bibliotek som ROS 2, dvs roscpp (c++) eller rospy (python).

Det er opprettet én ROS 1 pakke for dette prosjektet. Det er nesten en direkte kopi av tidligere utført teknikk for Lone Wolf 2022 A2. Grunnlaget for at det ble opprettet en svært lik pakke var fordi det var behov for å legge til en ekstra mikrokontroller til systemet og fordi det var ønskelig å kunne kjøre girsystemet fra LoneWolfKongsberg2023 git repo uten å måtte være avhengig av eventuelle endringer utført på LoneWolfKongsberg2022 repo. Rosserial pakken er hentet fra [21].

Pakken rosserial_python benyttes for å aktivere Arduino enhetene. serial_node.py er kopiert opp i tre eksemplarer og det er opprettet en launch fil som kjører alle tre seriellnodene. Også denne launchfilen er sterkt inspirert av ros1.launch fra A2.

```
<launch>
  <!-- Launch serial_node -->
  <node pkg="rosserial_python" type="serial_node.py" name="serial_node"
    output="screen" />

  <!-- Launch serial_node_2 -->
  <node pkg="rosserial_python" type="serial_node_2.py" name="
   serial_node_2" output="screen" />

  <!-- Launch serial_node_3 -->
  <node pkg="rosserial_python" type="serial_node_3.py" name="
   serial_node_3" output="screen" />
</launch>
```

7.4.6.1 Rosserial

Mikrokontrollere implementert på ATV benytter Rosserial biblioteket for å opprette en ROS node.

Rosserial er et bibliotek laget for ROS 1. For å kunne inkludere biblioteket rosserial i eksempelvis Arduino IDE eller VS Code er det anbefalt å benytte operativsystemet Linux Ubuntu 20.04 LTS (Focal) sammen med ROS Noetic Ninjemys (nyeste utgave av ROS 1 pr 21. mai 2023). Via Noetic kan man installere Rosserial og deretter inkludere biblioteket i ønsket IDE. [32]

Er det hensiktsmessig å benytte rosserial sammen med enda en Arduino kontroller? Begge de allerede implementerte mikrokontrollerne er programmert med rosserial biblioteket

med de fordeler og ulemper det medfører.

Fordelen er at når IDE er satt opp med rosserial biblioteket er det svært enkelt å programmere en Arduino enhet for å kommunisere med et allerede eksisterende ROS oppsett. Opprettelse av node, navn og datatype på emnet (topic), funksjonalitet slik som publisering eller abonnering på emner programmeres i Arduino kildekoden. Linux operativsystemet trenger skrive/lese rettigheter til USB enheten for å kommunisere når ROS 1 kjører.

Ulempen er at en Arduino programmert med rosserial biblioteket ikke kan kommunisere med ROS2 uten at det opprettes en ROS Bridge til ROS 1. En Arduino kontroller satt opp med rosserial trenger også at det kjøres en seriell node i ROS 1 som har til hensikt å detektere og kommunisere med enheten.

Se mer om fordeler og ulemper ved ROS1 i delkapittel [7.4.6] og overgang til ROS2 i [7.4.7].

ROS mellomleddpakken rosserial_python har et par begrensninger som at services ikke er støttet, og at subscribers på mikrokontroller må være uten namespace.

Etter samtale med ekstern veileder Chris A. Brombach kom det frem at det er ønskelig å løse seriell kommunikasjon med Arduino uten å måtte bruke ROS Bridge.

7.4.7 ROS 2

Tidligere Lone Wolf prosjekter har satt opp PU med operativsystemet Ubuntu 20.04 LTS og blant annet installert ROS 2 Foxy og Gazebo 11. Lone Wolf Kongsberg 2022 [33] anbefaler å oppgradere ROS 2 Foxy til en nyere versjon grunnet end-of-life i mai 2023 [34]. Humble Hawksbill er nyeste distribusjonen av ROS 2, men krever Ubuntu 22.04 LTS [35] for å kunne kjøre. En oppgadering av ROS 2 vil da bety oppgradering av Linux distrubisjonen til en nyere versjon. Det er en kjent sak fra internet forum at det er kompatibilitetsproblemer med Ubuntu 22.04 LTS og Gazebo 11, noe Chris Brombach også har nevnt i e-post korrespondanse. Før en eventuell oppgradering av ROS 2 Foxy, med tilhørende oppgradering av Ubuntu 20.04 LTS, må alle slike usikkerheter testes ut før implementasjon. Se [Teststasjon: 7.4.1].

7.4.7.1 micro-ROS

Micro-ROS er et Arduino-bibliotek og grensesnitt som implementerer rclc for Arduino, og gjør det mulig å kommunisere via ROS2 topics og services mellom Arduino og en datamaskin koblet til Arduino via en transport (USB, WiFi eller Bluetooth). Dette biblioteket har ingen offisiell støtte for Arduino Uno eller Arduino Mega[36], og krever, i følge tidligere LoneWolf rapport [33], mer RAM enn det Arduino Uno og Arduino Mega har å tilby.

I sprint 6 ble flere offisielt støttede mikrokontrollere vurdert for bruk med micro-ROS, samt Arduino Zero og Arduino Due, men gruppen landet til slutt på å bruke Arduino Portenta H7, ettersom denne var den eneste tilgjengelig på lager hos Norske leverandører. Mer informasjon om Portenta H7 står i kap. 7.3.2.2.

Micro-ROS gir dermed a mulighet for å bruke både topics og services håndert i en node på mikrokontrolleren, uten de samme utfordringene libserial har å by på, og uten behov for ROS-bridge. Micro-Ros er kompatibel med ROS 2 Humble dersom det blir anledning til å oppdatere ROS 2 på det nåværende systemet.

Fra micro-ROS sin offisielle nettside har systemet blant annet følgende nøkkelegenskaper:

Microcontroller-optimized client API supporting all major ROS concepts

Seamless integration with ROS 2

Extremely resource-constrained but flexible middleware

Long-term maintainability and interoperability

Det har vist seg at å implementere egne service-typer for micro-ROS er vanskelig, ettersom rosidl for micro-ROS ser ut til å ikke fungere.

7.4.8 Revisjon av eksisterende Throttle-Arduino kildekode

Som nevnt i analysen av Steering-Arduino 7.1.3.1 kan funksjonaliteten til bryteren 7.2 revideres, eller det kan legges til abonnering på eksempelvis topic /uiSelector slik det er konkludert med i analysen av Throttle-Arduino 7.1.3.2.

Ved nærmere overveielser ble det bestemt at PU skal avgjøre hvorvidt girsystemet eller

operatørposisjon via UM600 skal aktiveres. Det er løst ved å la ønsket bruker-grensesnitt sende én publisering via topic /uiSelector (lw_gear/UiSelector ved micro-ROS) til head-noden 7.4.9. Les mer om bruker-grensesnitt TUI 7.4.12 eller GUI 7.4.13.

7.4.8.1 Hva skal revideres?

For å akkommodere ønsket funksjonalitet i girsystemet iht analysen 7.1.3.2 til Throttle-Arduino skal følgende funksjonalitet legges til eller endres:

- Det skal abonneres på /uiSelector med datatype Bool
- Det skal abonneres på /setThrottle med datatype Int64
- Det skal publiseres til /uiThrottle med datatype Int64
- Det skal publiseres til /getThrottle1 med datatype Int64 (kun for feilsøking, girsystemet bruker ikke denne)
- Det skal ikke være nødvendig å publisere ønsket verdi mer enn én gang til /uiSelector og /setThrottle for å oppnå ønsket resultat
- Alle funksjoner som tar i mot instruksjoner på UM600 relaterte topics må endres slik at de ikke aktiveres med mindre abonnert verdi på /uiSelector er usann og /inputSelector er sann
- Det må opprettes funksjoner som tar i mot instruksjoner fra girsystemets topics og de må ikke aktiveres med mindre abonnert verdi på både /uiSelector og /inputSelector er sann

Det er også ønskelig å endre filstrukturen til opprinnelig kildekode som består av throttle.ino og update.ino. Den nye strukturen kan oppsummeres slik:

- throttle.ino: Det som strengt tatt er nødvendig å ha med i hovedfilen, slik som inkludering av bibliotek og DAC samt funksjonene setup og loop
- variables.hpp: Globale variabler
- ros.hpp: Initialisering og deklarering av ROS relatert funksjonalitet inkludert callback funksjoner
- functions.ino: Globale funksjoner

7.4.8.2 Revisjon 4 av kildekoden

Modifikasjoner beskrevet i dette avsnittet er utført for å tillate at Throttle-Arduino utfører handling når og bare når det er bestemt av bruker gjennom ønsket grensesnitt. Koderevisjoner:

1. Splittet opp kildekoden fra to til fire filer
2. Alle endringer i kode foruten det som er nevnt i rev 1, 3 og 4
3. Lagt til subscriber på /uiEmergencyStop
4. Endret fra /setUiThrottle til /setThrottle

I tillegg til at deler av kildekoden er flyttet til hhv. ros.hpp, functions.ino og variables.hpp er det utført endringer og lagt til ny funksjonalitet i nevnte filer. Se Doxygen rapport for revidert utgave av Throttle-Arduino her A39. Se revidert kildekode til Throttle-Arduino her A37. Se den opprinnelige kildekoden til Throttle-Arduino her A36.

throttle.ino: I loop funksjonen i throttle.ino er betingelsen endret slik at det tas hensyn til hvorvidt det registreres et nødstopp signal fra enten via UM600 eller girsystemet. Kun ny kode eller endringer vises i kodesnutten. Se liste nedenfor for nøyaktig hvilke kodelinjer som er endret:

- Linje 2-3: Lagt til include for ros.hpp og variables.hpp
- Linje 6-9: Lagt til subscribers og publishers for nye topics ved hjelp av callback funksjoner 7.4.8.2
- Linje 14: Endret slik at data mottatt på /uiEmergencyStop kan tas hensyn til

```
1  //------------------------ Includes ----------------------------
2  #include "ros.hpp"
3  #include "variables.hpp"
4
5  void setup() {
6    nh.subscribe(SetThrottleSub);
7    nh.subscribe(uiSelectorSub);
8    nh.advertise(uiThrottlePub);
9    nh.advertise(getThrottle1Pub);
10 }
```

```
11
12 void loop() {
13   //For um600 and UI
14   if (emergencyStop.data and (!um600EmergencyStop or !uiEmergencyStop))
       {
15     ...
16   }
17 }
```

variables.hpp: Det er lagt til to nye globale variable i variables.hpp. Kun ny kode eller endringer vises i kodesnutten. Se liste nedenfor for nøyaktig hvilke kodelinjer som er endret:

- Linje 1: Lagt til pragma once
- Linje 3-4: Lagt til to nye variable. Med mindre det mottas data på /uiSelector eller uiEmergencyStop, er disse satt til å være usanne som standard.

```
1 #pragma once
2
3 bool uiSelector = false;
4 bool uiEmergencyStop = false;
```

ros.hpp: Merk at funksjonen uiEmergencyStopCb er opprettet for å behandle abonnering på topic /uiEmergencyStop som er ment å benyttes sammen med GUI 7.4.13. Kun ny kode eller endringer vises i kodesnutten. Se liste nedenfor for nøyaktig hvilke kodelinjer som er endret:

- Linje 1: Lagt til pragma once
- Linje 2: Lagt til include for variables.hpp
- Linje 8: Callback funksjonen um600ThrottleCb er endret slik at den tar hensyn til både /inputSelector og /uiSelector
- Linje 13-18: Kopiert kommentarer fra um600ThrottleCb og endret slik at den passer til funksjon i linje 19.
- Linje 19-25: Kopiert funksjonen um600ThrottleCb (med endringene gjort i linje 8) og endret navn til SetThrottleCb. Lagt til linje 23 der pwmThrottle skriver til data

som publiseres på /uiThrottle

- Linje 28-35: Lagt til callback funksjoner for topic /uiSelector og /uiEmergencyStop
- Linje 37-41: Oppretter ROS subscribers og publishers på nye topics

```
1  #pragma once
2  #include "variables.hpp"
3
4  std_msgs::Int64 uiThrottle;
5  std_msgs::Int64 getThrottle1;
6
7  void um600ThrottleCb(const std_msgs::Int64& pwmThrottle_) {
8    if (inputSelector && !uiSelector) {
9      ...
10   }
11 }
12
13 /*
14     From UI
15     Updates (and maps) the pwmThrottle which will be constrained to the
       throttle1 variable
16     in void loop which is then passed as parameter to the
     updateThrottle function.
17     Notice: valid datarange for data in this topic is -100 to 100,
     where 0 is idle and no brake.
18 */
19 void SetThrottleCb(const std_msgs::Int64& pwmThrottle_) {
20   if (inputSelector && uiSelector) {
21     int pwmThrottle1 = pwmThrottle_.data;
22     pwmThrottle = map(pwmThrottle1, minInputThrottle, maxInputThrottle,
       minPWM, maxPWM);
23     uiThrottle.data = pwmThrottle;
24   }
25 }
26
27 //Selects whether the ATV will be controlled by UI(True) or UM600(False
     )
28 void uiSelectorCb(const std_msgs::Bool& uiSelector_) {
29   uiSelector = uiSelector_.data;
```

```
30 }
31
32 //Detects whether emergency stop signal is sent from UI
33 void uiEmergencyStopCb(const std_msgs::Bool& uiEmergencyStop_) {
34   uiEmergencyStop = uiEmergencyStop_.data;
35 }
36
37 ros::Subscriber<std_msgs::Int64> SetThrottleSub("setThrottle",
       SetThrottleCb);
38 ros::Subscriber<std_msgs::Bool> uiSelectorSub("uiSelector",
       uiSelectorCb);
39 ros::Subscriber<std_msgs::Bool> uiEmergencyStopSub("uiEmergencyStop",
       uiEmergencyStopCb);
40 ros::Publisher uiThrottlePub("uiThrottle", &uiThrottle);
41 ros::Publisher getThrottle1Pub("getThrottle1", &getThrottle1);
```

functions.ino: Kun ny kode eller endringer vises i kodesnutten. Se liste nedenfor for nøyaktig hvilke kodelinjer som er endret:

- Linje 6-7: Funksjonen publishMsg er oppdatert slik at det også publiseres på /uiThrottle og /getThrottle1
- Linje 17: Funksjonen updateThrottle er endret slik at verdien fra throttle1 variablen skrives til aktuell data som sendes på topic /getThrottle1
- Linje 24: Funksjonen updateInputs er endret slik at den tar hensyn til både /inputSelector og /uiSelector

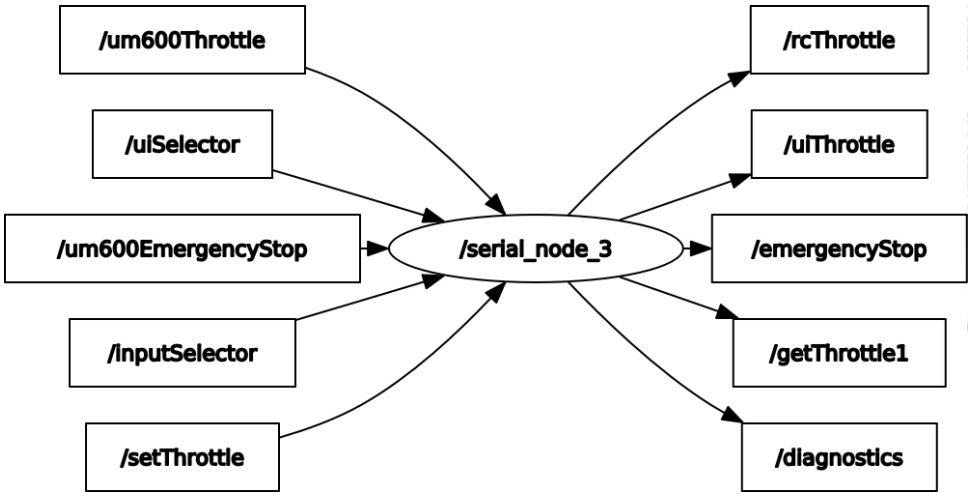
```
1  void publishMsg() {
2    ...
3    if (currentMillis - startMillis >= period)  //test whether the period
       has elapsed
4    {
5      ...
6      uiThrottlePub.publish(&uiThrottle);
7      getThrottle1Pub.publish(&getThrottle1);
8      ...
9    }
10 }
11
```

```
/*
   Updates the voltage to the DAC
*/
void updateThrottle(int throttle1) {
  ...
  getThrottle1.data = throttle1; //Debugging purposes
}

/*
   Updates the Input pin for the radiocontroller
*/
void updateInputs() {
 if (!inputSelector && !uiSelector) {
  ...
 }
  ...
}
```

7.4.8.3 Kommunikasjon med ROS

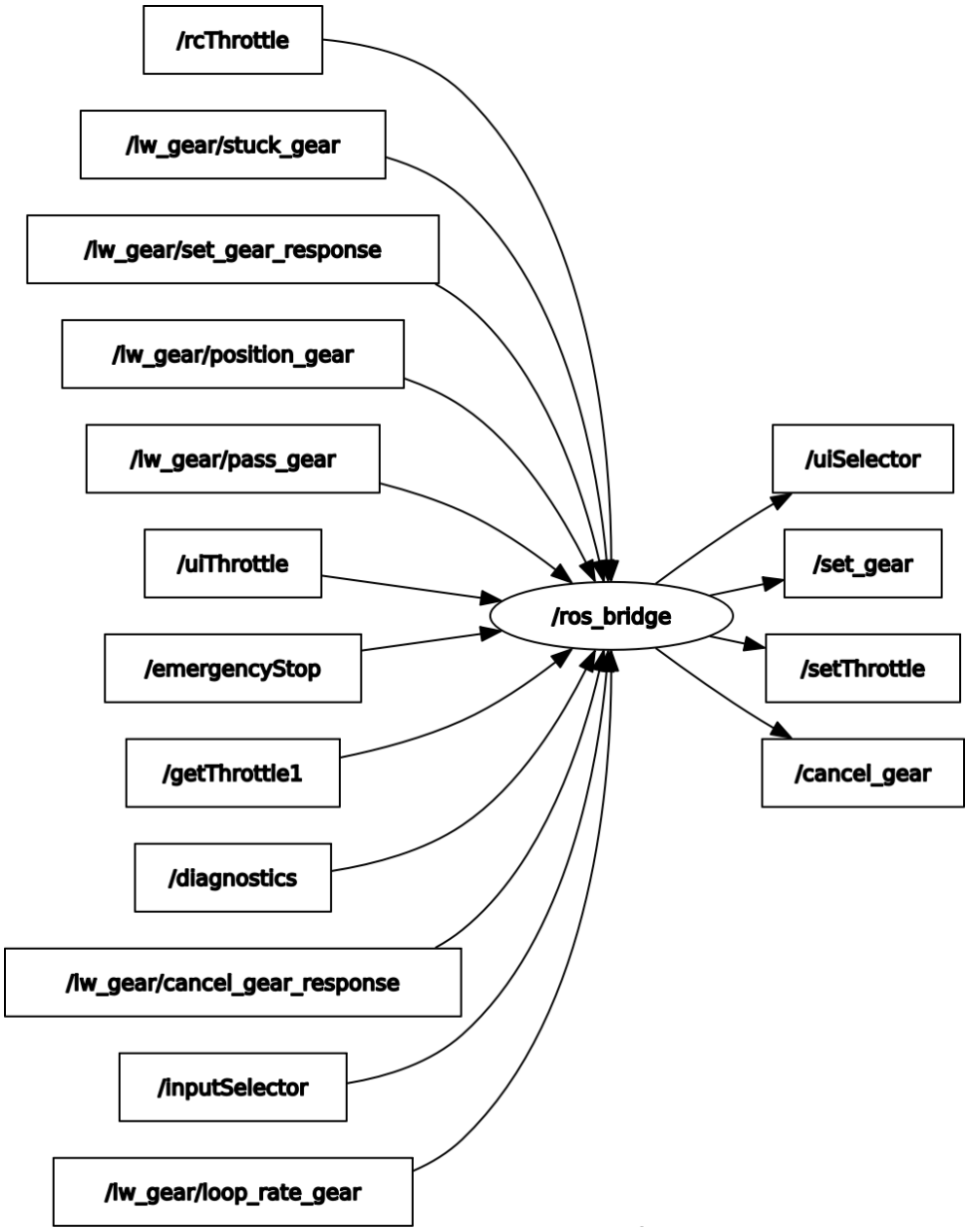
Den reviderte kildekoden tillater at gassrespons kan styres fra begge de tidligere grensesnittene uten å bli påvirket av nytt grensesnitt. Alle tidligere topics er fremdeles aktive og det er i tillegg lagt til abonnering på nevnte /uiSelector og /setThrottle, samt publisering til /uiThrottle og /getThrottle1.



Figur 7.34: rqt_graph utskrift av aktive topic etter revidert kildekode. MERK: /serial_node_3 representerer her Throttle-Arduino

7.4.8.4 Kommunikasjon med Head og TUI

Som tidligere nevnt i 7.1 er alle Arduino mikrokontrollere koblet til PU med to-veis kommunikasjon direkte med ROS 1 Noetic som via /ros_bridge kommuniserer med ROS 2. I figur 7.35 vises det til hvordan ROS 2 kommunikasjon sendes via /ros_bridge til både Throttle-Arduino og Gir-Arduino.



Figur 7.35: rqt_graph utskrift av trafikk gjennom /ros_bridge

Se mer om kommunikasjon relatert til Head og Tui her 7.4.9.4.

7.4.8.5 Test og verifikasjon av fjerde koderevisjon

Ved prosjektets start ble det raskt klart at oppdragsgiver ikke ønsker at nye implementasjoner på ATV skal forringe tidligere funksjonalitet. Grunnet dette ble det utført en egen test etter at original Throttle-Arduino kildekode ble oppdatert til fjerde revisjon og flashet til Throttle-Arduino montert på ATV.

Prosjekt:	Girmekanisme del 1		
Test ID:	Throttle 1	Dato:	08.05.2023
Testklasse:		Resultat:	G
Krav:			
Kravklasse:			

Pass/Fail Kriterie:	ATV skal aktivere brems når melding mottatt på /setThrottle -100 < x < 0 ATV skal aktivere gasspådrag når melding mottatt på /setThrottle 0 < x 100
Utførsel av testprosedyre:	1. Flash throttle_V4 til throttle arduino aller først. (Trippelsjekk at ikke steering arduino flashes) 2. Følg standard startprosedyre frem til motor skal starte (ikke start motor) og PU skal startes fra batteristrøm. Throttle arduino skal være tilkoblet ATV og PU. 3. Koble PU til internett og kjør git pull i LoneWolfKongsberg2023 mappen 4. Kill all terminals og kjør lw_light_start.sh 5. kontroller at /inputSelector er true ved å echo ut publiseringen. 6. Manuelt publiser true på /lw_gear/uiSelector 7. Kontroller at ATV er i PARK eller N. og at bremsen ikke er aktivert. 8. Manuelt publiser til /setThrottle verdier mellom -100 til 0. --------- 9. ATV skal nå stå i Park 10. Start motor 11. Manuelt publiser til /setThrottle verdier mellom 0 til 100
Hvis kriterie feiler:	Se på kode for å finne feil. Aktiver RCLCPP_INFO og les fra topics: /getThrottle1 og /uiThrottle
Resultat av test:	Når nevnte kriterier er oppfylt, dvs at /inputSelector og /uiSelector begge er usanne, fungerer radiokontrolleren på nøyaktig samme måte som før koderevisjon. Når nevnte kriterier er oppfylt, dvs at /inputSelector og /uiSelector begge er sanne kan det sendes instruksjon fra terminal om gasspådrag ved å benytte topic /setThrottle MERK: Grunnet tidsbegrensning ble det kun utført test med motor av og følgelig ble det da bare sendt verdier mellom -100 til 0. Siden all funksjonalitet virker på samme måte i Throttle-Arduino sin kildekode, er det ingen mulighet for at verdiene 0 til 100 ikke ville fungert på nøyaktig samme måte.

Figur 7.36: Vellykket test av fjerde kildekode revisjon for Throttle-Arduino

Se figur 7.45 som viser test av girsystemet del 2, der det fremkommer at verdi mellom 0 til 100 mottatt på /setThrottle fungerer slik det skal.

7.4.9 ROS 2 Head pakken

Head-noden fungerer som et grensesnitt mellom UI, TUI eller GUI og kommuniserer direkte med ROS 2 Foxy. Alle instruksjoner sendt fra bruker gjennom ønsket grensesnitt går via Head-noden før det videresendes til relevante mikrokontrollere.

Hovedfunksjonaliteten til Head-noden i tillegg til å videreformidle instruksjoner sendt fra bruker er å prosessere og reagere på eventuelle hendelser som kan oppstå ved utføring av nevnte instruksjoner. Head-noden vil automatisk respondere på feilmeldinger som kan oppstå ved girskift og utføre nødvendige korrigeringer underveis.

7.4.9.1 Direktekommunikasjon med Gir- og Throttle-Arduino uten å bruke Head-noden

Det er tatt hensyn til at bruker kan sende instruksjoner direkte til Gir-Arduino ved behov. Merk at alle instruksjoner sendt til nevnte mikrokontroller da ikke vil kvalitetssikres av Head-noden. Gir-kontrolleren vil ikke på egenhånd klare å fullføre instruksjonen dersom en låsesituasjon inntreffer. Det er likevel implementert flere sikkerhetsanordninger i kildekoden til Gir-Arduino i tilfelle instruksjon sendes utenom Head-noden 7.4.9. Det er også tilrettelagt for at det kan sendes instrukser til Throttle-Arduino direkte fra terminal uten å måtte kommunisere gjennom Head-noden.

Følgende terminalkommandoer kan benyttes for manuell publisering av VERDI (0 til 4) til Gir-Arduino og VERDI (-100 til 100) til Throttle-Arduino:

```
ros2 topic pub /set_gear std_msgs/msg/UInt8 "{data: VERDI} --once"
ros2 topic pub /setThrottle std_msgs/msg/Int64 "{data: VERDI} --once"
rostopic pub /set_gear std_msgs/msg/UInt8 "data: VERDI" --once
rostopic pub /setThrottle std_msgs/msg/Int64 "data: VERDI" --once
```

Se grafisk oversikt over tilgjenglige topics for Gir-Arduino 7.4.4.4 og den fjerde revisjon av Throttle-Arduino 7.34.

7.4.9.2 Klassen GearHead

Se Doxygen rapport for head.cpp her A34. Se kildekode til head.cpp her A35.

Se figur 7.37 for UML Class diagram av GearHead.

Figur 7.37: UML Class diagram for GearHead

I tillegg til nevnte doxygen rapport og kommentarer i kildekoden til head.cpp vil dette avsnittet oppsummere funksjonalitet i head pakken.

Head pakken er i sin helhet kodet i C++ og øverst i filen inkluderes nødvendige ROS rclcpp biblioteker samt datatyper for ROS2 meldinger. Det inkluderes kun det som er nødvendig for å kompilere og kjøre pakken. Alle datatypene benyttes av klassen.

Det er opprettet makrodefinisjoner for ulike konstante verdier og funksjonalitet som kan endres før kompilering. Det som er spesielt interessant å nevne av disse er ENABLE_ROSSERIAL_COMPATABILITY_MODE (Linje 28). Opphavet til dette er at det er utviklet to forskjellige Gir-Arduino mikrokontrollere for bruk sammen med girsystemet. Den ene er Arduino UNO 7.3.2.1 som benytter Rosserial 7.4.6.1 og den andre er Arduino Portenta H7 7.3.2.2 som benytter micro-ROS 7.4.7.1. Grunnet nevnte begrensninger på namespace ved abonnering på topics med Rosserial 7.1.4.1, kan man endre verdien på denne makrodefinisjonen til sann dersom Arduino UNO er koblet til girsystemet eller usann dersom Arduino Portenta H7 er koblet til.

Det er satt opp en enum for å representere de forskjellige tilstandene.

Konstruktøren oppretter et GearHead-objekt og initialiserer det ved å kalle konstruktøren til rclcpp::Node-klassen med 'head' som navn og oppretter ROS subscribers og publishers. De fleste er satt opp med namespace noen er satt opp uten og andre er satt til å aktivere namespace dersom makrodefinisjonen er satt til usann. Grunnet at eksempelvis subscriber for /inputSelector alltid er satt opp uten namespace er at Steering-Arduino publiserer uten namespace på denne og det var ikke ønskelig å revidere kildekoden til Steering-Arduino i løpet av dette prosjektet. Merk at konstruktøren setter tilstandsmaskinen 7.4.9.5 til å starte i tilstand Idle (State::Idle).

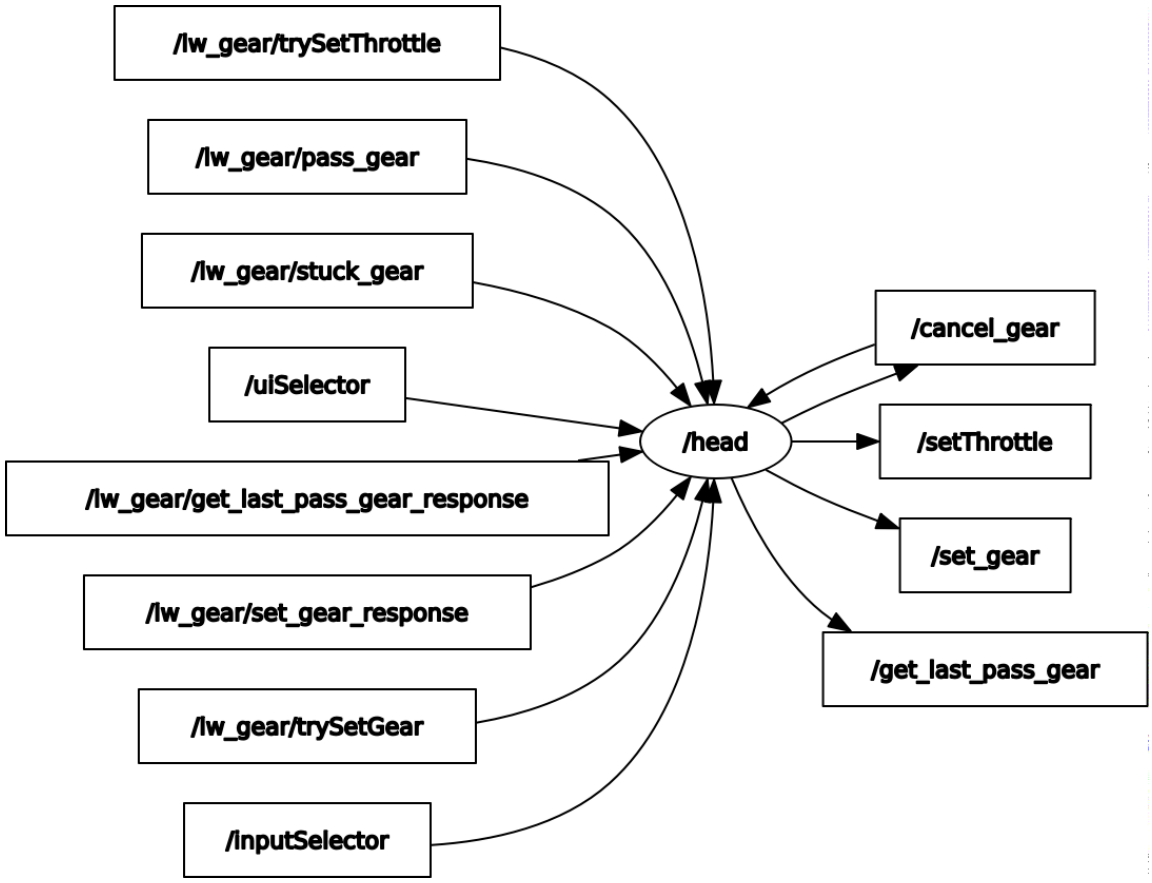
Alle medlemsfunksjoner og variable er private grunnet at de kun brukes internt i GearHead. Det er benyttet selvforklarende kode gjennom hele kildekoden.

Behandling av publisering og respons til mottatt data på abonnerte topic er lagt til hver sine respektive funksjoner. Callback funksjonene er kodet med hensyn på at de skal ha en interrupt funksjonalitet, men responsen er begrenset til hva som er ønskelig å utføre basert på nåværende systemtilstand. Se eksempel på hvordan er slik funksjon samarbeider med tilstandsmaskinen 7.4.9.5.

Det var opprinnelig tatt hensyn for at data fra VectorNav kunne benyttes for å bestemme hvorvidt ATV var i bevegelse eller ikke. Grunnet at analyse fra VectorNav ikke har blitt implementert i det nåværende girsystemet har det blitt lagt inn sikkerhetsmarginer. For å legge til ønsket sikkerhetsmargin uten at nevnte callback funksjoner risikerer å gå glipp av abonnerte data, er det lagt til en timer som vil kjøre i bakgrunnen og kalle tilhørende funksjon etter ønsket forsinkelse. Resten av systemet vil fortsette å kjøre parallelt uten å bære blokkert av ventetiden til timeren 7.4.9.5.

Det er ønskelig å revidere kildekoden til head.cpp ytterligere. Det innebærer å skille initialisering og deklarasjon av funksjoner i klassen i separate filer og finjustere gassrespons ved låsesituasjon og ventetid for girskift etter at brems er aktivert. Det nevnes mer om forbedret funksjonalitet i kaptittel om fremtidig utvikling 7.5.

7.4.9.3 Kommunikasjon med ROS

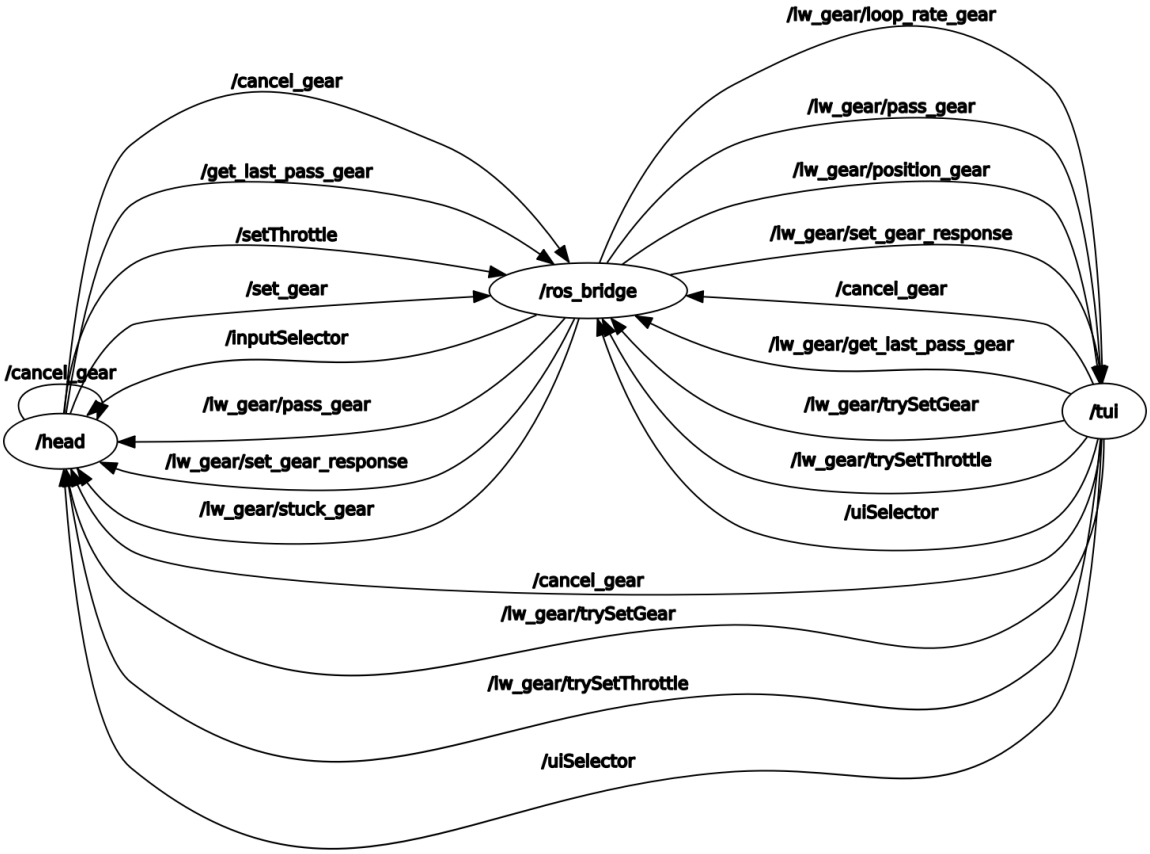


Figur 7.38: rqt_graph utskrift av aktive topic for Head-noden

7.4.9.4 Kommunikasjon med TUI-Noden

Les også avsnittet angående Text-Based User Interface 7.4.12.

I figur 7.39 vises en grafisk oversikt over kommunikasjon mellom Head-noden og Tui-noden. Kommandoen rqt_graf er kjørt når ROS 2 Foxy er aktivt og derfor er resten av systemet på ROS 1 siden kanalisert gjennom ros_bridge-noden.



Figur 7.39: rqt_graph utskrift av kommunikasjon mellom Head- og Tui-noden

7.4.9.5 Anti-Jam algoritme - Tilstandsmaskin

Les om planleggingstadiet til anti-jam systemet her 7.3.6.

Det er tre callback funksjoner som fungerer som en interrupt for gå til tilstand som er aktuell på tidspunktet det mottas data på abonnert topic. Disse tre funksjonene er hhv trySetGearCallback, stuckGearCallback og passGearCallback. Systemet er utformet slik at det er alltid interessant å reagere umiddelbart når det mottas instruks om girskift.

I funksjonen trySetGearCallback endres tilstanden til CheckPreviousGear for å kontrollere at forutsetningene for girskift er tilstede. Der kontrolleres at giret er gyldig og at det er ulikt hva det allerede er fra før.

I funksjonen stuckGearCallback tas det hensyn til hvilken tilstand systemet befinner seg i når det mottas melding fra Gir-Arduino om at giret er låst. Hvis tilstand er GearShifting settes ny tilstand til RevertGearShifting. Hvis tilstand er ReverGearShifting og det ikke er tre mislykkede forsøk, settes ny tilstand til ThrottleControl og det utføres en kort gassrespons.

Funksjonen passGearCallback er forklart i detalj nedenfor:

- Linje 1-2: Behandler mottatt data sendt på /pass_gear topic og tilordner den verdien til medlemsvariabel.
- Linje 3: Hvis tilstanden til systemet er GearShifting og ønsket gir er likt med mottatt data, så:
  - Linje 4: Går til tilstand Idle, som betyr at systemet venter på nye instruksjoner (merk at ved overgang til Idle, sendes beskjed til UI om at tilstand er Idle. Kode for UI informasjon om nåværende tilstand er plassert øverst i hver tilstand i tilstandsmaskinen.)
  - Linje 5: Brems slippes slik at bremsene ikke lenger er aktivert
  - Linje 6: Gir informasjon til UI at bremsene er sluppet
- Linje 8: Hvis tilstanden til systemet er RevertGearShifting og ønsket reversjeringssgir er likt med mottatt data, så:
  - Linje 9: Går til tilstand CheckPreviousGear, der det undersøkes videre om girskift skal tillates. (Merk at det som tidligere nevnt vil skrives til UI informasjon om ny aktiv tilstand
- Linje 11: Hvis sist passerte gir ikke lenger er likt med nyeste passerte gir, så:
  - Linje 12: Settes sist passerte gir til å være nyeste passerte gir
  - Linje 13: Informasjon om at girspak har passert gyldig reversibelt gir sendes til UI

```
1  void passGearCallback(const std_msgs::msg::UInt8::SharedPtr msg) {
2    this->m_PreviousTargetGear = msg->data;
3        if (this->m_current_state == State::GearShifting && this->
     m_TargetGear == msg->data){
4          this->transitionToState(State::Idle);
5              this->publish_set_throttle(SET_THROTTLE_VALUE_IDLE);
6              RCLCPP_INFO(get_logger(), "Brakes released");
7        }
8        if (this->m_current_state == State::RevertGearShifting && this->
     m_RevertGear == msg->data){
9          this->transitionToState(State::CheckPreviousGear);
10       }
11       if (this->m_PassGear != msg->data){
12         this->m_PassGear = msg->data;
13             RCLCPP_INFO(get_logger(), "Callback received: pass_gear");
14       }
15 }
```

Det er også lagt til tidsforsinkelsesfunksjoner i klassen som også har myndighet til å endre tilstand. Som et eksempel forklares timer funksjonaliteten for tilstanden EngageBrake i detalj nedenfor:

- Linje 14: Oppretter en smartpeker til et WallTimer objekt arvet fra den abstrakte baseklassen TimerBase fra rclcpp biblioteket
- Linje 11: I tilstanden EngageBrake kalles funksjonen brakeTimerCallback etter 3 sekunder (uten å blokkere resten av systemet i ventetiden), der:
  - Linje 3: Sikkerhetsanordning for å kun utføre handling dersom tilstanden faktisk er EngageBrake
  - Linje 6: Skriver ut informasjon til bruker via UI at bremser nå er satt på full styrke
  - Linje 7: Siden det nå er 3 sekunder siden brems ble aktivert, anses det som garantert at ATV står stille. Endrer så tilstand til GearShifting, slik at girskift kan skje når ATV er stillestående

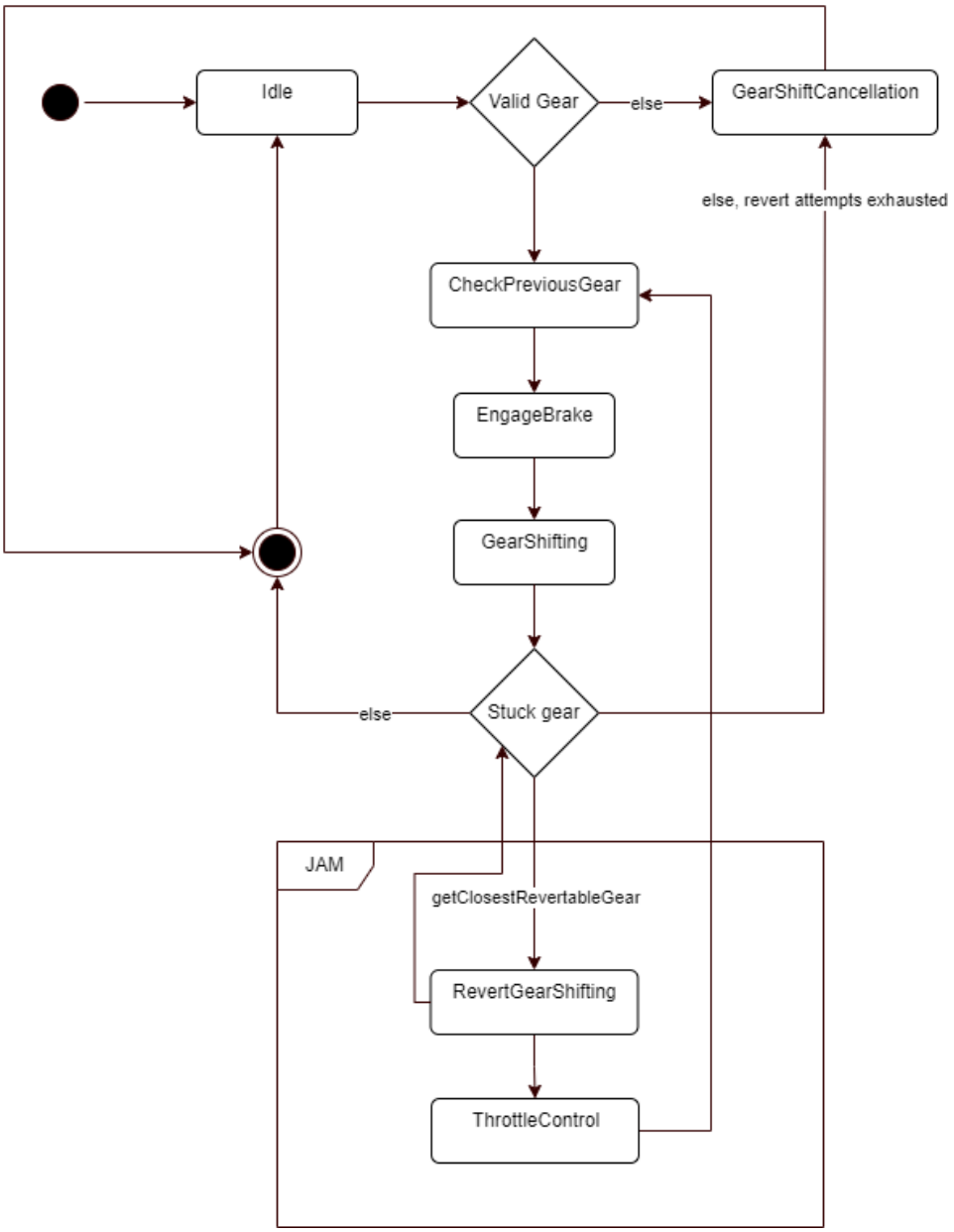
```
1 //Fra linje: 244 i head.cpp
2 void brakeTimerCallback(){
```

```
  if(this->m_current_state != State::EngageBrake){
        return;
      }
      RCLCPP_INFO(get_logger(), "F and R brakes set to FULL");
      transitionToState(State::GearShifting);
}

//Fra linje: 385 i head.cpp
m_brakeTimer = create_wall_timer(std::chrono::seconds(3), std::bind(&
   GearHead::brakeTimerCallback, this));

//Fra linje: 459 i head.cpp
rclcpp::TimerBase::SharedPtr m_brakeTimer;
```

Tilstandsmaskinen er forsøkt vist på forenklet vis i state diagrammet nedenfor 7.40 med tilhørende svært forenklet forklaring på de forskjellige tilgjengelige tilstandene til klassen. 7.4.9.5



Figur 7.40: State diagram for Head-noden

Tilstander

1. **Idle:** Avventer instruksjoner sendt fra bruker.
2. **CheckPreviousGear:** Verifiserer ønsket gir fra bruker samt hvorvidt ATV er i en tilstand der det er lov til å gire.
3. **EngageBrake:** Publiserer instruks til ATV om full brems og venter 3 sekunder før overgang
4. **GearShifting:** Publiserer instruks til ATV om ønsket girskift

5. **RevertGearShifting:** Publiserer instruks om girskift til nærmerste passerte gir av R, H eller L.
6. **ThrottleControl:** Publiserer instruks til ATV om 25 prosent gassrespons i 1 sekund
7. **GearShiftCancellelation:** Publiserer instruks til ATV om kansellering av gir

7.4.9.6 Test og verifikasjon av funksjonalitet

Med nevnte oppsett av teststasjon 7.4.1 var det mulig å utføre omfattende testing og verifisering av kildekode underveis i prosessen. Dette var spesielt gunstig sett i forhold til tester som krever at flere enheter eller undersystemer må kommunisere med hverandre.

Under utvikling av Head pakken var alltid følgende enheter og undersystemer tilkoblet:

- Identisk OS (klonet) fra Processing Unit på ATV
- Steering-Arduino med uendret kildekode fra tidligere prosjekter
- Throttle-Arduino med stadig nye revisjoner av kildekoden
- Gir-Arduino med stadig revisjoner av kildekoden + Aktuator og tilhørende kretskort, motordriver, etc
- Rosserial_python med serial_nodes for hver Arduino enhet
- ROS 1 med bridge
- Head pakken
- Stort sett alltid TUI pakken

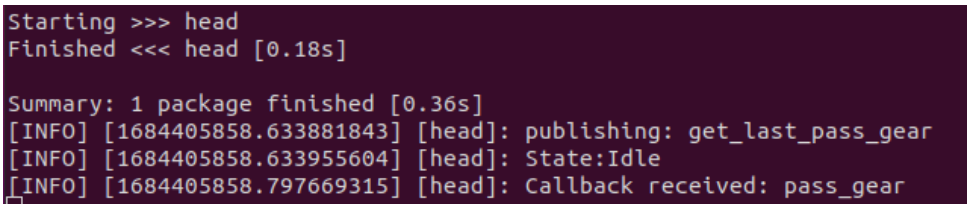
I løpet av sprint 8 og 9 ble Head pakken ferdig utviklet og testet. Etter flere vellykkede forsøk og simueleringer på teststasjonen, var det klart for fysisk test på ATV.

Det ble utført en fysisk test på ATV av tilstandsmaskinen før den var ferdig utviklet og ikke godkjent resultat. Erfaringene derfra medførte gjennomgang av tilstandsmaskinen overganger og på kontrollstrukturene i callbackfunksjonene. Det ble nødvendig å endre koden slik at brems fikk tid til å slippe opp før gasspådrag ble gitt.

Det vises til en rekke skjembilder tatt av teststasjon under testing av Head-noden og dens tilstandsmaskin nedenfor. Legg merke til at den kun skrives ut om endring av tilstand

slik: 'State::Idle' eller 'State::ChecPreviousGear'. Det er derfor et faktum at nåværende tilstand tilsvarer den siste utskriften med 'State::eksempeltilstand'. Callback funksjoner og andre viktige kontrollstrukturer skriver ut informasjon til skjerm der det er relevant. Alle utskrifter fra Head-noden i skjermbildene er iht. tilstandsmaskinen beskrevet visuelt i 7.40.

Figur 7.41 viser oppstarten av noden og skjermbilde det produserer. Tilstanden er Idle fordi det ikke er skrevet ut informasjon om ny tilstand etter Idle, slik som nevnt ovenfor.



Figur 7.41: Oppstart av Head-noden, tilstand: Idle

Figur 7.42 viser at noden mottar instruksjon om girskifte, endrer tilstand til CheckPreviousGear, endrer tilstand til EngageBrake, endrer tilstand til GearShifting og deretter tilbake til Idle. Legg gjerne merke til at hver tilstand utfører og skriver ut relevant informasjon.

Figur 7.42: Flere vellykkede girskift uten at låsing er fremprovosert

Figur 7.43 viser en fremprovosert en låsing av gir (JAM) ved å fysisk holde igjen aktuator. Gir-Arduino publiserer da en Empty verdi på /lw_gear/stuck_gear. Dette detekteres

av stuckGearCallback og siden antall forsøk på å omgå låsing ikke er utmattet endres tilstander deretter. Legg spesielt merke til utskriftene av 'Callback received: pass_gear' der Head-noden underveis lagrer siste vellykkede passering av girene Reverse, High og Low. Dette fordi det er disse girene det er interessant å forsøke å gire tilbake til ved en eventuell låsing for å påføre litt akselleras jon.

```
[INFO] [1684407568.869093063] [head]: publishing: set_gear
[INFO] [1684407568.869123093] [head]: Attempting to gear after JAM, attempt: 1
[INFO] [1684407568.869143927] [head]: State:ThrottleControl
[INFO] [1684407568.869171945] [head]: Throttling to rotate transmission
[INFO] [1684407568.869192575] [head]: publishing: setThrottle 25
[INFO] [1684407568.889950712] [head]: Callback received: set_gear_response
[INFO] [1684407568.902159570] [head]: Callback received: stuck_gear
[INFO] [1684407568.902191147] [head]: Throttling to rotate transmission
[INFO] [1684407568.902232875] [head]: publishing: setThrottle 25
[INFO] [1684407569.902343956] [head]: State:CheckPreviousGear
[INFO] [1684407569.902395839] [head]: State:EngageBrake
[INFO] [1684407569.902451100] [head]: publishing: setThrottle -100
[INFO] [1684407572.902568460] [head]: F and R brakes set to FULL
[INFO] [1684407572.902599597] [head]: State:GearShifting
[INFO] [1684407572.902628977] [head]: publishing: set_gear
[INFO] [1684407572.919612795] [head]: Callback received: set_gear_response
[INFO] [1684407573.943618109] [head]: Callback received: pass_gear
[INFO] [1684407574.303991676] [head]: Callback received: pass_gear
[INFO] [1684407575.021486128] [head]: Callback received: pass_gear
[INFO] [1684407575.737986962] [head]: State:Idle
[INFO] [1684407575.738040295] [head]: publishing: setThrottle 0
[INFO] [1684407575.738084381] [head]: Brakes released
[INFO] [1684407575.738103775] [head]: Callback received: pass_gear
```

Figur 7.43: Vellykkede girskift ved én fremprovosert låsing

Figur 7.44 viser hva som skjer dersom tilstandsmaskinen har forsøkt totalt 3 ganger å oppnå brukers ønske om girskift, etter å ha forsøkt å gire tilbake til sist reversible gir mellom hvert forsøk. Det betyr på ingen måte at systemet er låst eller krasjet. Head-noden er når som helst klar til å ta i mot instruksjon om girskift via den interrupt formulerte callback funksjonen trySetGearCallback. Den funksjonen vil alltid undersøke om forholdene er til rette for et nytt girskift og deretter utføre ønsket gir om gyldig verdi er sendt. Tanken bak dette er at dersom tilstandsmaskinen gir opp, kan personell gå bort til ATV, dytte på den og forsøke hele prosessen på nytt.

```
[INFO] [1684408069.471867216] [head]: Callback received: set_gear_response
[INFO] [1684408069.484897399] [head]: Callback received: stuck_gear
[INFO] [1684408069.484927451] [head]: Attempts exhausted, JAM circumvention fail
ed
[INFO] [1684408081.018058293] [head]: Callback received: trySetGear: 0
[INFO] [1684408081.018102911] [head]: State:CheckPreviousGear
[INFO] [1684408081.018136730] [head]: State:EngageBrake
[INFO] [1684408081.018159662] [head]: publishing: setThrottle -100
[INFO] [1684408084.018232648] [head]: F and R brakes set to FULL
[INFO] [1684408084.018264499] [head]: State:GearShifting
[INFO] [1684408084.018275441] [head]: publishing: set_gear
[INFO] [1684408084.036781277] [head]: Callback received: set_gear_response
[INFO] [1684408085.081153561] [head]: Callback received: pass_gear
[INFO] [1684408085.442014328] [head]: Callback received: pass_gear
[INFO] [1684408086.161897187] [head]: Callback received: pass_gear
[INFO] [1684408086.879752157] [head]: State:Idle
[INFO] [1684408086.879785154] [head]: publishing: setThrottle 0
[INFO] [1684408086.879793397] [head]: Brakes released
[INFO] [1684408086.879818812] [head]: Callback received: pass_gear
```

Figur 7.44: Vellykket girskift etter at tilstandsmaskinen ga opp å fullføre det opprinnelig ønskede giret

Siste og avgjørende test i Teknologiparken ble utført vellykket på grunnlag av test og simulering vist ovenfor.

Prosjekt:	Girmekanisme del 1 og del 2 (HEAD/TUI)		
Test ID:	Head-Tui	Dato:	19.05.2023
Testklasse:		Resultat:	G
Krav:			
Kravklasse:			

LONE WOLF

Pass/Fail Kriterie:	Systemet skal kunne iverksette ønsket gir fra UI/TUI Dersom JAM oppstår skal statemachinen få gir ut av låst posisjon og prøve maks 2 ganger til Dersom JAM har oppstått 3 ganger skal man kunne gi ny instruksjon om girskift og starte state machinen på nytt.
Utførsel av testprosedyre:	1. Koble alt fysisk til ATV (gir arduino, aktuator) 2. Koble PU til internett og kjør git pull i LoneWolfKongsberg2023 mappen 3. Følg standard startprosedyre og start motor. PU skal startes fra batteristrøm. 4. Verifiser at Radiokontroller fungerer som før prosjektet startet (bryter:MANUAL) 5. Kill all terminals og kjør lw_test_start.sh 6. Sett bryter til RADIO 7. Kontroller at /inputSelector er true ved å echo ut publiseringen. 8. Kontroller at ATV er i PARK eller N. og at bremsen ikke er aktivert. 9. Benytt TUI for ønsket girskift --------- Forsøk å gire til og fra alle mulige gir Forsøk å få JAM situasjon ved å holde igjen girspak i begge retninger dersom det ikke naturlig oppstår.
Hvis kriterie feiler:	Se på kode for å finne feil. Aktiver RCLCPP_INFO og les fra topics Verifiser fysisk oppsett og referer med elektrostudent angående Gir-Arduino og aktuator.
Resultat av test:	Ca 20 minutter gikk bort til finkalibrering av jam deteksjon for Gir-Arduino. Det var brukt feil tverrsnitt på ledninger til og fra kretskort og motordriver. Dette fordi gruppen til stadighet har koblet aktuator til og fra UNO oppsett og H7 oppsett. Det er naturlig at feil kan oppstå. Vellykket test uten JAM: Brems aktiveres alltid før girskift foretas og det tar 3 sekunder til girskift iverksettes. Head-noden endrer tilstand iht. til forventet oppførsel underveis i girskifteprosessen. Brems slippes etter endt girskift. MERK: Ved noen få tilfeller rapporterer Gir-Arduino om feil gir, spesielt ved Lavgir, da ATV sier den er i Høygir, men Gir-Arduino tror den er i Lavgir. Testen gjelder forøvrig Head- og TUI-noden og ikke direkte relatert til elektronikkingeniør arbeidet. Vellykket test med JAM: Tilstandsmaskinen klarte med 100% sikkerhet å få giret til ønsket plassering etter x antall forsøk. Det ble forsøkt å holde igjen girskift-stag med hånden 1, 2 og 3 ganger. Når den ble holdt igjen 3 ganger så vises forventet resultat om at "Jam circumvention has failed". Det er iht. til forventingene at ny instruks om girskift kan sendes fra TUI uten å starte systemet på nytt og det fungerte helt fint. MERK: Det var ved ett tilfelle tvil om tilstandsmaskinen ga 1 sekunds gassrespons i N (noe som er meningsløst). Det antas at dette kan skyldes at Gir-Arduino trodde den var i giret over, altså H. Dette må sees i sammenheng med unøyaktighetene forklart ovenfor. Se video fra testene på vedlagt minnepenn. Test utført av Joachim, Martin og Sigurd

Figur 7.45: Vellykket test av Head- og TUI-noden og dermed girsystem del 1 og 2

7.4.9.7 Bruk av Head-noden i fremtidige systemer

Som vist i figur 7.38 er det for tiden en begrensning på funksjonaliteten med tanke på å kjøre ATV fullverdig via Head-noden. Noden abonnerer på /trySetGear og /trySetThrottle, men det bør legges til en ny abonnent på eksempelvis /trySetSteering. Selv om noden abonnerer på /trySetThrottle, er det ikke opprettet en tilstand for normal kjøring. Noden er kun

satt opp for å manipulere gass og brems via automatikk ved girlåsing. Se oppsummering av hva som skal til for å bruke head-noden eksempelvis sammen med um600:

- Legg til funksjonalitet for å reagere på abonnert data fra topic /trySetThrottle
- Abonner på eksempelvis /trySetSteering og legg til funksjonalitet for dette
- Deretter må eventuell fremtidig node konfigures slik at den publiserer på /trySetGear, /trySetThrottle og /trySetSteering til Head-noden

7.4.10 Oppstartsskript for girsystemet for bruk på PU

LoneWolf 2022 opprettet et praktisk og driftssikkert oppstartsskript for PU A2. Med utgangspunkt og inspirasjon fra oppstartsskriptet lw_start.sh utviklet av LoneWolf 2022, ble dette skriptet slanket, tilpasset og justert for å fungere som et individuelt oppstartsskript for girsystemet. Merk at dette oppstartsskriptet ungår å starte noe annen funksjonalitet enn det som er nødvendig for at girstystemet skal kunne benyttes på ATV se skriptet lw_girsystem_start.sh her A41.

Som oppsumemring kan det nevnes at skriptet aktiverer relevante mikrokontrollere, utviklermiljø (ROS med bridge) og starter de tre pakkene utviklet av LoneWolf 2023.

Det ble også før dette tilpasset et oppstartsskript for å teste hvorvidt endringer i kildekoden til Throttle-Arduino påvirket tidligere funksjonalitet 7.4.8.5. Se skriptet lw_light_start.sh her A40. Dette skriptet er utdatert og er bare nevnt grunnet bruksområde i løpet av utviklingsprosessen.

7.4.11 Installasjonsskript - SW6

For videreutvikling av systemet er det viktig at nødvendig programvare og dets avhengigheter installeres slik at kode kan kompileres, testes og implementeres. Tidlig i prosjektutviklingsfasen ble det foretatt flere forsøk på å bygge tidligere prosjekter for å få innsikt i funksjonaliteten av ROS, men endeløse kompileringsfeil førte til en hindring i progresjonen på forståelsen av de tidligere systemene. Ubuntu 20.04 LTS med ROS1, ROS2 og Gazebo er installert på PU som nevnt i underkapittel 7.4.6 og 7.4.7, men ettersom videreutvikling foregår på egne datamaskiner må programvaren installeres ikke bare de fire datamaskinene til elektroingeniør-/ og dataingeniør-studentene, men også fremtidige

utviklere. For å effektivisere og forenkle installasjonsprosessen ble et bash (Bourne Again SHell) skript utviklet. Skriptet er laget for Ubuntu 20.04 med ROS2 Foxy og ROS Noetic. Et tilsvarende installasjonsskript ble i første omgang utviklet for ROS2 Humble, og kan brukes for fremtidig utvikling av systemet.

7.4.11.1 Behov for installasjonsskript

Risikovurdering avdekket en sannsynlighet for at PU eller SSD installert i PU kan ta skade, som vil føre til at utviklermiljø må installeres på nytt. Ettersom en lokal sikkerhetskopi ble tatt av systemet tidlig i prosjektet kan sikkerhetskopien benyttes for å komme tilbake til en tidligere utgave av systemet, men avhengig av hyppighet på sikkerhetskopier vil data bli tapt. Det anbefales å fortsette å benytte og videreutvikle installasjons-skriptene, og lage nye skript som med få tastetrykk kan opprette et utviklermiljø for videreutvikling dersom uhellet forekommer.

7.4.11.2 Funksjonalitet av skript

Installasjons-skriptene for Ubuntu 20.04 er tredelt og følger installasjonsprosedyren beskrevet av ROS utviklerne, samt installeres avhengigheter fra tidligere prosjekter. Skriptene er selvstendige og ved behov kan for eksempel kun ROS 2 Foxy Fitzroy eller ROS 1 Noetic Ninjemys installeres på Ubuntu 20.04 eller ROS 2 Humble Hawksbill på Ubuntu 22.04.

Installasjonsskriptene installerer nødvendig programvare for å opprette utviklermiljø for ROS 1 og ROS 2, og bruk av skriptene er dokumentert på Azure DevOps wiki-en for Bachelorprosjekt USN 2023 etter gjennomføring av test med et gruppemedlem med lite erfaring med bruken av Linux operativsystemer. Skriptene er vedlagt som A30, A29, A31 og A32. Skriptet installerer også avhengigheter essensielt for å bygge prosjektet, blant annet under utvikling og implementasjon av GUI dukket et behov for pakkene til ros-foxy-qt-*.

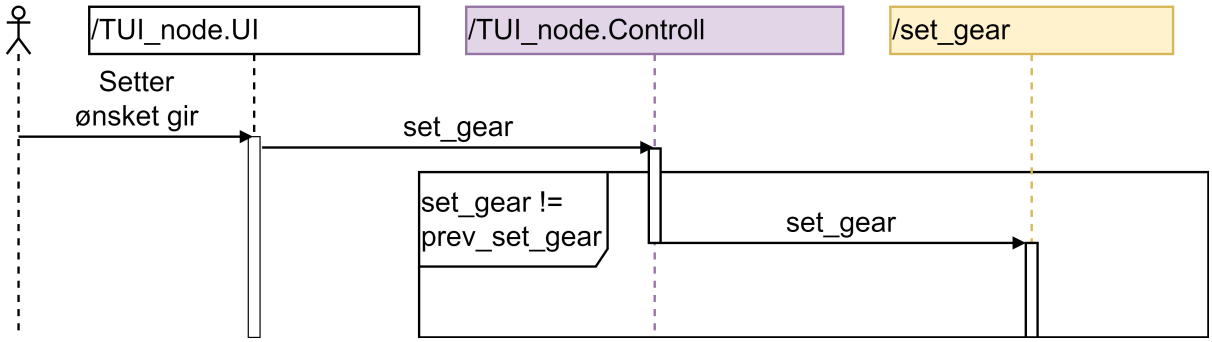
7.4.11.3 Test av skript

Flere tester ble gjennomført på virtuelle maskiner av typen Ubuntu 20.04 LTS under utvikling av skriptene, samt etter skriptene var ferdigstilte. Som nevnt i kapittelet over ble tester utført med en uerfaren Linux bruker - Denne testen avdekket punkter som

måtte tydeliggjøres for fremtidige brukere. En beskrivelse av installasjonsprosedyren ble revidert etter test i form av utførelse av testprosedyre for test T_SW6, se side 11 og 12 i Appendiks A33. Testen var vellykket, skriptet installerte nødvendige pakker og gjorde det mulig for bruker å kompilere ROS pakkene for prosjektet. Krav SW6 er godkjent.

7.4.12 Text-Based User Interface - SW2, SW4 og SW7

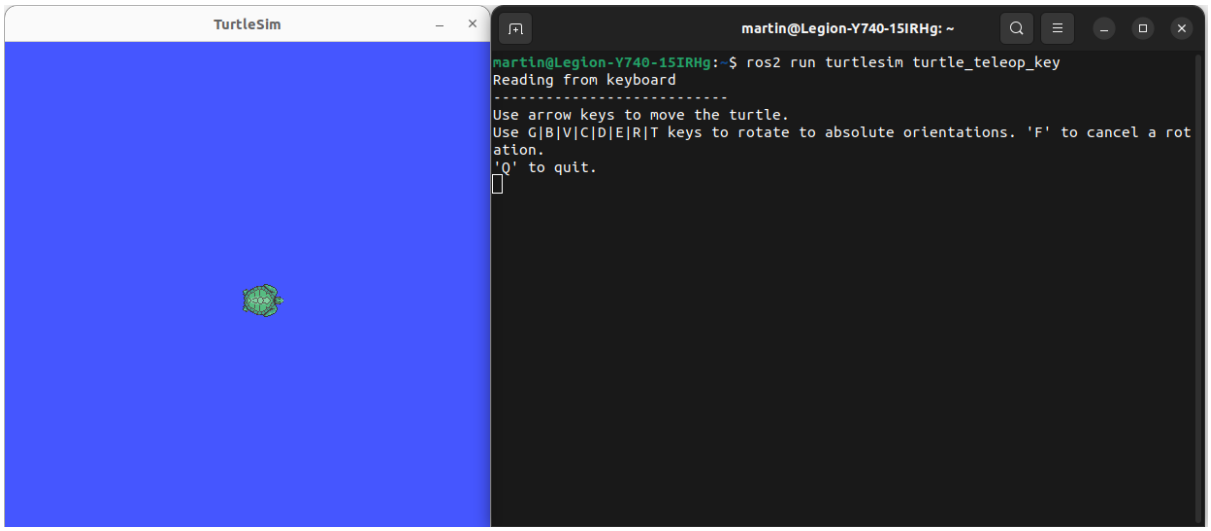
Kort tid etter andre presentasjon oppstod et behov for effektiv testing av girsystemet ettersom mikrokontroller for girskift ikke konsekvent klarte å gjennomføre girskift. Under utviklingsfasen for mikrokontroller for girskift benyttet studentene terminal-kommandoer i form av manuell publisering til topic /set_gear for å utføre girskift, se kodesnutt i kapittel 7.4.9.1. Manuell publisering gikk på bekostning av tiden studentene hadde til rådighet ved test av implementasjon på ATV. Det ble kritisk for testing av systemet å opprette et brukergrensesnitt som effektivt kunne utføre girskift fra en datamaskin, uten å benytte terminalkommandoer. I denne sammenheng ble et enkelt sekvensdiagram utformet for direkte kontroll av girsystemet fra PU, se Figur 7.46.



Figur 7.46: Sekvensdiagram for utvikling av simpel TUI

Gjennom informasjons-innhentingsfasen benyttet dataingeniør-studentene et opplæringsverktøy for å oppnå en bedre forståelse for ROS 2 funksjonalitet. Opplæringsverktøyet beskriver kommunikasjon fra node til node via topics gjennom bruken av en simulator ved navn TurtleSim. Pakken turtlesim benytter en node turtlesim_node som viser en simulator i form av et grafisk brukergrensesnitt. For å opprette kommunikasjon med simulatoren, ble et Tekstbasert brukergrensesnitt benyttet i form av en tastaturleser [37], se Figur 7.47.

Som nevnt i kapittel 7.2.2, 7.2.4 og 7.2.7 gjenkjente dataingeniør-studentene et behov



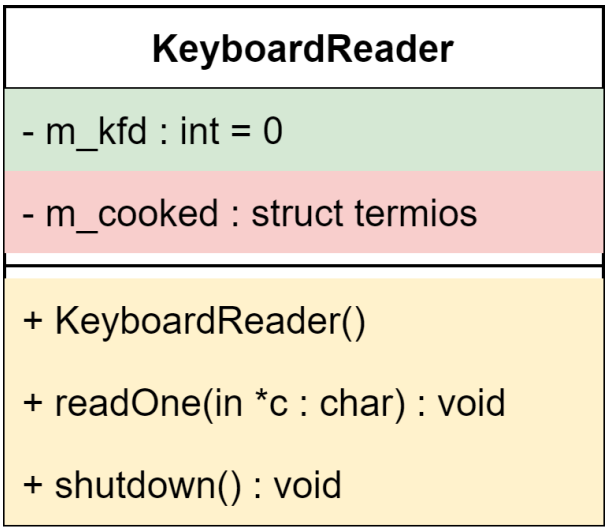
Figur 7.47: ROS 2 pakken TurtleSim

for å implementere et brukergrensesnitt som tillater bruker å kommunisere med PU. Diagrammet for brukergrensesnitt på Figur 7.20 beskriver funksjonaliteten for et tekstbasert brukergrensesnitt navngitt TUI. Etter gjennomgang av opplæringsverktøy gjenkjente dataingeniør-studentene at tastaturleseren turtle_teleop_key kunne benyttes som et grunnlag for å opprette et tekstbasert brukergrensesnitt for å oppfylle kravene SW2, SW4 og SW7.

To klasser; TUI_node beskrevet i kapittel 7.4.12.2 og KeyboardReader beskrevet i kapittel 7.4.12.1 ble implementert basert på turtle_teleop_key noden funnet på GitHub, se https://github.com/ros/ros_tutorials/blob/humble/turtlesim/tutorials/teleop_turtle_key.cpp

7.4.12.1 KeyboardReader klassen

Som nevnt over er TUI basert på turtle_teleop_key noden. For å oppnå synlighet for hva som er nytt, hva som er endret på og hva som er uberørt, er klassediagrammene for KeyboardReader og TUI_node klassene fargekodet med fargene grønn, gul og rød. Grønn fargekode beskriver hva som er nytt, gul beskriver hva som er gjort endringer på, og rød beskriver hva som er uendret.



Figur 7.48: KeyboardReader klassediagram

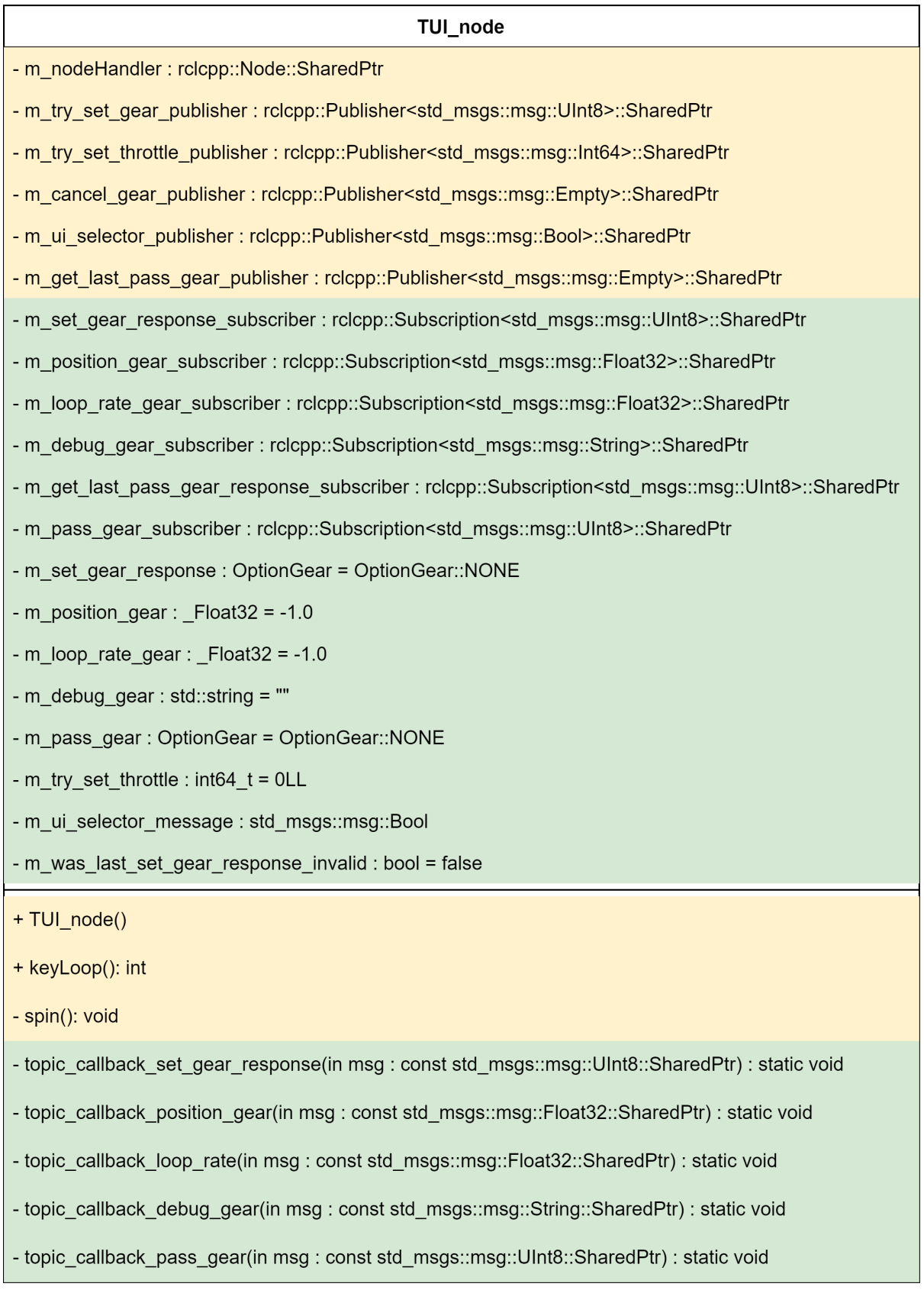
Klassen KeyboardReader håndterer terminalen ved å benytte C biblioteket, termios. Ved å opprette et objekt av typen KeyboardReader vil konstruktøren hente attributtene fra terminalen og lagre dem som en termios struct i medlemsvariabelen m_cooked - Dette er for å overlevere terminalen fra programmet til brukeren etter endt kjøretid. Medlemsvariabelen m_kfd ble opprettet for å forbedre lesbarheten av koden og er en filbeskrivelse for tcsetattr og tcgetattr funksjonene til termios.

For å oppnå ønsket funksjonalitet ved utskrift i terminalen til TUI, var det nødvendig å foreta endringer på kildekoden til konstruktøren KeyboardReader() basert på turtle_teleop_key noden. Ved å fjerne uønskede kontrollverdier i strukturen termios.c_cc på posisjon VTIME og VMIN ble ønskelig funksjonalitet oppnådd, og en tilstand hvor uforutsigbar utskrift ble rettet på.

Funksjonen readOne er endret på ettersom funksjonalitet på Windows systemer ikke var nødvendig, dette førte til at i stedet for å sende tastetrykket som et parameter i funksjonkallet, sendes en peker inn og kontrolleres i funksjonen.

Funskjonen shutdown er implementert som en separat funksjon med samme funksjonalitet til destruktøren fra kildekoden til turtle_teleop_key, ettersom et 'singleton' design mønster ble benyttet for implementasjonen av KeyboardReader klassen i TUI_node klassen. Implementasjonen med singleton design mønsteret førte til et behov for å håndtere nedstenging og overlevering av terminal tilbake til opprinnelig tilstand ved avslutning av kjøretiden til programmet.

7.4.12.2 TUI_node klassen



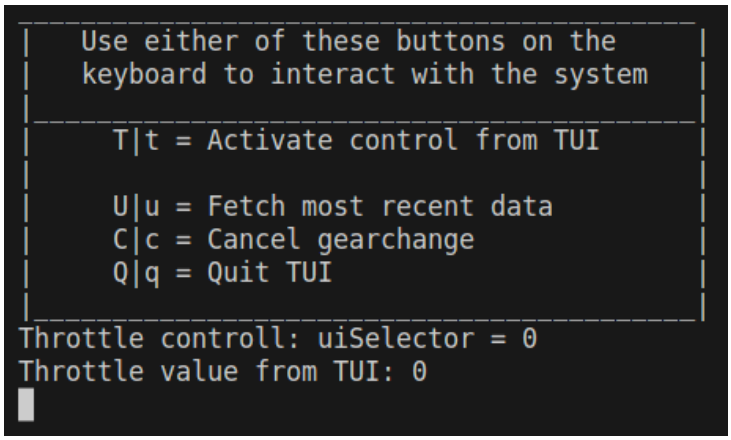
Figur 7.49: TUI_node klassediagram

Klassen TUI_node er basert på klassen TeleopTurtle fra turtle_teleop_key noden. Som nevnt over ble design mønsteret singleton tatt i bruk og betyr i praksis at kun et objekt av en type kan eksistere. Hvis et nytt objekt av klassen TUI_node skapes vil det nye objektet overta det forrige objektet. De 'private' topic_callback funksjonene mottar data ved å abonnere på topics, og når en topic blir publisert til vil verdien lagres i parameteret msg. Singleton ble benyttet da det var ønskelig å hente og bruke verdiene msg.data fra topic_callback funksjonene. Kompilatoren gav feilmelding på at verdiene msg.data ikke kunne benyttes ettersom en midlertidig verdi ikke kunne tilordnes msg.data verdien, singleton design mønsteret utførte ønsket funksjonalitet.

Implementasjonen av konstruktøren i klassen TUI_node fraviker fra implementasjonen av klassen TeleopTurtle for turtle_teleop_key noden i form av singleton design mønsteret. En macro er også definert for en fremtidig implementasjon av Arduino Portenta H7 fremfor Arduino Uno implementasjonen. Ved prosjektinnlevering er ikke Arduino Portenta H7 ferdigstilt, men ved å endre den boolske verdien til true, kompilere og installere pakken, er TUI klar for funksjonalitet med Arduino Portenta H7. Konstruktøren fraviker også med tanke på implementasjon av topic abonnenter.

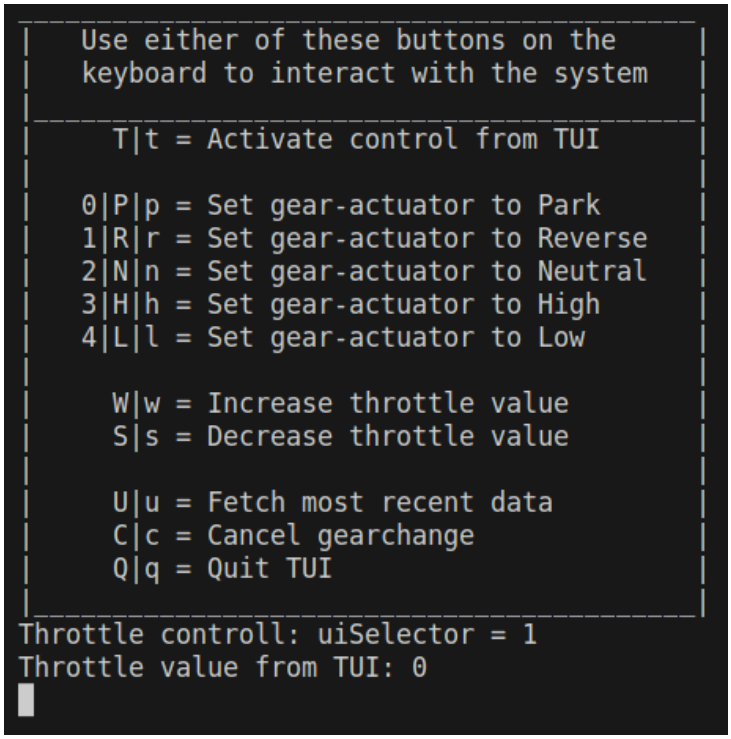
Funksjonen keyLoop har tilsvarende samme funksjonalitet, men presentasjon av data på TUI er nytt. For å feilsøke under testing ble data fra TUI benyttet i form av tilbakemeldinger fra mikrokontroller for girskift. Funksjonen starter opp TUI_noden på en egen tråd og går inn i en while løkke. While løkken flytter markøren i terminalvinduet slik at gammel utskrift forskyves oppover, før den presenterer bruker med instruksjoner for bruken av TUI. Se figur 7.50. Tidlig i utviklingen ble funksjonen system("clear") benyttet fremfor å flytte markøren, men dette førte til at terminalvinduet mistet tidligere innhold. Ved å flytte markøren kan bruker bla gjennom historikken for TUI for å se hvilke verdier som blir presentert gjennom programmets levetid.

For å aktivere kontroll av girsystemet fra TUI, må tasten T trykkes inn. Det er lagt inn funksjonalitet for tastetrykk for stor og liten T, dette betyr at om bruker med uhell aktiverer CAPS LOCK, vil TUI fremdeles gjenkjenne tastetrykket. Først når TUI kontroll blir aktivert blir bruker presentert med flere instruksjoner for bruk av girsystemet. TUI er inaktiv og vil ikke gjøre ha funksjonalitet før dette steget er gjennomført, med unntak av tastetrykkene for T, C og Q. Ved å trykke tasten C, blir en melding av type



Figur 7.50: Oppstart av TUI før kontroll

std_msgs::msg::Empty publisert til topic /cancel_gear som avbryter girskift.

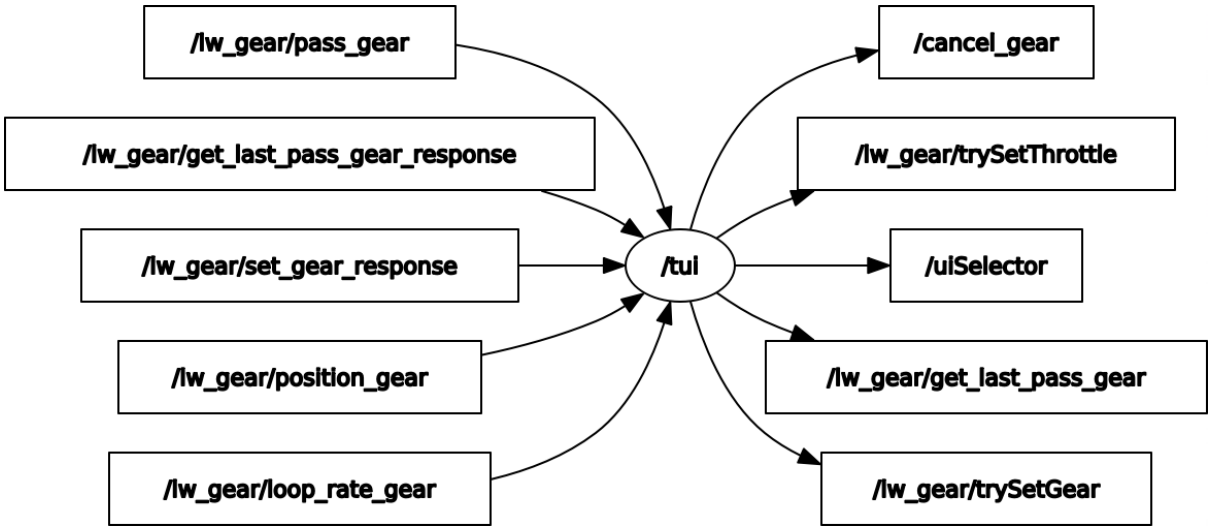


Figur 7.51: Oppstart av TUI etter kontroll

Etter aktivering av TUI blir en boolsk verdi sendt til mikrokontroller for brems- og gasspådrag som tillater head noden beskrevet i kapittel 7.4.9 for å gjennomføre girskift.

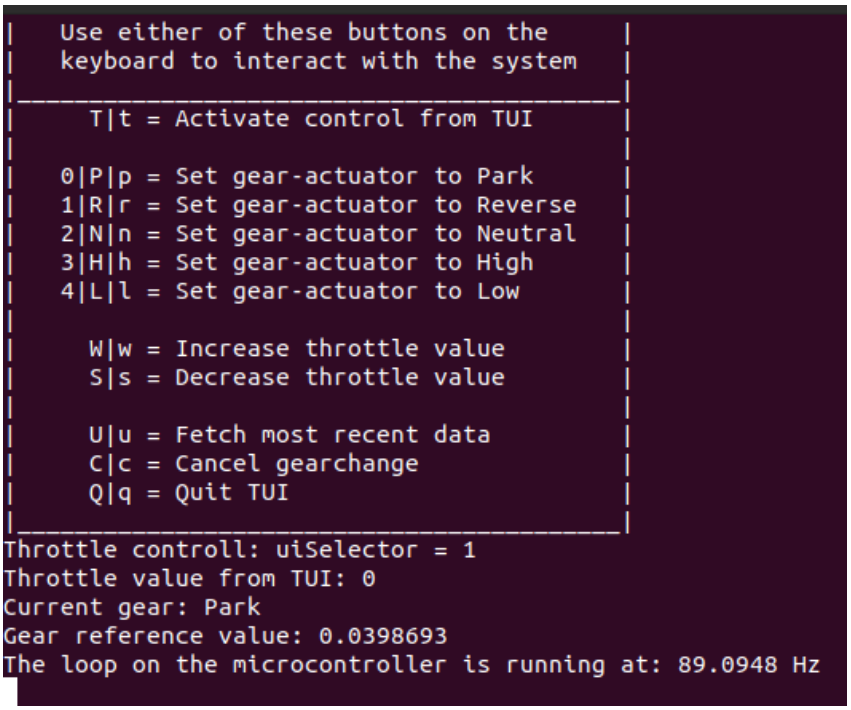
Tastene er kartlagt i TUI ved hjelp av et terminal program showkey. Programmet showkey returnerer en HEX verdi som representerer tastetrykket, og blir dermed hardkodet inn som constexpr char i filen 'compile_time_constants.hpp'. Når terminal som inneholder TUI er aktivert, kan bruker trykke på en av tastene beskrevet i terminal vinduet, se figur 7.51.

TUI abonnerer på topics med pil inn til /tui, og publiserer til topics med pil ut fra /tui, illustrert i figur 7.52. Topic pass_gear inneholder informasjon om hvilket gir mikrokontroller for girskift sist passerte og viser bruker hvilket gir ATV befinner seg i. Topic get_last_pass_gear_response er en topic som kun sender informasjon tilsvarende pass_gear topic, men på forespørsel. Topic pass_gear er implementert på Arduino Uno og publiserer kontinuerlig, men først ved å sende en tom melding av datatype std_msgs::msg::Empty til topic get_last_pass_gear, vil get_last_pass_gear_response respondere. Systemet fungerer tilsvarende som en service, men i stedet for å publisere kun til klient på forespørsel, blir meldingen kringkastet til hele systemet. Topic set_gear_response er et svar fra mikrokontroller for girskift, den forteller hvilken verdi mikrokontroller mottok, og hvorvidt verdien var gyldig. Ved implementasjon av systemet vil TUI aldri publisere direkte til mikrokontroller for girskift, ettersom head noden skal forsørge tilstandene for girskift slik at ikke girskift blir foretatt i tilfeller hvor ATV er i bevegelse. TUI publiserer til /trySetGear topic, og vil aldri publisere tallverdier utenfor de definerte verdiene 0 til 4.



Figur 7.52: Utskrift fra rqt_graph av topics tilknyttet TUI

Figur 7.53 viser TUI under kjøring, tilkoblet girsystemet. Verdien på uiSelector er som nevnt nødvendig for å bruke TUI, uten å påvirke tidligere systemer fra UM600 operatørstasjonen. Det er lagt inn funksjonalitet for å kunne påføre brems- og gasspådrag fra TUI, og som beskrevet senere i kapittel 7.5 for fremtidig utvikling kan utviklere publisere en tallverdi fra 0 til 90 til topic /um600Steering fra TUI, som skal gjøre det mulig å kontrollere sving fra TUI dersom dette er ønskelig.



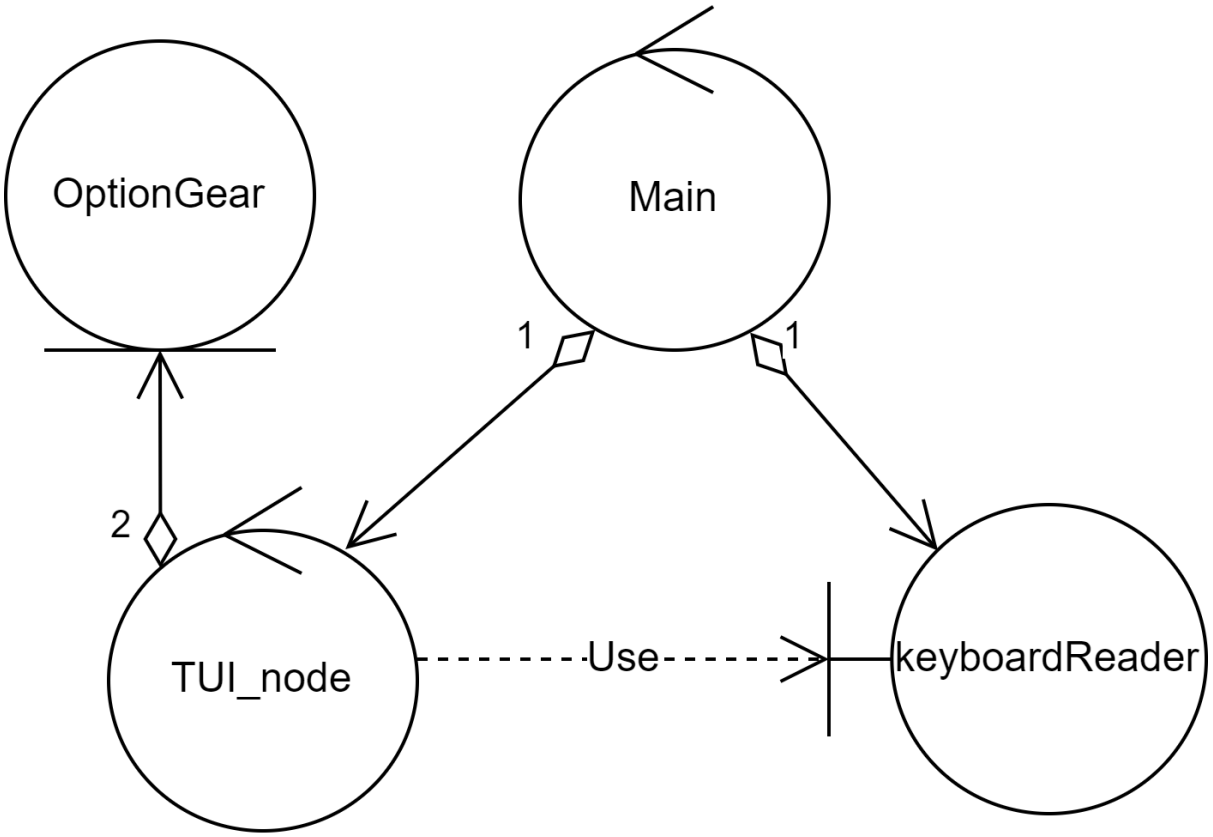
Figur 7.53: Test av TUI

7.4.12.3 Forhold mellom klasser

Programmet starter med å initialisere rclcpp biblioteket for ROS 2. En singleton TUI_node blir deretter opprettet og programmet går inn i funksjonen TUI_node::keyLoop() som skriver ut instruksjoner til bruker som beskrevet i kapitlene over. TUI_node klassen bruker den type-sikre OptionGear klassen som ble utviklet av og implementert ved hjelp av elektroingeniør-studenten. OptionGear sørger for at udefinerte gir ikke publiseres til mikrokontroller for girskift. Mikrokontroller for girskift har derimot innebygd funksjonalitet slik at ved feil gir vil den publisere verdien 255 på topic /lw_gear/set_gear_response, denne funksjonaliteten er nærmere beskrevet i Tabell 7.3

TUI_node klassen bruker også KeyboardReader klassen for å ta over terminalen slik at bruker kan gi inndata til TUI.

Et sekvensdiagram er utformet og dekker funksjonaliteten til TUI programmet ved implementasjon, se Appendiks A45.



Figur 7.54: Klasseforhold

7.4.12.4 Test av TUI

Som tidligere nevnt ble det sett som nødvendig fra utviklernes perspektiv å ha et feilsøkingsverktøy for implementasjonen av girsystemet. TUI ble integrasjonstestet underveis i prosjektløpet, og tester har medført nye iterasjoner og revisjoner av kildekoden for å utvikle nødvendig funksjonalitet for et nyttig brukergrensesnitt. Test av TUI er beskrevet i feltet 'Resultat av test' side 6 for T_SW2, side 8 for T_SW4 og side 13 for T_SW7 i Appendiks 4.4.

Krav SW2 beskriver at bruker skal ha toveiskommunikasjon med PU, og at PU skal ha tilgang på data i form av akselerasjon, brems, girposisjon og GPS. Resultat for test av krav SW2 konkluderer med at PU inneholder relevant data gjennom topics /vectornav/imu, /rcThrottle, /lw_gear/position_gear og /vectornav/gps. TUI presenterer kun data relatert til girsystemet, men ved behov kan nevnte topics abonneres på og presenteres i TUI, GUI eller andre brukergrensesnitt.

Krav SW4 beskriver at PU skal være kjent med absolutt girposisjon. Mikrokontroller for

girskift publiserer til topic /lw_gear/pass_gear for hvert gir den passerer, men i enkelte tilfeller kan girposisjon fravike fra instrumentpanelet til ATV. Mot slutt av prosjektløpet ble et fokus for elektroingeniør-studenten satt på implementasjon av Arduino Portenta H7, som medførte at funksjonaliteten som overvåker passerte gir er mer følsom på implementasjon av Arduino Portenta H7 enn Arduino Uno. Med dette ble det konkludert at implementasjon av Portenta H7 vil føre til et mer nøyaktig system. Topic /lw_gear/position_gear er derimot nøyaktig i form av referanseverdi.

Krav SW7 beskriver blant annet at bruker skal lese klarsignal til giring fra et brukergrensesnitt. Utføring av prosjekt førte til redundans av klarsignal, men ettersom andre punkter i beskrivelsen av kravet er utført og etter samtale med kunde ble krav SW7 godkjent.

Kildekode av TUI er vedlagt, se Appendiks A38.

7.4.13 Graphical User Interface - SW2 og SW7

Prosjektgruppens beslutning om å utvikle et Grafisk Brukergrensesnitt, eller Graphical User Interface (GUI), ble motivert av flere faktorer og behov. Det ville gi enklere testing for utviklere ved å eliminere behovet for terminalkommandoer for å styre ATV gjennom programvaren. Tilbakemeldinger fra eksterne veiledere spilte en viktig rolle i beslutningen om å inkludere et GUI i prosjektet. Deres veiledning og innspill bidro til å sikre at utviklingen av brukergrensesnittet fikk retning og rettelse når det trengte det, og hjalp teamet følge profesjonelle standarder og fokuserte på brukervennlighet og intuitiv systemkontroll. Teamet gjennomførte også en grundig analyse av fordelene og betydningen av et GUI, og identifiserte fordeler som forbedret brukervennlighet, effektivitet og produktivitet. GUIen ville også åpne for potensiell videreutvikling, slik at den kunne brukes fra en tilkoblet maskin over kabel eller WiFi.

7.4.13.1 Planlagt liste av GUI elementer

I den første fasen av utviklingen av GUI var fokuset på det konseptuelle og teoretiske. For å legge et godt og grunndig grunnlag for å bevege seg til videre steg, ble det utarbeidet en liste over elementer og funksjoner som skulle inkluderes, samt hvordan disse skulle presenteres for brukeren. Målet var å utvikle et GUI som tillot interaktivitet og ga brukeren

muligheten til å påvirke ATV gjennom digitale signaler utenfor terminalen. Det var viktig å beskrive brukerens håndtering av GUI-en og hvordan GUI-en skulle kunne motta og behandle brukerens instruksjoner.

I den første fasen av utviklingen av GUI var fokuset på det konseptuelle og teoretiske, så for å legge et godt og grunndig grunnlag for å bevege seg til videre steg, ble det utarbeidet en liste over elementer og funksjoner som skulle inkluderes, samt hvordan de skulle presenteres for brukeren. Teamet ønsket å utvikle et GUI som tillot interaktivitet og ga brukeren muligheten til å påvirke ATV gjennom digitale signaler utenfor terminalen. Det var viktig å beskrive brukerens håndtering av GUIen og hvordan GUIen skulle kunne motta og behandle brukerens instruksjoner.

Et sentralt element i GUIen var girfunksjonaliteten. Det var nødvendig å vise en tydelig oversikt over tilgjengelige gir og gi brukeren mulighet til å velge ønsket gir via GUIen. Dette ble realisert ved å vise det aktive giret som en av de fem bokstavene på en rekke (P, R, N, H eller L), med en markering som tydelig viste hvilket gir som var valgt. Interaktivitet ble implementert slik at brukeren kunne klikke på en av bokstavene for å sende en forespørsel om å skifte gir. All annen funksjonalitet i GUIen ble utviklet for å støtte dette sentrale formålet.

For å gi brukeren tilbakemelding og oversikt over girspakens posisjon og bevegelse, ble det implementert visning av girspaken som en heltallverdi mellom 0 og 255, samt en prosentverdi mellom 0 og 100 med desimalverdi og prosenttegn. Dette ble presentert grafisk ved hjelp av en pil som beveger seg på en farget bar under de fem girknappene.

I tillegg til gir var det essensielt å vise annen relevant informasjon i GUIen, inkludert hastighet og gasspådrag. Hastigheten ble vist numerisk i km/t og visuelt representert ved hjelp av et simulert speedometer. Teamet observerte muligheten for å gi mer eller mindre gass basert på tidligere erfaring med ATV-styring via radiobølgebasert fjernkontroll og studering av tidligere utviklet Arduino-kode. For å representere gasspådrag valgte gruppen å bruke en bar som viser graden av aktivitet.

Statusen for bremsen ble også vurdert som viktig å inkludere i GUIen. Teamet observerte at variabelen for pådrag var den samme som for bremsen, og definerte verdier som indikerte nøytral, full gass og full brems. Det ble diskutert om det var hensiktsmessig å representere

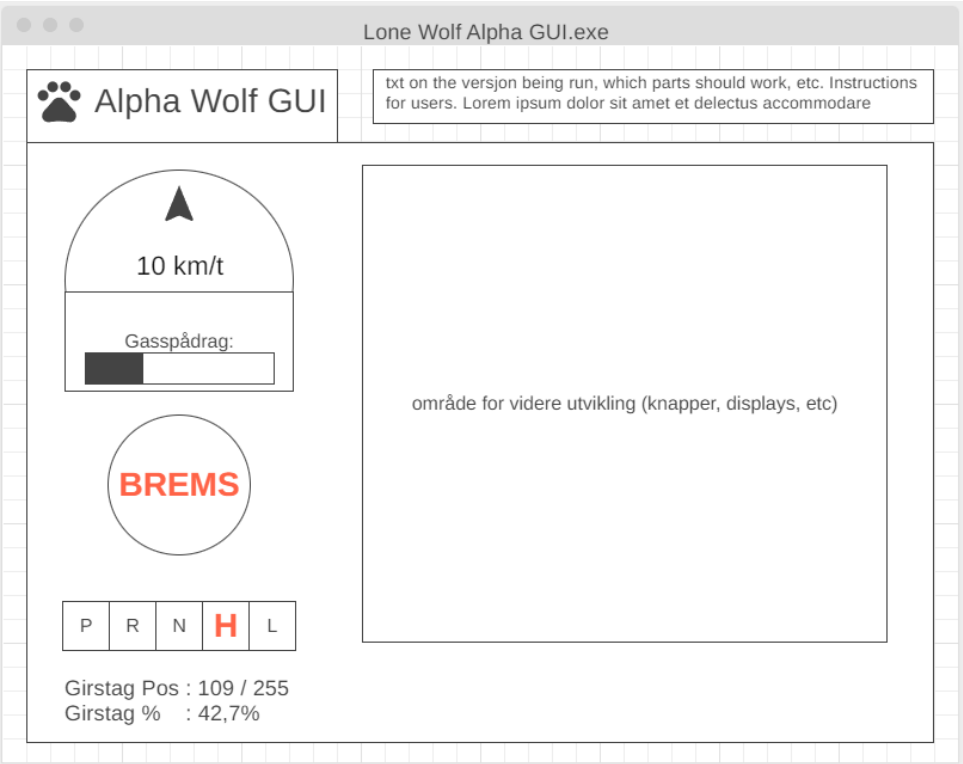
bremsen på en annen måte enn som av eller på. En knapp med teksten 'BREMS' som endrer farge når bremsen er aktivert ble vurdert som passende, da dette visuelt ville ligne et bremselys på en bil. Denne knappen kunne også utvikles for interaktivitet, slik at GUIen kunne sende et signal til ATV-en om å aktivere bremsene og dermed gi en form for nødstopp. Grad av bremsekraft kunne eventuelt vises som en del av gasspådrags-baren, der hele spekteret fra 1200 til 1800 ville bli representert, med fargekoding og tydelig markering for å indikere at verdier over 1500 er pådrag og verdier under 1200 er brems.

Disse elementene utgjorde kjernen av et potensielt grafisk brukergrensesnitt. Andre informative og interaktive elementer kunne vurderes, designes, programmeres og integreres i GUIen etter behov.

7.4.13.2 Grafisk design av GUI

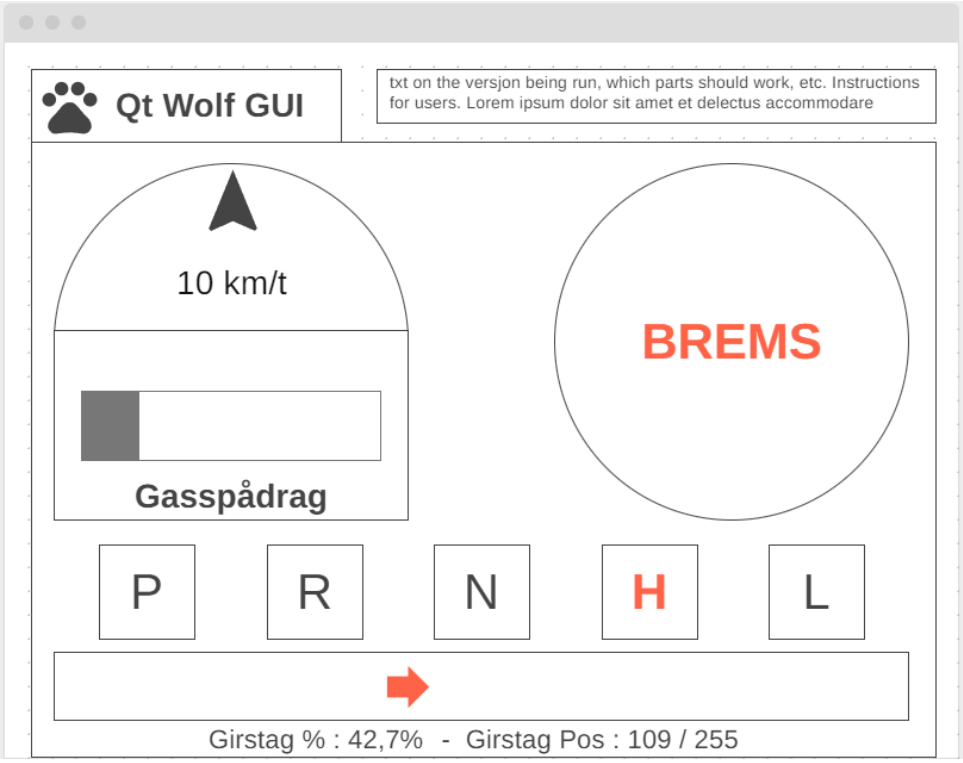
For å konvertere den konseptuelle visjonen av GUI-en til en konkret representasjon, igangsatte teamet en wireframe- og iterativ designprosess. Dette er vanlig praksis å starte med før programmeringen av det grafiske brukergrensesnittet (GUI) påbegynnes utviklingen av illustrasjoner som gir en visuell representasjon av det endelige programmet. Disse illustrasjonene tar form av wireframes og mockups, som er enkle skisser som viser hvordan applikasjonen skal se ut og hva den skal inneholde. Wireframes fungerer som verktøy for å organisere innholdet på en enkel måte og lar designerne eksperimentere med ulike layouter og funksjoner før den faktiske utviklingen starter. Dette resulterer i en mer effektiv designprosess ved å unngå unødvendige endringer senere i prosessen. I tillegg fungerer wireframes som et kommunikasjonsverktøy for å sikre en felles forståelse av applikasjonens visjon blant utviklingsteamet, kunder og interessenter.

Denne prosessen involverte flere iterasjoner hvor wireframe-designet ble utviklet og forbedret basert på tilbakemeldinger fra interessenter og veiledere, samt intern diskusjon i teamet. Gjennom denne iterative prosessen ble ulike designmuligheter utforsket grundig, og det endelige GUI-designet ble nøye tilpasset oppgavegivers forventninger, estetiske hensyn og funksjonelle krav. Den grundige wireframe- og designprosessen la dermed et solid grunnlag for påfølgende GUI-utvikling og implementering. En illustrasjon av det første utkastet av wireframe-designet, som inkluderte alle viktige elementer og muligheter for videreutvikling, finnes i figur A19.



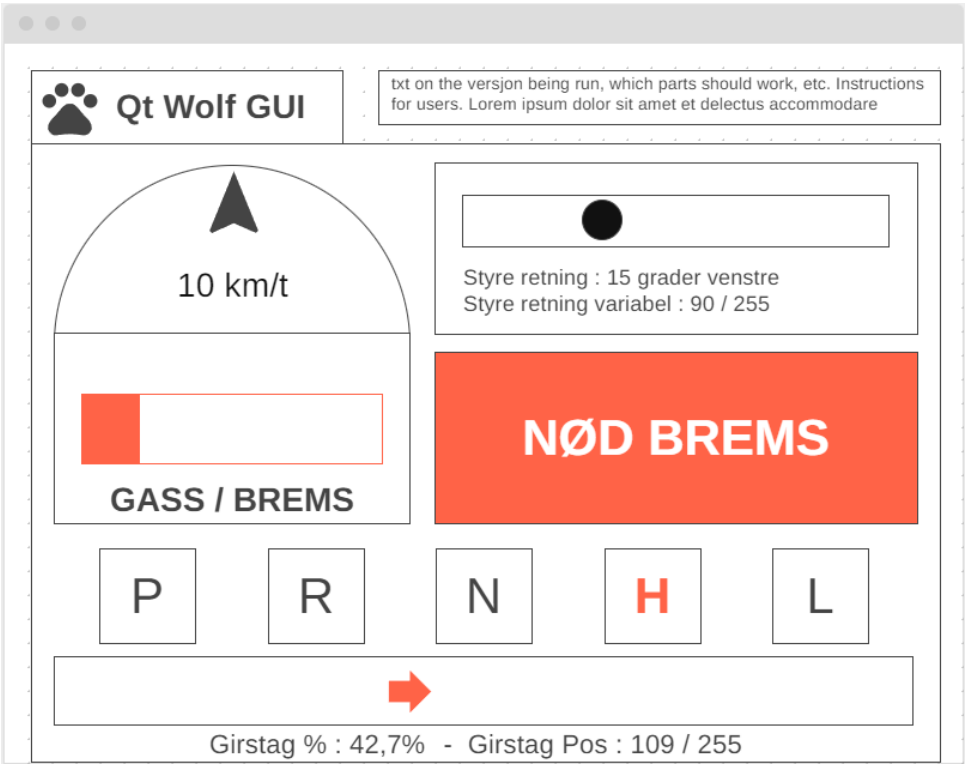
Figur 7.55: Wireframe design av GUI

Underveis i utviklingsprosessen gjennomførte teamet flere iterasjoner av wireframe-designet. I den andre iterasjonen, kjent som 'Simplified', ble denne versjonen presentert for veiledere fra KDA. Veilederne påpekte behovet for tydeligere identifisering av informative og interaktive elementer, samt inkludering av informative elementer for kontroll. En illustrasjon av denne versjonen kan sees i figur A20.



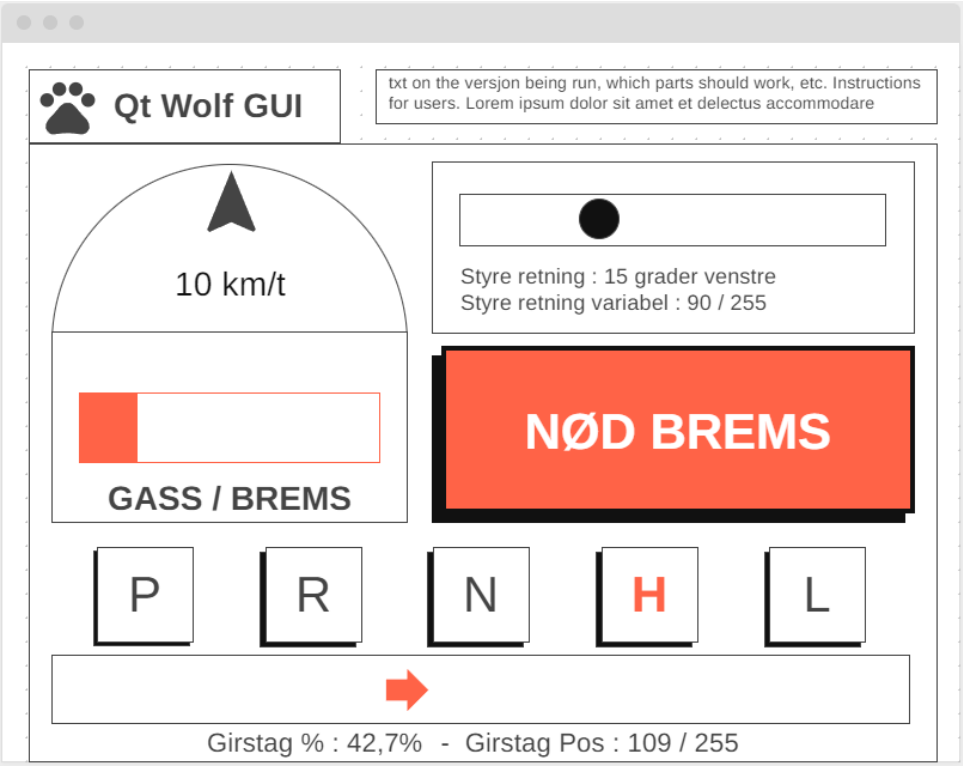
Figur 7.56: Andre iterasjon av wireframe

Deretter, i versjon 3 som er kjent som 'Added Directional Info', ble det gjort betydelige justeringer for å gjøre nødstoppknappen mer fremtredende ved å endre den til en firkantet form som optimerer plassutnyttelsen. Dessverre klarte ikke denne versjonen å tydelig skille mellom informative og interaktive elementer. Se figur A21



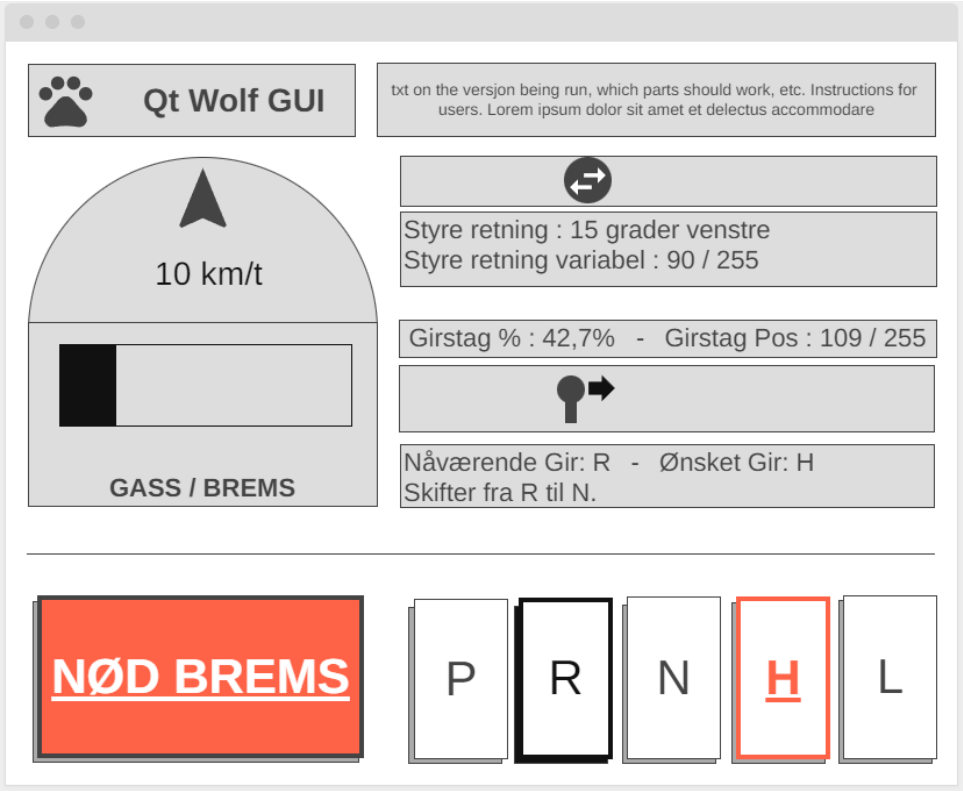
Figur 7.57: tredje iterasjon av wireframe

I versjon 4, kalt 'Button Shadow', ble det gjort mindre justeringer basert på den forrige versjonen. En subtil skyggeeffekt ble lagt til de interaktive knappene for å øke deres visuelle tydelighet. Det er planlagt ytterligere justeringer av denne skyggeeffekten, og dette markerer begynnelsen på implementeringen av denne visuelle løsningen. Likevel gjenstår det en utfordring med at elementene kan oppleves som blandet sammen. Se figur A22



Figur 7.58: Fjære iterasjon av wireframe

Den siste iterasjonen, versjon 5, kjent som 'Split Groups Horizontal', innfører en horisontal oppdeling av brukergrensesnittet (GUI). Informative elementer er plassert over de interaktive elementene. For å øke klarheten i designet, har ikke-interaktive elementer også blitt farget lysgrå i den øvre delen av grensesnittet. Det er imidlertid ikke nødvendig å opprettholde denne fargekodingen i det endelige designet. Mindre justeringer med symboler, farger og visuelle effekter er også utført i denne versjonen for å fremme bedre forståelse av elementenes funksjon og hvilken type informasjon de er ment å formidle. Se figur A23



Figur 7.59: Femte iterasjon av wireframe

7.4.13.3 Valg av rammeverk for GUI-utvikling

Før teamet kunne starte arbeidet med videreutvikling av GUI-en, var det nødvendig for gruppen å ta en beslutning angående valg av rammeverk. En liste over de nødvendige kravene for GUI-en ble presentert: muligheten til å vise ulike datavariabler, sende signaler og utføre handlinger, støtte for interaktive elementer som knapper, slidere og fyllbare barer, kompatibilitet med ROS, ROS2 og Linux-plattformen Ubuntu 20.04, samt evnen til å håndtere komplekse oppgaver med god ytelse og stabilitet. I tillegg var tilgjengelig dokumentasjon og støtte for utvikling og feilsøking viktige faktorer som ble vurdert.

I tillegg til de nødvendige kravene for GUI-en, ble også andre viktige egenskaper ved valg av rammeverk vurdert. Det var ønskelig å ha muligheten til å utvikle og teste wireframe-modeller for å visualisere applikasjonslayouten før selve utviklingen startet. Tilpasningsmuligheter og støtte for ulike plattformer og enheter ble også vurdert som viktige faktorer. Videre var det avgjørende å ha tilgang til ressurser, verktøy og komponenter som kunne forbedre brukeropplevelsen og funksjonaliteten til GUI-en. Valget av et rammeverk som imøtekom disse ønskede egenskapene ville sikre at GUI-en ble

utviklet med høy kvalitet og effektivitet, samt være skreddersydd etter prosjektets behov og krav.

Etter nøye vurdering av alternativene, bestemte gruppen seg for å utforske ulike rammeverk som oppfylte alle de nødvendige kravene og inkluderte flere av de ønskelige bonusegenskapene. Under denne prosessen ble følgende alternative rammeverk undersøkt:

WxWidgets: Et tverrplattform-applikasjonsrammeverk som gir et bredt spekter av verktøy og komponenter for utvikling av desktopapplikasjoner med moderne brukergrensesnitt. Det støtter flere programmeringsspråk, inkludert C++, Python og Ruby, og muliggjør utvikling av applikasjoner som kan kjøre på ulike plattformer som Windows, macOS og Linux. WxWidgets har et aktivt utviklerfellesskap og tilbyr omfattende dokumentasjon og støtte.

Electron: Et populært rammeverk for utvikling av desktopapplikasjoner ved bruk av webteknologier som HTML, CSS og JavaScript. Rammeverket er basert på Chromium-renderingsmotoren og Node.js, og muliggjør utvikling av kraftige applikasjoner som kan kjøre på tvers av ulike operativsystemer som Windows, macOS og Linux. Electron er kjent for sin enkle utviklingsprosess og har et omfattende økosystem av tredjepartsbiblioteker og verktøy.

GTK: Et open-source GUI-rammeverk som brukes til å utvikle desktopapplikasjoner for Linux-systemer. Rammeverket er skrevet i C og tilbyr et sett med verktøy og komponenter for bygging av grafiske brukergrensesnitt som er integrert med GNOME Desktop Environment. GTK er anerkjent for sin enkelhet, fleksibilitet og tilpasningsmuligheter, og det støtter flere programmeringsspråk som C, C++, Python og Rust.

JavaFX: Et Java-basert GUI-rammeverk som tilbyr en omfattende samling av verktøy og komponenter for utvikling av desktopapplikasjoner med rike brukergrensesnitt. JavaFX er en del av Java Development Kit (JDK) og støtter flere plattformer som Windows, macOS og Linux. Rammeverket har avanserte funksjoner som støtte for 2D- og 3D-grafikk, animasjon, multimedia og databinding. Det har også en solid integrasjon med Java-programmeringsspråket og tilbyr omfattende dokumentasjon og ressurser.

.NET: Microsofts plattform for utvikling av desktop- og mobilapplikasjoner med rike brukergrensesnitt. .NET støtter flere programmeringsspråk som C#, Visual Basic og F#,

og gir muligheten til å bygge applikasjoner for Windows-plattformen. Plattformen tilbyr et omfattende sett med verktøy og komponenter, inkludert Windows Presentation Foundation (WPF), som muliggjør utvikling av moderne og attraktive brukergrensesnitt. .NET har også en sterk integrasjon med andre Microsoft-teknologier og et aktivt utviklerfellesskap.

eProsima Fast DDS: En open-source implementasjon av Data Distribution Service (DDS)-standarden, en protokoll for sanntids- og skalerbar datakommunikasjon. Rammeverket er kjent for sin høye ytelse og pålitelighet, og det er spesielt egnet for utvikling av sanntidssystemer med strenge krav til timing og pålitelighet, som industriell automasjon, romfart og robotikk.

Etter å ha utforsket disse rammeverkene nøye og vurdert deres egnethet i forhold til prosjektets behov og krav, ble det besluttet at Qt var det foretrukne rammeverket. Qt oppfylte alle nødvendige krav, inkludert støtte for visning av variabler, signalbehandling og interaktive elementer. Qt hadde god omtale blant rådgivere og andre utviklere, og var blitt tatt i bruk ved tidligere prosjekter rundt Lone Wolf ATV-en.

7.4.13.4 Qt Designer utvikling av GUIs utseende

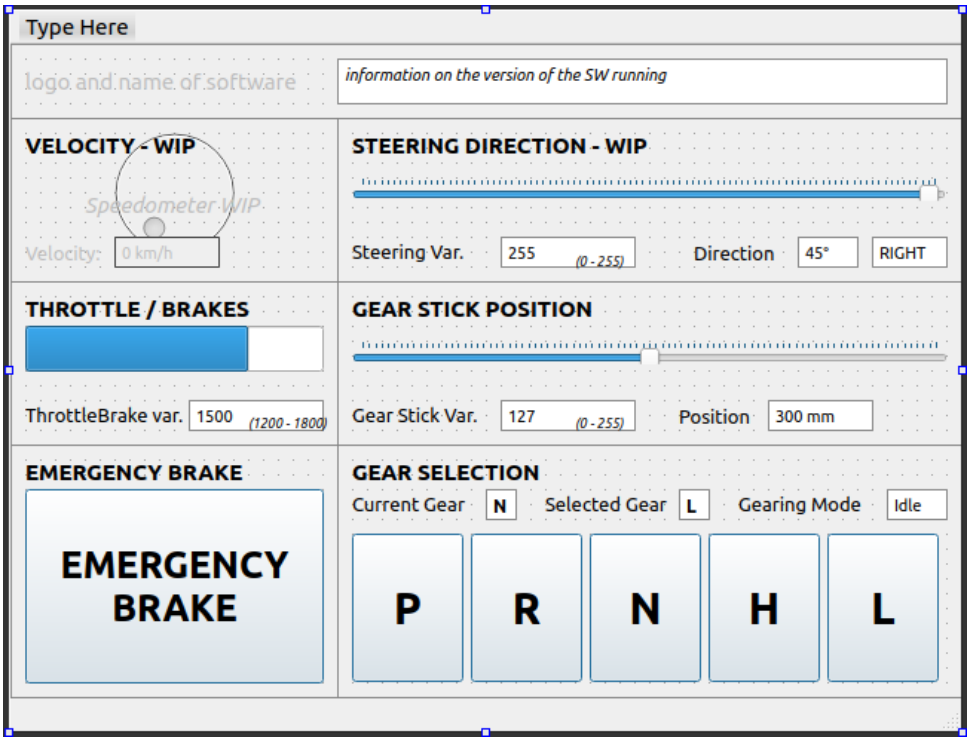
'Qt Designer' ble brukt til å designe utseendet til GUI-en. Programvaren tilbød et intuitivt brukergrensesnitt og et bredt spekter av verktøy for enkel tilpasning av GUI-en. Dra-og-slipp-funksjonaliteten gjorde det enkelt å legge til og plassere widgeter på det visuelle lerretet.

Widgetene kunne tilpasses ved å endre størrelse, posisjon og andre visuelle egenskaper. Bruk av frames gjorde organisering og gruppering av flere widgeter lettere, og resulterte i komplekse og hierarkiske grensesnitt som møtte applikasjonens behov. Egenskapsinspektøren i Qt Designer ga muligheten til å endre widgetenes egenskaper og oppførsel, inkludert tekst, størrelse, interaksjon og tilknyttede handlinger.

Designet var basert på den siste wireframe-modellen, som vist i figur A23. Noen elementer fra wireframen ble nedprioritert til fordel for grunnleggende funksjonalitet på grunn av utfordringer med implementeringen i Qt Designer.

Etter at utseendet på GUI-en i Qt Designer var ferdigstilt for første iterasjon, se figur A24 - teamet eksporterte deretter designet til Qt Creator. Funksjonalitet ble generert til C++

kode av samme navn som .ui filen, GUI_Mainwindo.h og GUI_MainWindow.cpp, og hver widget i designet med ønsket interaktivitet fikk knyttet seg til tilhørende slot-funksjoner i denne koden.



Figur 7.60: Qt Widgets 0.1 Design

Disse funksjonene mottok signaler fra knappene i GUI-vinduet, basert på innebygd Qt logikk som koblet sammen signaler og slotter uten at dette ble gjort av utviklerne på teamet. Disse funksjonene ble utløst hver gang en knapp ble trykket, og en enkel lokal logikk ble implementert inne i gui_mainwindow-klassen: når en knapp ble trykket, endret et tekstfelt i GUI-vinduet sin verdi. A25 Gjennom denne prosessen oppnådde teamet lokal funksjonalitet der GUI-en kunne bygges og kjøres, og viste hvordan det skulle se ut når var fullstendig implementert og samarbeidet med ROS. Fra utsiden skapte denne funksjonen en illusjon av tilknytning til et underliggende system, men det gjensto et trinn for å sende data til og fra ROS-nodene.

```
class GUI_MainWindow : public QMainWindow
{
    Q_OBJECT

    signals:
        void gearButtonClicked(uint8_t newGearValue);

    public slots:
        void on_Button_Gear_0_Park_clicked();
        void on_Button_Gear_1_Reverse_clicked();
        void on_Button_Gear_2_Neutral_clicked();
        void on_Button_Gear_3_High_clicked();
        void on_Button_Gear_4_Low_clicked();
        //void testSlot(uint8_t value);
```

```
GUI_MainWindow& GUI_MainWindow::getWindowInstance()
{
    if (gui_window_singleton == nullptr)
    {
        gui_window_singleton = new GUI_MainWindow();
    }
    return *gui_window_singleton;
}

void GUI_MainWindow::on_Button_Gear_0_Park_clicked()
{
    ui->Info_Text_SelectedGear->setPlainText("P");
    qDebug() << "Emitting signal 0";
    emit gearButtonClicked(0);
}
```

Figur 7.61: Qt generated button slot funksjoner

7.4.13.5 Utfordringer med integrasjon av ROS i Qt prosjekt

Implementeringen av det grafiske brukergrensesnittet støtte på flere utfordringer, spesielt når det kom til kompilering og strukturering av ulike klasser og filer etter introduksjonen av ROS i Qt-prosjektet. Det krevde betydelig tid og innsats å oppnå en optimal struktur og organisering av klassene for å sikre en logisk og hensiktsmessig sammenkobling. Selv om det fantes informasjon tilgjengelig om kommunikasjon og samarbeid mellom Qt og ROS, var det en betydelig utfordring å finne relevant informasjon spesifikt for ROS2 Foxy-koblingen med Qt 5.15, samt bruken av Qt Widgets og CMake i stedet for Qt Quick og qmake. Mange av disse kildene og ressursene var utdaterte, og det var usikkert om de ville fungere for nyere versjoner. I slike tilfeller måtte teamet vurdere om det var hensiktsmessig å prøve dem ut eller lete etter alternative løsninger. Dette førte til flere utfordringer enn forventet når det gjaldt å oppnå et godt samarbeid mellom disse komponentene.

Dette inkluderte også håndtering av versjonskompatibilitet mellom versjoner av Qt og ROS, samt konfigurering og sikring av riktige biblioteker og avhengigheter. CMakeLists.txt måtte tilpasses med flere forskjellige versjoner, ettersom ingen av de andre CMake-filene var tilstrekkelige på grunn av den ekstra kompleksiteten som Qt-komponentene introduserte. Det ble også klart at det var nødvendig å bytte IDE til Visual Studio Code, da det viste seg at en av de mest utfordrende feilene forsvant helt når testing ble utført på denne plattformen, error beskjeden kan sees her: A26. I tillegg foretrakk teamet å bygge gjennom colcon build fremfor Qt Creators byggeprosesser.

Figur 7.62: Qt Build error: Findament_cmake

For å effektivt håndtere kommunikasjonen mellom GUI-en og andre ROS-noder ble en dedikert klasse, GUI_Node, introdusert. Denne klassen bygde videre på eksisterende kode fra tui.hpp-filen, som allerede hadde etablert abonnementer og publiseringer til andre ROS-noder via topics. Sammenkoblingen mellom de to klassene var utfordrende å håndtere, og flere versjoner ble forsøkt underveis. Et forsøk var å implementere GUI-en i en enkelt fil eller som en mellommannsklasse, men dette viste seg å være upraktisk på grunn av kompleksitet og problemer med biblioteker og inkluderinger. Derfor ble arbeidet omarbeidet for å følge en mer modulær og strukturert tilnærming med separate filer for ulike funksjonaliteter.

7.4.13.6 GUI Node med Qt og ROS implementasjon

Denne klassen ble konstruert med utgangspunkt i eksisterende kode fra tui.hpp-filen, som allerede hadde etablert abonnementer og publiseringer til andre ROS-noder gjennom topics. Å integrere denne klassen med GUI-koden var en utfordrende oppgave som involverte flere iterasjoner og forsøk. Etter grundig feilsøking ble det oppdaget at det var mulig å konfigurere GUI_Node-klassen som et QObject og bruke slots og signals. Dette var en overraskende løsning, da det tidligere hadde vært problemer med å kombinere ROS-noden med Qt-funksjonaliteten. Integreringen av GUI_Node-klassen med eksisterende kode kreve nøye tilpasninger for å oppnå ønsket funksjonalitet og samhandling med GUI_MainWindow-klassen.

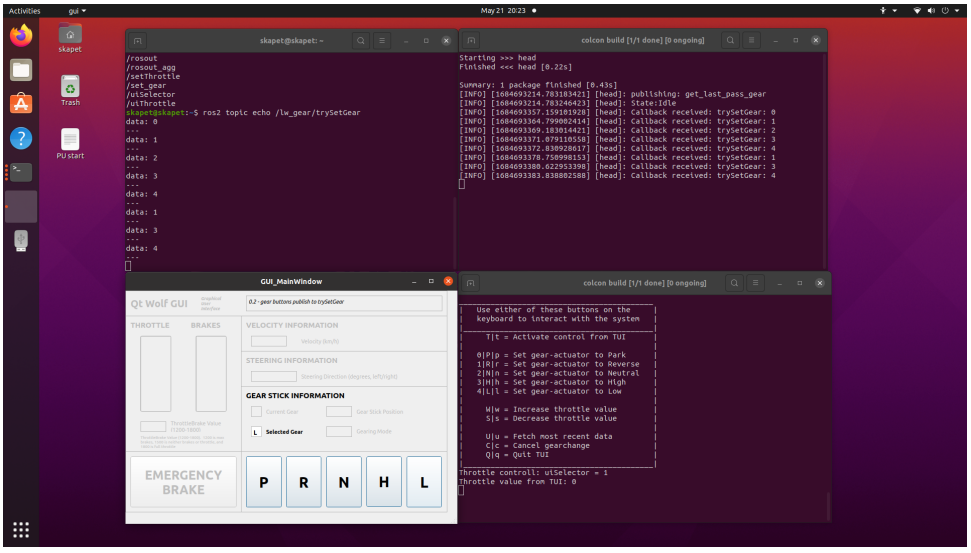
For å sikre effektiv kommunikasjon mellom de to QObject-klassene, var det klart at slots og signals var den mest hensiktsmessige løsningen. Etter grundig utforskning av hvordan dette skulle implementeres, ble det tydelig at connections måtte etableres for å knytte sammen slots og signals på riktig tidspunkt og i riktig rekkefølge under kompileringen.

Det ble funnet ut at det optimale stedet for å etablere disse connections var direkte i main.cpp-filen, og denne tilnærmingen fungerte godt i praksis.

En viktig del av implementeringsprosessen var testing og verifisering av signaloverføringen mellom klassene. Dette innebar å teste at signaler ble korrekt sendt, mottatt og behandlet av tilhørende slots. Ved hjelp av QDebug-funksjonaliteten og QMessageBox kunne det verifiseres at GUI_Node-klassen mottok riktig informasjon fra GUI-vinduet. I tillegg til å oppnå effektiv kommunikasjon mellom GUI-en og andre ROS-noder, var det nødvendig å inkorporere publiseringslogikk for å oppfylle kjernekravene til GUI-en.

Publiseringsfunksjonaliteten fra tui.hpp-filen trengte deretter å bli omformet og tilpasset i GUI_Node-klassen for å oppnå full funksjonalitet. Gjennom grundige tilpasninger og modifikasjoner ble det mulig for GUI-en å publisere relevant data til det spesifiserte trySetGeartopicet. Denne integrasjonen var avgjørende for å oppfylle GUI-ens hovedformål, som var å muliggjøre styring av girskifting gjennom et brukervennlig grafisk grensesnitt.

For å verifisere funksjonaliteten, kunne man enkelt åpne en terminal og sjekke om topicet trySetGear"mottok de samme verdiene som ble sendt ut. Denne testen bekreftet at publiseringen fra GUI-en var vellykket, og dataen ble korrekt overført til de relevante ROS-nodene. Videre ble GUI-en også testet mot resten av ROS-systemet ved å publisere data mens Head-noden fulgte med, som vi kan se i A27. Denne testingen bekreftet at GUI-en var i stand til å samhandle sømløst med resten av ROS-systemet, og oppfylte dermed kravene for integrasjon og funksjonalitet,



Figur 7.63: Qt Widgets 0.1.3 Design

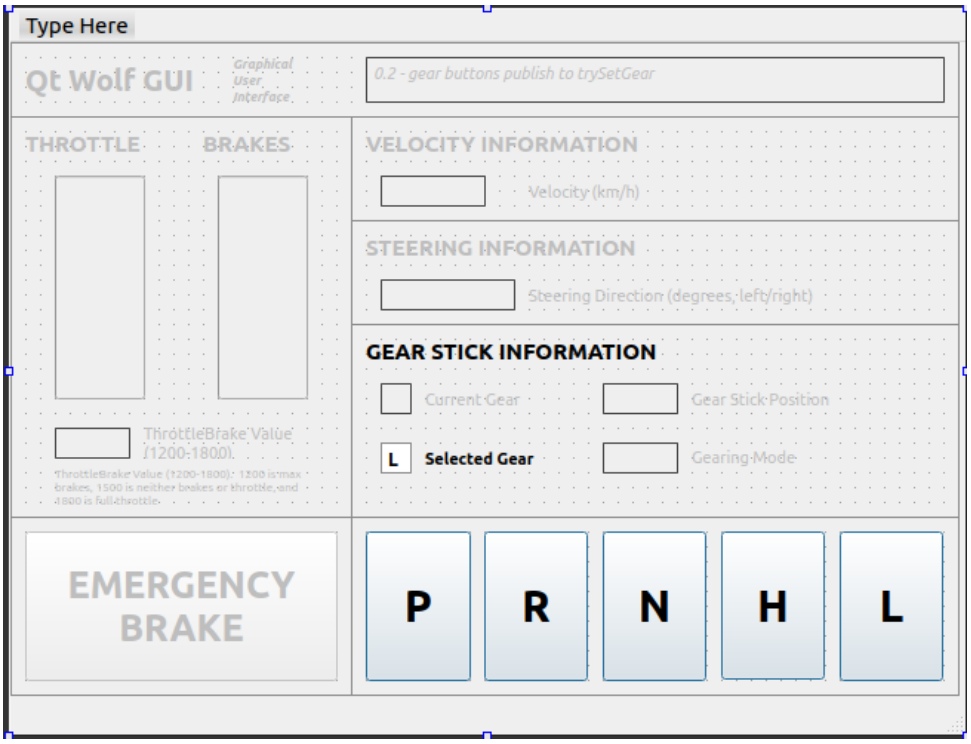
Videre oppdaget vi at for at GUI-en skal kunne kjøre, må en ny maskin ha ROS Foxy-pakkene som støtter Qt-applikasjonsvisning installert. For å håndtere dette, ble installasjonsskriptene oppdatert med en linje som installerer alle ROS Foxy-pakker relatert til Qt (ros-foxy-qt-*). Dette sikrer at alle nødvendige ROS Foxy Qt-pakker blir installert for å kjøre GUI-en på den aktuelle maskinen.

For mer informasjon om GUI koden sin oppbyggning i form av Doxygen rapport, se Appendiks **??**.

7.4.13.7 Avsluttende om GUI

Selv om det gjenstår ytterligere utvikling, har de betydelige utfordringene knyttet til signalhåndtering, connections, subscriptions og publiseringer blitt vellykket adressert. Tilleggssignaler, slots, tilkoblinger, abonnementer og publiseringer kan enkelt opprettes basert på den eksisterende infrastrukturen. gui_mainwindow.ui-filen er blitt endret for å gjenspeile de funksjonelle delene, og de gjenværende komponentene kan gjenaktiveres og kobles til passende signal- eller slotfunksjoner for å samsvare med gui_node-klassen. Imidlertid vil videre utvikling innen disse områdene bli utsatt til senere faser av prosjektet på grunn av tidsbegrensninger.

Designet av GUI-en ble raskt tilpasset for å fokusere på de fungerende elementene og ta hensyn til tilbakemeldinger fra interessenter og veiledere, se A28. Dette tilpasningen gir et godt inntrykk av hvordan GUI-en har potensial til å nå sitt fulle potensial med litt mer innsats og videreutvikling.



Figur 7.64: Qt Widgets 0.1.3 Design

7.5 Fremtidig utvikling

Som siste steg av programvareutviklingen har elektro- og dataingeniør-studentene etablert en god forståelse for systemet og dets funksjonalitet, begrensninger og arkitektur. Studentene har tilegnet seg erfaring ved programvareutviklingen og foreslår potensialer for forbedring og videreutvikling av systemet, spesielt med tanke på girsystemet.

7.5.1 Omdesigne GUI

Tidligere design av GUI inkluderer visse elementer som fortsatt ikke er implementert i Qt Designer. Widgets kan spesifiseres i detalj i dette programmet, men det er også mulig å bygge egendefinerte widgets fra bunnen av. Dette ville være en passende tilnærming for å implementere et eventuelt speedometer-element. Ved å benytte mulighetene i Qt Designer kan det skapes et detaljert og skreddersydd speedometer som passer perfekt inn i det overordnede designet av GUI-en.

7.5.2 Implementere full publisering og subscription i funksjonalitet i GUI

Per nå sender GUI-en bare signaler fra Gear-knappene som publiseres. Videre vil funksjonaliteten utvides for å inkludere nødbrems-knappen, slik at den også fungerer som forventet. I tillegg bør det implementeres subscriptions av informasjon fra ROS-nodene, slik at GUI-en kan oppdateres og vise data som blir hentet fra mikrokontrollerne som er koblet til ATV-en. Dette vil gi en mer omfattende og dynamisk brukeropplevelse i GUI-en.

7.5.3 VectorNAV integrasjon i gir-logikken

For øyeblikket mottar ikke girsystemet noen data fra VectorNAV-topicene, noe som begrenser muligheten til å beregne fart. Tilgang til fartsinformasjon er viktig for intern fartskontroll ved bestemmelse av girskift og for å kunne presentere denne informasjonen til brukeren gjennom GUI-en.

7.5.4 Sette opp Micro-ROS-Agent på PU

Det eneste som gjenstår for å bruke Portenta-implementasjonen av girkontrolleren er å installere Micro-ROS-Agent på PU-maskinen. Dette ble forsøkt, men viste seg å være risikabelt fordi eksisterende versjon av "libc" må nedgraderes for å kunne fullføre installasjonen. Det er usikkert om dette kan føre til problemer i andre deler av systemet, så dette må undersøkes nøye før det tas en endelig beslutning. Det er ytterst viktig at det foretas en backup av systemet før dette steget utføres.

7.5.5 Styre ATV via kontroller på PU

Internt i prosjektgruppen ble det tidlig ytret et ønske om å kunne kjøre ATV fra systemet som har blitt utviklet. Dette innebærer at Head-noden må ta hensyn til instruksjoner sendt fra egnet grensesnitt med tanke på gassrespons og sving av kjøretøyet, i tillegg til muligheter for girskift. Gjennom utviklingsfasen for prosjektet ble manuell kontroll av brems- og gasspådrag implementert for TUI, men grunnet tidsbegrensninger ble ikke kontroll av sving implementert.

En vurdering for hvorvidt en Playstation kontroller kunne implementeres som en selvstendig

node i systemet, gjennom grensesnitt i form av TUI/GUI, eller direkte kommunikasjon med Head-noden. Grunnet dette behandler nåværende revisjon av Head-noden kun instruksjoner angående girskift via TUI, eller direkte via publisering til topic /lw_gear/trySetGear.

Et forsøk på å utføre nøkkeltilordning fra DualShock 4 kontroller ble utført etter installasjon av .deb-pakken Input Remapper. Input Remapper tillater tilordning av taster fra en DualShock 4 kontroller til simulerte tastetrykk fra et tastatur. Som nevnt i kapittel 7.4.12.4 tar TUI inn tastetrykk fra tastaturet til en bruker, og publiserer til topic /lw_gear/trySetGear. En test ble utført hvor taster på DualShock 4 kontrolleren ble tilordnet til tastene 0 til 4, og ved å aktivere TUI vinduet var det mulig å styre girskifting fra kontrolleren.

For fremtidig utvikling kan tilsvarende funksjonalitet utvikles som leser fra en kontroller og publiserer verdier mellom 0 til 4 til topic /lw_gear/trySetGear for å styre girskift, verdier mellom -100 til 100 til topic /lw_gear/trySetThrottle for å styre brems- og gasspådrag i tillegg til å publisere verdier mellom 0 til 90 til topic /um600Steering for å styre sving fra PU. Utvikling av et slikt system vil gjøre det mulig å kontrollere all funksjonalitet til ATV fra en kontroller tilkoblet PU. Bryter på skap som inneholder mikrokontrollerne må være stilt inn på Radio, og operatørstasjon fra UM600 må ikke være aktiv.

7.5.6 Merge av branches for gir-kontroller-kode

Gir-kontrollerkoden består per dags dato av 3 branches. Den første er gjort uten ROS eller ROS2, men har en del ekstra features for analyse og feilsøking av kontrollsløyfen, som kan være nyttig å flette inn i den nyeste versjonen av koden. Den andre branchen har implementert noden gjennom Rosserial, og den siste har noden implementert via rclc_cppb og Micro-ROS. Dersom branchen for Arduino Uno fortsatt skal benyttes, så burde denne slås sammen med Portenta-branchen, men hvor da node-implementasjonen blir valgt ut ifra preprosessor-flagg avhengig av maskinvaremål. Dermed kan man sikre at resten av koden er den samme på begge platformer, og slippe å måtte implementere funksjonalitet dobbelt opp i hver branch.

7.5.7 Tilstander for tillat girskift

Etter samtale med ekstern veileder, Chris Andre Brombach, ble et forslag om å benytte flere parametere som kan sikre tilstander for tillat girskift. Gjennom prosjektet ble det planlagt for slike parametere i form av data fra vectorNAV (VN-300). VN-300 publiserer akselerasjon og fart til topic /vectornav/imu. Ved oppstart av ATV, eller ved signal via TUI/GUI kan det instrueres om å kalibrere akslerasjonsdata fra VN-300, til sammenligning av fartsdata. Ved å samle inn data over X antall sekunder på en gitt frekvens, regne ut snittet av akselerasjonen og under kjøring av systemet trekke fra feilmarginen, kan en mer nøyaktig hastighetsmåling utføres. Ved å ta inn parametere i form av fart og akselerasjon kan systemet sikres ytterligere.

7.5.8 Migrere hele systemet til ROS2 Humble Hawksbill

Ettersom nåværende distribusjon ROS 2 Foxy Fitzroy har EOL[15] Mai 2023 er det sterkt anbefalt å migrere hele systemet over til ROS2 Humble Hawksbill, EOL Mai 2027. For å oppnå dette må de to andre Arduino'ene porteres til Micro-ROS, og en rekke noder på PU må porteres over til ROS2. Målet er å da unngå å måtte bruke ROS Bridge og ha alle noder under samme system.

For å utføre en oppgradering til ROS 2 Humble Hawksbill, eller andre distribusjoner må det gjennomføres tester for å avdekke hvorvidt Gazebo 11 støttes på Ubuntu 22.04 LTS, alternativt om ROS 2 Humble Hawksbill kan installeres på Ubuntu 20.04 LTS.

7.5.9 Sammenslåing av ROS 2 pakker for /lw_gear

Under utvikling av systemet var det ønskelig fra elektro- og dataingeniørstudentene å slå sammen nodene Head, TUI og GUI under en pakke med navn /lw_gear. Dette ble det ikke tid til før innleveringsfrist av prosjektet, men vil gjøre det ryddigere og mer oversiktlig ved fremtidig utvikling. For fremtidig utvikling foreslås det også å påbegynne prosjektet med en sammenslått pakke, slik at etter pakken er kompilert og installert vil alle nodene i pakken kunne kjøres ved kommandoen ros2 run pakke_navn node_navn

[15] End-of-life

8 Nettside

I forbindelse med markedsføringen av bachelorprosjektet ble gruppen tildelt en oppgave om å utvikle en dedikert prosjektnettside. Hovedformålet med nettsiden var å etablere en offentlig tilgjengelig plattform hvor all relevant informasjon om prosjektets fremgang kunne gjøres tilgjengelig for sensorer, oppdragsgivere og andre bachelorgrupper. Dette ville gi oss muligheten til å presentere oss på en profesjonell måte for potensielle arbeidsgivere.

Nettsiden er tilgjengelig på følgende adresse: https://itfag.usn.no/grupper/D03-23/.

8.1 Webdesign prosessen

Det er blitt gjennomført en omfattende webutviklingsprosess som bestod av flere iterasjoner for å sikre en funksjonell og intuitiv nettside. I prosessens tidlige stadier, prioriterte gruppen direkte koding av designet i stedet for å utarbeide skisser for å raskt kunne visualisere designideene i praksis. Samtidig ble det utført en grundig analyse av andre nettsteder, både i forhold til design og funksjonalitet, for å dra nytte av vellykkede og feilaktige elementer på disse nettstedene. Ved hjelp av utviklerverktøy i nettleseren ble det hvordan andre nettsteder var kodet, og brukte denne innsikten til å utvikle kode som leverte lignende resultater.

Ved oppstarten av prosjektet oppsto en del utfordringer knyttet til overføringen av filer til den tiltenkte serveren for nettsiden. Det ble opplevd problemer med å oppnå adgang til serveren, og overføringen tok lengre tid enn forventet som et resultat. Til tross for disse hindringene underveis, ble nettsiden utviklet til et punkt hvor den kunne legges ut for å presentere bachelorgruppen.

8.1.1 Verktøy

For å realisere nettsiden ble det benyttet følgende materialer og verktøy: FileZilla for å koble til servere via FTP og håndtere filers tillatelser. Brackets for å utvikle HTML- og CSS-filene.

8.1.2 Første iterasjon

Den første utgaven av nettsiden ble konstruert i forkant av den første presentasjonen, og mens den oppfylte enkelte av kravene som var satt, viste den seg å ha flere problemområder som krevde oppmerksomhet. Tittelen som sa at nettsiden var et Work in Progress var det mest åpenbare elementet som måtte bli fikset eller fjernet. Det var en mangel på en nødvendig navigasjonsmeny som ville gjøre det enklere for besøkende å bevege seg rundt på nettstedet og utforske innholdet. Videre var logoene som var plassert både øverst og nederst på siden betydelig større enn ønskelig, og dette skapte visuelle forstyrrelser. Det var også en mangel på forklarende tekst som ville hjelpe besøkende med å forstå hva nettstedet handler om og hva som det sentrale bildet var av, i stedet ble plassholder tekst, Lorem Ipsum", brukt for å gi en foreløpig representasjon av hvordan teksten ville se ut når den endelige versjonen var på plass. I sum, ga denne iterasjonen av nettsiden et inntrykk av at den ikke var klar til å bli presentert, og det sies at det var en feilvurdering å laste opp nettsiden i dette tidlige utviklingsstadiet. For å rette opp disse manglene ble det nødvendig å utføre en revisjon.

8.1.3 Andre iterasjon

Etter grundig arbeid og innsats ble nettsiden revidert, og de fleste identifiserte problemområdene ble adressert. Dessverre var det ikke nok tid til å fullføre implementeringen av navigasjonsmenyen, da tiden satt av til arbeidet gikk ut før JavaScript kode kunne bli implementert slik at knappene skulle fungere som tiltenkt. Som følge av dette ble teksten fjernet fra knappene. Dette gjorde at de ikke var synlige i sin uferdige tilstand og ble etterlatt uten funksjon i denne revisjonen.

Logoene ble endret til en mer passende størrelse, og dette forbedret det estetiske utseendet på nettsiden og reduserte visuelle forstyrrelser. Teksten ble også endret fra Lorem Ipsumtil en klar og forklarende tekst som presenterte innholdet på en mer engasjerende måte, og ga en bedre forståelse av hva nettsiden handlet om.

Tross manglene i navigasjonsmenyen, var nettsiden nå i stand til å oppfylle de fleste av kravene som var satt og ga en klar og tydelig representasjon av innholdet. Nettsiden var nå klar til å bli presentert. Gruppen er klar over manglene i navigasjonsmenyen og planlegger

å adressere dette i en senere iterasjon når det blir tid til å utvikle nettsiden videre.

9 Prosessløp

Kapittelet beskriver gruppens progresjon, hindringer, lærdom, gjennom sprintene i prosjektet.

9.1 Sprinter

Etter Scrum Master ble etablert har Scrum gradvis falt på plass, sprint etter sprint. Underkapitlene beskriver denne prosessen.

9.1.1 Sprint 1, 11. - 27. januar

Sprint Goal: Ikke definert

Dataingeniør-studentene startet semesteret med en intens uke med forelesninger, det var derfor ikke mulig å samle gruppen før i uke 3. Elektro- og maskiningeniør-studentene starter derimot i uke 2. I den første sprinten var fokuset på å gjøre rede for effektivt samarbeid og bestemme dokumentasjonssystemer.

For å oppnå et effektivt samarbeid mellom gruppemedlemmene, ble det opprettet en kontrakt for gruppen og det ble enighet om et system for dokumentasjon. Microsoft OneDrive og Teams ble valgt som dokumentasjonssystem, ettersom det tillater enkel deling av dokumenter. Facebook Messenger ble også brukt som en enkel kommunikasjonsplattform mellom medlemmene.

En fysisk inspeksjon på ATV ble nødvendig. Der ble det kartlagt mulighetene for å feste aktuator til gir staget. Gruppen oppdaget også vanskeligheter med girskift på ATV uten å gynge den. Forsøket ble utført uten motor aktiv.

Etter å undersøke girsystemet, diskuterte gruppen mulige løsninger på problemet. En løsning omhandlet en mekanisk innretning som fysisk beveger på ATV. En annen mulig løsning ville være å programmere motorstyrt bevegelse på ATV når giret henger.

Den første sprinten førte til et godt dokumentasjonssystem ved hjelp av Microsoft OneDrive og Teams. Facebook Messenger var også et nyttig verktøy for enkel og rask kommunikasjon mellom medlemmene. Gruppen identifiserte også potensielle problemer og diskuterte mulige løsninger på problemet med girsystemet.

9.1.2 Sprint 2, 1. - 7. februar

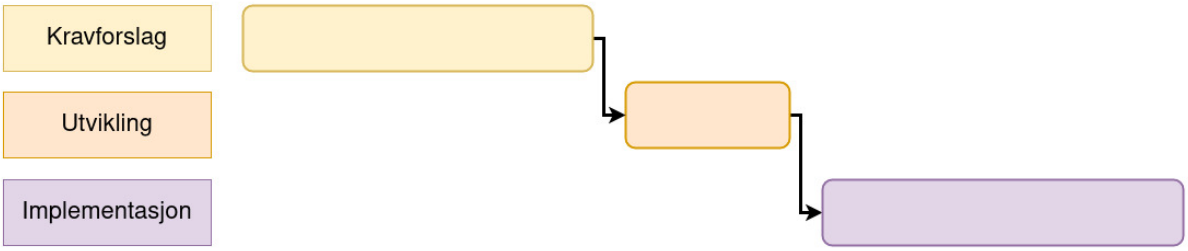
Sprint Goal: Ikke definert

Denne sprinten har hatt fokus på forberedelse til første presentasjon. Arbeidet er dokumentert og samlet til et sammenhengende dokument. Fysisk testing av aktuatorstyring ble gjennomført uten last. Gruppen testet om problemet med girspaken fortsatt gjaldt når motoren var i gang, og det ble konkludert med at gruppen må sette seg dypere inn i girkassemekanismen. Diverse målinger ble også gjort for å finne ut av aktuatorposisjon for forskjellige gir, inkludert vinkelendringer og lengdemål av girspak og giringsstag. Vedlegg, referater, rapporter og annet finnes på en minnepenn datert i perioden.

9.1.3 Sprint 3, 8. - 17. februar

Sprint Goal: Ikke definert

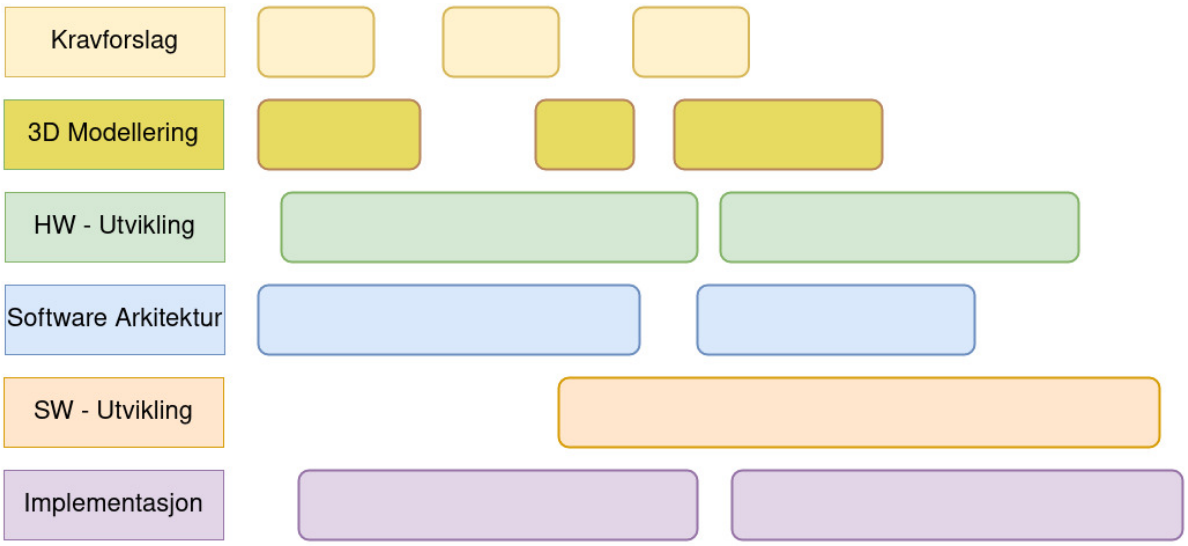
Scrum Master satt seg dypere inn i Scrum, og fikk en oppklaring etter å ha lest The Scrum Guide [2]. Gruppemedlemmene, også kjent som utviklerene har brukt mye tid på å fullføre én oppgave, før neste oppgave startet. noe som indikerte en vannfallsmodell, se figur 9.1. Det var et behov for en Definition of Done, samt å effektivisere arbeidsoppgaver ved hjelp av tidsfrister.



Figur 9.1: Forenklet visning av arbeidsprosess før Scrum

Timeboxing ble introdusert etter utviklerne brukte flere dager på å utforme første utkast av kravforslaget. Det ble bestemt av Scrum Master å ferdigstille første utkast av kravforslaget før endt arbeidsdag, 17.02.2023. Dette tiltaket ble iverksatt for å øke effektiviteten på arbeidet. Etter endt tidsfrist ble kravforslaget sendt for tilbakemeldinger. Kravforslaget med kommentarer ble mottatt påfølgende uke, der det ble satt av maksimalt 2 timer til å revidere kravforslaget før det ble overlevert til kravansvarlig. Se figur 9.2, tabell **??** og **??** for å se forskjellen før og etter Scrum ble implementert med tidsfrister og timeboxing.

Siste dag i Sprint 3 ble Sprint Review og Sprint Retrospektiv gjennomført for første gang. Gjennomføringen var ikke godt organisert, ettersom Scrum Master var ny i rollen med manglende forståelse.



Figur 9.2: Forenklet visning av arbeidsprosess etter Scrum

9.1.4 Sprint 4, 20. februar - 8. Mars

Sprint Goal: Fullføre Girmekanisme Del 1

Målet for sprinten går ut på å bevege linære gir-aktuator fra PU (PC på ATV).

Maskiningeniørene-studentene er avhengig av at elektroingeniør-studenten skal være klar med kontroll av gir-aktuator, slik at den kan monteres og måles for bevegelse gjennom girene.

Midlertidig har maskiningeniør-studentene tatt manuelle mål av gir staget, utført FEM analyse for styrkeberegning, og laget 3D modeller.

Dataingeniør-studentene er avhengig av at elektro-/ og maskiningeniør-studentene klargjør maskinvarekomponentene og monterer dem før programvare kan testes på gir-aktuator.

Midlertidig har dataingeniør-studentene brukt tiden på å planlegge arkitekturen for programvaren, samt satt seg inn i teknologier som skal brukes. Mot slutten av sprint 4 nærmet fristen seg for å fullføre del 1 av prosjektet som betyr at gir-aktuator skal være klar til å drives av ROS2. Ettersom Sprint 4 endte onsdag 08.03.2023, ble det satt en tidsfrist kl. 15 for å ferdigstille implementasjonen på bevegelse av aktuatorstyring fra Arduino.

Senere iterasjoner kan forbedre sluttproduktet, men for at dataingeniør-studentene skal komme i gang vil denne tidsfristen være til hjelp for å øke effektiviteten av første iterasjon for styring av gir-aktuator.

På slutten av sprinten ble det utført en kortvarig Scrum Review, der gruppen gikk over progresjonen mot sprint målet. Konklusjonen var at målet ikke ble nådd, men gir-aktuator ble klar til stabil bevegelse. På slutten av dagen ble det utført en skriftlig Sprint Retrospektiv. For å øke engasjementet fikk gruppemedlemmene i oppgave å skrive et paragraf om hva som var forventet å utføre, hva som ble gjort, hva som hindret dem, hvordan det ble løst og hva læringsutbytte var. Flere utviklere opplevde at arbeidsoppgavene tar lenger tid enn tidligere antatt, og at effektivitet kan forbedres ytterligere.

9.1.5 Sprint 5, 9. - 22. Mars

Sprint Goal: Fullføre dokumentasjon til andre presentasjon

Ettersom eksamensperioden nærmer seg har gruppen bestemt seg for å ha hovedfokus på å fullføre dokumentasjonen til andre presentasjon. Som en del av dette arbeidet har maskin- og elektroingeniør-studentene i gruppen arbeidet på å videreutvikle ATV, spesielt med fokus på å teste kontroll av girskiftsystemet med motor påskrudd og signaler fra seriell kommunikasjon. Maskiningeniørstudentene har også startet å se på den alternative oppgaveutvidelsen som omhandler kraftoverføring til styrestammen.

I denne sprinten har gruppen jobbet nøye med dokumentasjonen for å sikre at alt blir riktig dokumentert og presentert klart og forståelig. Første iterasjon av dokumentasjonen ble skrevet i Microsoft Word, men etter ønske om å øke kvaliteten på dokumentasjonen fra gruppen, ble innholdet overført over til LaTeX. Det har også vært fokus på testing av deler av girskift systemet som har gitt stor fremgang i prosjektet.

Sprint Review gjennomføres i to deler: Del 1 hvor utviklerne bruker en time på en gjennomgang av Sprint Backlog, denne timen brukes også til planlegging for Sprint Review del 2 som gjennomføres og presenteres for arbeidsgiver. Arbeidsgiver, også kjent som stakeholder diskuterer progresjonen, gir tilbakemeldinger og foreslår endringer for videre arbeid. Sprint Review ble kombinert med progresjonsmøte med arbeidsgiver for å redusere antall møter i løpet av sprinten. Ettersom eksamen for dataingeniørstudentene var planlagt

23. mars, og eksamen for elektro- og maskiningeniørstudentene var planlagt 24. mars 2023, avsluttet dagen etter progresjonsmøte, og Sprint Retrospective ble avlyst. Fra et retrospektivt synspunkt kunne dette avverges med bedre planlegging.

9.1.6 Sprint 6, 23. Mars - 12. April

Sprint Goal: Fullføre første versjon av testplan

Sprinten startet opp samme dag som dataingeniør-studentene gjennomførte eksamen i faget Digital Circuits Synthesis som tas ved siden av bachelor prosjektet. Etter enighet i gruppen fortsatte elektroingeniørstudenten med øving til eksamen for Instrumentering og styring og maskiningeniørstudentene fortsatte med eksamensøving til faget Tilvirkningsteknikk. Etter gjennomføring av eksamen satt to dataingeniør-studenter igjen og sørget for utskrift og innlevering av dokumentasjon til andre presentasjon.

Etter andre presentasjon ble Sprint Planning gjennomført. Et usikkerhetsmoment rundt gjennomføring og godkjenning av tester ble tatt opp og gruppen kom til enighet om å gjøre klar første utgave av en testplan som skulle sendes til arbeidsgiver for tilbakemelding.

Maskiningeniør-studentene fikk godkjenning fra arbeidsgiver på utvikling av styresystemet. I denne sammenheng ble krav sendt inn for godkjenning til arbeidsgiver.

Elektroingeniør-studenten utviklet en analog krets basert på utregninger for kraftgrensen til systemet. Kretsen avbryter girskift prosessen i situasjoner hvor girsystemet henger fast.

Dataingeniør-studentene arbeidet på design av grafisk brukergrensesnitt, kommunikasjon over seriell protokoll, testing av ROS 2 funksjonalitet med gir systemet, samt utvikling av programvare arkitektur.

9.1.7 Sprint 7, 13. - 26. April

Maskiningeniør-studentene ble ferdig med bidraget for girskift systemet og har startet på utviklingen av styresystemet. Dette førte til et splittet fokus på målet for sprinten, og to mål ble satt for hvert delprosjekt.

Sprint Goal Styre: Bestille deler til valgt konsept for styremekanisme

Rotordisker ble ferdig designet og overlevert for produksjon, med forventet leveringstid innen 12. Mai. En løsning i form av et stag mellom rotordiskene ble designet.

Maskiningeninør-studentene testet det maksimale dreiemomentet stepper motor leverte, dette var viktig for å sette opp simulasjoner i SolidWorks av styrestamme og rotordisker.

Sprint Goal Gir: Styre girskift fra PU

Et tekstbasert brukergrensesnitt (TUI) ble utviklet basert på eksempelkode fra en node gjennom et veiledningskurs fra utviklerne av ROS 2. TUI ble benyttet under testing av mikrokontroller for girskift. Et installasjonsskript for å opprette utviklermiljøer på virtuelle maskiner for elektro- og dataingeniør-studentene ble ferdigstilt og testet. Arbeid ble foretatt på kildekoden til mikrokontroller for brems- og gasspådrag. Det ble også gjort undersøkelser rundt kompatabilitet for micro-ROS og arduino enheter. Ettersom testing av systemet begynte å nærme seg, ble et oppstartsskript opprettet for å effektivisere gjennomføring av tester på den begrensede tidsperioden gruppen hadde til rådighet for arbeid på ATV. Planlegging av head noden ble påbegynt.

9.1.8 Sprint 8, 27. April - 10. Mai

Sprint Goal: Ferdigstilling av nyutvikling

Gruppen kom til enighet at for å fullføre dokumentasjonen og for å ikke utsette gruppen for 'Feature creep' var det nødvendig å sette en frist til 10. Mai for implementasjon av ny funksjonalitet på systemet. Gruppen var enige i at etter endt sprint skulle det kun utføres finpussing på eksisterende systemer.

Dataingeniør-studentene ferdigstilte det tekstbaserte brukergrensesnittet, det grafiske brukergrensesnittet ble ferdig designet og implementert i Qt, men ROS 2 funksjonalitet gjenstod. Oversiktsdiagrammet for nodene på PU ble ferdigstilt, kildekoden på mikrokontroller for brems- og gasspådrag ble ferdigstilt, oppstartscript for girsystem ble ferdigstilt og programmering av head noden ble påbegynt.

Maskiningeniør-studentene har arbeidet på å støpe deksel til styresystemet og har fullført produsering av alle delene til rotordisken, men forsinkelser fra leverandør førte til hindringer for ferdigstillingen av styresystemet.

Elektroingeniør-studenten arbeidet med implementasjon av micro-ROS for utvikling av kildekode til Arduino Portenta H7. Et bibliotek for sammenkobling av C++ og C biblioteket RCLC ble opprettet med navn rclc_cppb. Kretskort ble designet, bestilt og loddet for Arduino Portenta H7, og arbeidet med til remote procedure calls mellom de to

kjernene til Arduino Portenta H7.

9.1.9 Sprint 9, 11. - 22. Mai

Sprint Goal: Fullføre dokumentasjon til tredje presentasjon
Siste sprint før innlevering har ført til høyt fokus på dokumentasjon.

Maskiningeniør-studentene arbeidet med tilpasning av deksel for styresystemet, utførte test av styremekanisme med påmontert deksel på ATV, utførte NDT testing ved bruk av væske-penetrant metoden og spraylakkerte u-braketter og stag.

Dataingeniør-studentene verifiserte tester av systemet. Grafisk brukergrensesnitt ble ferdigstilt med funksjonalitet for ROS 2.

Elektroingeniør-studenten har arbeidet med å rydde i kildekoden, feilsøkt og integrert systemet ved å legge til en ny topic for å representere passert gir, til informasjon om girskift for TUI og head. En type-sikker representasjon av gir ble implementert med navn gear.hpp og gear_impl.hpp

9.2 Dokumentasjon

Ved den første dokumentasjonsinnleveringen ble det avdekket at en del av prosjektet hadde blitt undervurdert av gruppen og at det var nødvendig å revurdere vår fremgangsmåte for å oppnå et bedre resultat ved neste innlevering. Gruppen uttrykte et ønske om å anvende LaTeXfor dens funksjonalitet og å utarbeide en mer klar plan for arbeidet i forbindelse med neste innlevering.

9.2.1 Latex

Når det gjelder dokumentasjon av tekniske prosjekter, har LaTeX mange fordeler sammenlignet med vanlige tekstbehandlingsprogrammer som Word. For det første gir LaTeX brukeren en større grad av kontroll over layout og formatering, noe som er spesielt viktig når man skal skrive teknisk dokumentasjon med matematiske formler, kodeeksempler eller diagrammer. LaTeX er også spesielt egnet for å håndtere store dokumenter, og muligheten for å generere automatiske kryssreferanser, fotnoter og innholdsfortegnelser gjør det lettere å organisere og navigere i dokumentasjonen. En annen fordel med LaTeX

er at det er en åpen kildekode-plattform, noe som betyr at brukere har tilgang til et bredt spekter av maler, verktøy og plugins som kan tilpasses for å møte spesifikke behov. På grunn av disse fordelene har LaTeX blitt standarden innenfor akademisk og teknisk skriving.

9.2.2 Azure DevOps oppgaver og Dokumentasjon

Gruppen har kommet til en beslutning om at det skal implementeres en endring i gruppens arbeidsprosess som en del av prosjektetløpet kontinuerlige forbedring: Konkret ønskes det å justere gruppens Definition of Done"i Azure DevOps for å sikre fullstendig dokumentasjon av oppgaver. Endringen vil kreve at oppgaver ikke kan bli markert som resolved"i Sprint Backlog før den tekniske delen av oppgaven er fullført og oppgaven er dokumentert i hoveddokumentet på overleaf. Dette vil bidra til å forbedre kvaliteten på arbeidet og redusere stresset rundt dokumentasjonsprosessen frem mot siste presentasjon.

10 Økonomi

10.1 Regnskap

I tabell under ligger oversikt over utgifter gruppen har hatt i forbindelse med utvikling av ny funksjonalitet til ATV. Det har ikke vært et budsjett-tak for prosjektet. Det har istedet blitt foretatt fortløpende vurderinger sammen med veilederne på hva som bør bestilles.

Det skal og nevnes at gruppen har fått kryssfiner, 3D-printede deler, emner for maskinering av bolter, flatstål til u-braketter og materialer til beskyttelsesdeksel fra USN Kongsberg.

Innkjøpsliste Mekanisk

Innkjøpsobjekt:	Antall:	Pris:	Totalt:	Bestillingsdato:
Unbrako bolt M6 x 20mm	2	kr 6,00	kr 12,00	15.03.2023
Låsering Ø19 innvendig	4	kr 14,00	kr 56,00	27.04.2023
Låsering Ø7 utvendig	4	kr 5,30	kr 21,20	27.04.2023
Kulelager 7x19x6 (607 ZZ SKF)	6	kr 70,00	kr 420,00	03.05.2023
M6 Låsemutter	6	kr 4,00	kr 24,00	03.05.2023
M8 Skive	8	kr 1,00	kr 8,00	03.05.2023
Låsesplint hårnål 2x42	1	kr 3,60	kr 3,60	03.05.2023
Stålrør Ø8 x 1,5mm, 1000mm	2	kr 111,00	kr 222,00	09.05.2023
SUM:			**kr 766,80**	

Figur 10.1: Oversikt over innkjøp til de mekaniske systemene

Innkjøpsliste for Data og Elektro

Innkjøpsobjekt:	Antall:	Pris:	Toll:	Totalt:	Bestillingsdato:
Motorkontroller TR-EM-208-H	1	kr 916,00	kr -	kr 916,00	08.02.2023
Proto Shield Rev3 Arduino	1	kr 158,00	kr -	kr 158,00	02.03.2023
Arduino Portenta H7	1	kr 1 243,00	kr -	kr 1 243,00	12.04.2023
Startech ICUSB2321F USB to RS2	1	kr 378,00	kr -	kr 378,00	13.04.2023
Custom kretskort 1	1	kr 380,00	kr 291,00	kr 671,00	19.04.2023
Custom kretskort 2	1	kr 482,74	kr 342,00	kr 824,74	05.04.2023
ELETRA DB9	1	kr 89,00	kr -	kr 89,00	10.05.2023
Eletra USB-USBC Cable 2M	1	kr 199,00	kr -	kr 199,00	13.05.2023
SUM:				**kr 4 478,74**	

Figur 10.2: Oversikt over innkjøp til de elektroniske systemene

11 Ansvarsområder

11.1 Ansvarsområder og titler

Leif Arne Ulvestad Bastesen

- **Dokumentasjonsansvarlig** er ansvarlig for utforming, formatering, sammensetning, utseende og innhold i dokumentasjonen. Ansvarlig for hva som er lagt ved dokumentasjonen og hva som blir utelatt. Ansvarlig for å lære opp de i gruppen som ikke er kjent med LaTeX. Ansvarlig for å lære bruk av Doxygen og å implementere Doxygen generert kode i dokumentasjonen. Sørger for at dokumentasjonen er profesjonelt gjennomført.
- **HR og Sosialt Ansvarlig** har ansvar for håndtering av personal saker via møter og megling, og tar avgjørelser sammen med leder om regler ikke blir fulgt. Dette kombineres med ansvaret for å holde gruppen samlet og motivert ved hjelp av sosiale hendelser, inkludert å koordinere og booke/avtale sosiale arrangement.
- **Nettside ansvarlig** skal sørge for at nettsiden som representerer gruppen og prosjektarbeidet blir designet og programmert.

Vegard Skårdal Brenna

- **Krav ansvarlig** har ansvar for å opprette ett system for å dokumentere krav og konvertere «epics» og «stories» til dokumenterbare krav i nevnte system. Har også hovedansvaret med å kvalitetssikre at kravene er i henhold til kundens ønsker.
- **Økonomi ansvarlig** har ansvar for å hold oversikt over produkter som gruppen låner og ønsker å bestille, samt og sørge for at bestillinger blir utført.
- **Teknisk tegning ansvarlig** har ansvar for at alle tegninger og modeller er korrekt utført og i henhold til standard.

Joachim Jamtvedt Børresen

- **Prosjektleder** har ansvar for at gruppen yter sitt beste på alle områder og at gruppen oppnår enighet i strategi og valg av fokusområder. Dette innebærer å ta avgjørelser når gruppemedlemmer er usikre. Det er leders ansvar å ta tak i

utfordringer og finne en helhetlig løsning sammen med gruppen. Leder har fordelt og fortsetter å veilede andre gruppemedlemmer relatert til sine ansvarsområder. Leder har ansvar for å delegere arbeid som ikke naturlig faller inn under ansvarsområder.

- **Product Owner** har ansvar for å sørge for at Product Backlog til enhver tid er transparent, oppdatert og speiler progresjonen til prosjektet. Oppgaver (Tasks) kan legges inn i Sprint Backlog av andre kun etter avtale med Product owner. Det er essensielt at Product Owner kan kommunisere prosjektet og forstå behovene til alle stakeholders. Sprint Backlog justeres også etter Sprint Retrospektiv og i samarbeid med Scrum Master.
- **Kommunikasjonsansvarlig** sørger for at all kommunikasjon foregår på en oversiktlig og profesjonell måte. Det innebærer opprettelse av maler for referater, rapporter, etc. I hovedsak er det kommunikasjonsansvarlig som tar seg av telefon og elektronisk kommunikasjon ovenfor stakeholders.

Martin Jørgensen

- **Nestleder:** Som nestleder har Martin ansvar for å hjelpe gruppemedlemmer å komme til enighet, eller å ta avgjørelser når diskusjonen ikke konkluderer innen en hensiktsmessig tidsramme. Dette gjøres når gruppeleder ikke er tilstede, når gruppeleder ikke klarer å konkludere diskusjonen, eller om gruppeleder selv diskuterer.
- **Scrum Master:** Som Scrum Master har Martin ansvar for å øke effektiviteten i gruppen, hjelpe gruppen med implementasjonen av Scrum, sørge for at Scrum Events blir følgt etter planen, mm. Disse ansvarsområdene og flere står beskrevet i The Scrum Guide [2, p. 6].
- **Risikoansvarlig:** Som risikoansvarlig har Martin ansvar for å tenke over hva som kan gå galt, for å avverge problemer. Denne risikovurderingen foregår gjennom hele prosjektet, der Martin skal forsøke å forutse problematikk, og gjøre/foreslå tiltak for å redusere konsekvensen eller sannsynligheten for at problematikken kan oppstå.

Sigurd Sæterø Spangelo

- **Git-ansvarlig:** Som Git-ansvarlig har Sigurd ansvar for at Git-repoer blir opprettet

på riktig måte, at alt på Git er gjort på ryddig og systematisk vis, og sette regler for bruk av Git om nødvendig.

- **Standardansvarlig:** Som standardansvarlig har Sigurd ansvar for at løsningene vi bruker er standardisert så godt det lar seg gjøre og så lenge det er hensiktsmessig, og at vi konsekvent følger de standardene vi har valgt å bruke.
- **Kretskort- og kretsdesignansvarlig:** Som kretskort- og kretsdesignansvarlig har Sigurd ansvar for at alle kretskort og kretsdesign kontrolleres før bruk og er utviklet/designet i henhold til våre krav.

Vebjørn Aleksander Østlie

- **Test ansvarlig:** Som test ansvarlig har Vebjørn ansvar for å sette opp et system for å dokumentere tester og sørge for at de verifiserer om kravene er oppnådd på en tilstrekkelig måte.
- **Hardware ansvarlig:** Som hardware ansvarlig har Vebjørn ansvar for at nødvendig hardware blir produsert/bestilt i tide og at de er i tillstrekkelig kvalitet. Det innebærer kvalitetskontroll og verifisering av at delene som skal produseres holder mål. Ansvars området innebærer også kontrollering om at delene er riktig dimensjonert for å tåle belastningene de er utsatt for.
- **Instagram ansvarlig:** Som instagram ansvarlig har Vebjørn ansvar for å legge ut videoer og bilder av prosjektet. Hensikten med dette er å skape oppmerksomhet og interesse rundt prosjektet. Ansvarsområdet innebærer også å sørge for at innholdet som legges ut er trygt og ikke inneholder konfidensiell informasjon.

Figurliste

Tabelliste

Referanser

[1] S. C. et. al., "Git." https://git-scm.com/, 3 2023. [Nettkilde; hentet 22.03.2023].

[2] K. Schwaber and J. Sutherland, "The scrum guide." Scrum.org, November 2020. Accessed: February 17, 2023.

[3] S. Bai, X. Li, Z. Xie, Z. Zhou, and J. Ou, "A Wireless Fatigue Monitoring System Utilizing a Bio-Inspired Tree Ring Data Tracking Technique," Mars 2014. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4003947/figure/f14-sensors-14-04364/.

[4] J. N. Richard G.Budynas, *Shigley's mechanical engineering design eleventh edition in Si units*. McGraw Hill Education, 2021.

[5] Øystein Vollen, *Mekanikk for ingeniører, Statikk og fasthetslære 2. utgave*. Fagbokforlaget, 2010.

[6] "Screw thread design - Fastenal," Mars 2009. https://www.fastenal.com/content/feds/pdf/Article%20-%20Screw%20Threads%20Design.pdf.

[7] LINAK A/S, *Actuator LA14 User manual*, 9 2020. MA-M9-02-487-Q.

[8] LINAK A/S, *Actuator LA14 Data sheet*, 9 2020. MA-M9-02-481-O.

[9] LINAK A/S, *Actuator LA14 Absolute positioning - Analogue feedback Connection diagram*, 1 2021. MA-M9-02-591-B.

[10] LINAK A/S, *TR-EM-288 Positioning Driver Data sheet*.

[11] LINAK A/S, *TECH-system Type-208 For single operation Instructions for installation and use*, 2019. ver. 06.

[12] LINAK A/S, *TR-EM-208 Single Motor Control Unit Data sheet*.

[13] LINAK A/S, *QUICK GUIDE MOTOR CONTROLLER Type TR-EM-208*.

[14] LINAK A/S, *Self-help guide TR-EM-208*.

[15] Handson Technology, *User Guide L298N Dual H-Bridge Motor Driver*.

[16] STMicroelectronics, *L298 Dual Full-Bridge Driver*, 1 2000.

[17] W. Bolton, *Mechatronics: A Multidisciplinary Approach 5th Ed.* Pearson, 1 2013.

[18] Farnell, *Arduino Uno*.

[19] J. Staschulat and P. Garrido, "Ros2/rclc: Ros client library for the c language.," Feb 2015. https://github.com/ros2/rclc.

[20] W. Woodall and D. Thomas, "Ros2/rclcpp: Rclcpp (ros client library for c++)," Jul 2014. https://github.com/ros2/rclcpp.

[21] M. Purvis, M. Ferguson, and P. Bouchier, "A ros client library for small, embedded devices, such as arduino.," May 2011.

[22] P. Garrido, "Micro-ros/micro_ros_arduino at foxy," Oct 2020.

[23] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[24] D. Thomas, W. Woodall, and M. Arguedas, "Ros2/ros1_bridge at foxy," Jun 2015.

[25] "Ros 1 distributions." [Nettkilde; hentet 15.05.2023].

[26] C. Lalancette and D. Thomas, "Ros 2 distributions," Oct 2018.

[27] VectorNAV, *VN-300 DUAL GNSS/INS*, 2020. Rev2.

[28] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing 3rd Ed.* Pearson, 1989.

[29] crayzeewulf, "Libserial," 9 2020. v1.0.0.

[30] Future Technology Devices International Ltd., *FT232R USB UART IC Datasheet*, 5 2020. Version 2.16.

[31] Wiki.ros.org, "Ros distributions," 2023.

[32] Wiki.ros.org, "Arduino ide setup," 2022.

[33] K. Hope, S. Husa, E. Håve, B. Ims, J. Kloumann, S. Mellemseter, A. Moholth, H. Myrling, K. Nilsen, C. Nikolaisen, D. Skauge, and J. Sæter, "Lone Wolf 2022," August 2022.

[34] "Ros 2 foxy."

[35] "Ros 2 humble."

[36] "micro-ros," 2023.

[37] D. Thomas and W. Woodall, "Ros/ros_tutorials: Code used in tutorials found on ros wiki," Feb 2011.

Appendiks

A1 Gruppeavtale_V2.pdf

A2 LoneWolf2022_Kongsberg_Rapport.pdf

A3 Dokumentasjon, seriellprotokoll Mk.1.pdf

A4 LoneWolf2023GirskiftLogikk_doxygen_report.pdf

A5 LoneWolf2023GearingArduino_doxygen_report.pdf

A6 LoneWolf2023GearingArduino_doxygen_report_tidlig_ve

A7 uml_klassediagram_mcu.pdf

A8 lw_gear_doxygen_report.pdf

A9 Rapport Aktuatorkontroll Prototyp.pdf

A10 Sammenligning, Motordriver.pdf

A11 Sikkerhetsrutiner for håndtering av ATV.pdf

A12 Risikodiagram.pdf

A13 Wireframe-Modell_v1.png

A14 Nettside_24.03_del1_v2.png

A15 Nettside_24.03_del2_v2.png

A16 Oppgavetekst - KDA DLS - Lone Wolf.pdf

A17 Text-Based_User_Interface_doxygen_report.pdf

A18 guiDoxygenReport.pdf

A19 GUI_Wireframe_V1_Future_Development_Space.png

A20 GUI_Wireframe_V2_Simplified.png

A21 GUI_Wireframe_V3_Added_Directional_Info.png

A22 GUI_Wireframe_V4__Button_Shadow.png

A23 GUI_Wireframe_V5_Split_Groups_Horizontal_Version

A24 GUI_0.1.png

A25 QtButtonLogic_Signals_Slots.png

A26 QtCreator_Findament_cmake.cmake_error.png

A27 QtTest_Topic_head.png

A28 GUI_0.1.3.png

A29 Kildekode/installasjonsscript/ROS2Foxy_install.sh

A30 Kildekode/installasjonsscript/ROSNoetic_install.sh

A31 Kildekode/installasjonsscript/dependencies_install.sh

A32 Kildekode/installasjonsscript/ROS2Humble_install.sh

A33 Krav- og testspesifikasjon.pdf

A34 LoneWolf2023_Head_node_doxygen_report.pdf

A35 Kildekode/HEAD

A36 Kildekode/Throttle-Arduino-Uendret

A37 Kildekode/Throttle-Arduino-Rev-4

A38 Kildekode/TUI

A39 LoneWolf2023_throttle_arduino_doxygen_report.pdf

A40 Kildekode/oppstartsscript/lw_light_start.sh

A41 Kildekode/oppstartsscript/lw_girsystem_start.sh

A42 2D Tegning - Rotorskive - styrestamme.pdf

A43 SKF 626-2RSH specification.pdf

A44 TUI_UML-SEQUENCE DIAGRAM_V1

A45 TUI_UML-SEQUENCE DIAGRAM_V2

A46 Lone Wolf - Tankekart.png

A47 Gerber_PCB_LoneWolf Connector Board for Gearing-Controller (Arduino Uno).zip

A48 Gerber_PCB_LoneWolf Connector Board for Gearing-Controller (Portenta H7).zip

A49 SCH_LoneWolf Connector Board for Gearing-Controller (Arduino Uno)_2023-05-22.json

A50 SCH_LoneWolf Connector Board for Gearing-Controller (Portenta H7)_2023-05-22.json

A51 um600-tactical-broadband-radio-datasheet.pdf