Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Knut Harald Gabow Au
Tove Ingeborg Løvø
Karl Magnus Sundelin
Tor Berge Sæbjørnsen

Videreutvikling og testing av tobeinet robotprototype

Bacheloroppgave i Elektroingeniør - BIELEKTRO
Veileder: Torleif Anstensrud

Mai 2022

NTNU
Kunnskap for en bedre verden

Knut Harald Gabow Au
Tove Ingeborg Løvø
Karl Magnus Sundelin
Tor Berge Sæbjørnsen

Videreutvikling og testing av tobeinet robotprototype

Bacheloroppgave i Elektroingeniør - BIELEKTRO
Veileder: Torleif Anstensrud
Mai 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU

Kunnskap for en bedre verden

NTNU

Tittelside Bacheloroppgave BIELEKTRO

Oppgavetittel (norsk og engelsk):
Videreutvikling og testing av tobeinet robotprototype
Testing and further development of a biped robotprototype

Forfattere:
Knut Harald Gabow Au
Tove Ingeborg Løvø
Karl Magnus Sundelin
Tor Berge Sæbjørnsen

Prosjektnummer:	E2206
Innleveringsdato:	20.05.2022
Gradering:	[x] åpen [ ] lukket

Studium:	Elektroingeniør - BIELEKTRO
Studieretning:	Automatisering og robotikk og Elektronikk og sensorsystemer
Veileder internt:	Torleif Anstensrud
Institutt:	Institutt for Teknisk Kybernetikk
Oppdragsgiver:	Institutt for Teknisk Kybernetikk
Kontaktperson:	Torleif Anstensrud

Sammendrag (norsk og engelsk):
Det er utført en maskinvareundersøkelse av alle robotens komponenter da tidligere gruppers arbeid pekte på en del utfordringer knyttet til dem. Her ble det testet hvilke komponenter som var fungerende og hvilke som var defekte. Det ble bestilt nye komponenter og de defekte ble byttet ut. Det har vært en større utfordring enn tiltenkt å få de nye komponentene til å fungere. Nøyaktig 3D-modell er utformet slik at framtidige grupper enkelt kan foreta simuleringer. Reguleringsteori ble utformet for videre arbeid med roboten. Det er i stor grad jobbet med oppsett og kommunikasjon mellom komponenter via BeagleBone Black der SSH kommunikasjon har vært brukt for å kommunisere med en datamaskin og BeagleBone Black. Beaglebone Black kjører Ubuntu 20.04 operativsystem og ROS Noetic er installert.

A thorough hardware testing was carried out as previous groups had challenges regarding them. After testing the defective hardware was identified and replaced. Getting the new hardware to work was a lot harder than first anticipated. An accurate 3D-model of the robot is designed so future groups easily can run simulations. Control theory is ready for future work on the robot. Work has primarily been focused on setup and communication between BeagleBone Black and the other electronics. As there is no graphical interface on the BeagleBone Black, SSH communication was used on another computer to communicate with the BeagleBone Black. The BeagleBone Black is running Ubuntu 20.04 and has ROS Noetic installed.

Stikkord norsk:	**Stikkord engelsk:**
3D-modell, BeagleBone Black, ROS, SSH, Ubuntu	3D-model, BeagleBone Black, ROS, SSH, Ubuntu

Forord

Etter en lang, lærerik, og utfordrende prosess er vi nå ferdig og klar til å levere vår bacheloroppgave. Denne oppgaven er det avsluttende kapittelet av en 3-årig bachelorutdanning innen elektroingeniørlinjen, med tverrfaglig kompetanse der halve gruppen har studieretningen *Automatisering og robotikk* og andre halvdelen *Elektronikk og sensorsystemer*.

Grunner som fikk oss til å velge denne oppgaven var at det tverrfaglige aspektet gir oss muligheten til å bruke kunnskapen vår i en bredere sammenheng, samtidig å lære fra en annen studieretning. Robotikk er også et høyst relevant fagfelt i rask utvikling, som gjør oppgaven veldig attraktiv.

Vi vil takke våre forelesere som har gitt oss kunnskapen vi trenger gjennom årene på NTNU, veilederen vår Torleif Anstensrud og tidligere bachelorgrupper.

Innhold

Figurer

Tabeller

Sammendrag

Rapporten omhandler en underaktuert tobeinet robotprototype, dens tilknyttede komponenter og programvare gjennom vårt bachelorprosjekt våren 2022. Bilde av roboten sees i figur 3.1. Oppgaven bygger videre på tre tidligere gruppers arbeid.

Overordnet problemstilling for prosjektet er å løse maskinvareutfordringene, kartlegge programvarekompatibilitet, utvikle en funksjonell 3D-modell for simulering og visualisering, og regulering tilknyttet robotens gåmønster.

Oppstartsfasen var et forprosjekt hvor det ble laget tidsavgrensninger og arbeidspakker i tillegg til en fremdriftsplan. Tidligere gruppers arbeid pekte på en del utfordringer knyttet til komponentene, så disse er undersøkt ved å teste hvilke komponenter som var fungerende og hvilke som var defekte. Nye komponenter ble bestilt og de defekte ble byttet ut, deriblant motorer, enkodere, IMU-er og BeagleBone Black. De nye komponentene ble bestemt ut fra en kombinasjon av tilgjengelighet og kompatibilitet med eksisterende komponenter som skulle brukes videre.

Tidligere gruppers oppsett på programvaredelen var ikke mulig å gjenskape på nyanskaffet BeagleBone Black. Derfor ble det valgt å benytte nyere versjoner av programvaren som har støtte flere år fremover.

Koder til nye komponenter ble testet og utført på Arduino UNO, og ga tilfredsstillende resultat.

Det ble besluttet å beholde ROS for å videreutvikle kommunikasjonsplattformen. Gjennom terminalen i Linux ble det satt opp kommunikasjon mellom IMU-er og BeagleBone Black. Ved etterspørsel av data fra IMU-ene ble et tilfredsstillende svar til terminalen sendt. Det er blitt arbeidet med C++ kode direkte med VIM i Linux-terminalen, og ROS fra en ROS-terminal ved gjennomgang av tutorials i Turtlesim og Gazebo.

En funksjonell 3D-modell av roboten er konstruert i Fusion 360 og kan benyttes videre i sanntidsvisualisering og simulering. Reguleringsteori ble utformet for videre arbeid med roboten.

Abstract

This thesis deals with an under-actuated biped robot prototype and its associated software and components, regarding our bachelor project in the spring of 2022. A picture of the robot is shown in figure 3.1. The thesis builds on the work of three previous groups.

The problem statement is to solve the hardware challenges, map software compatibility, and develop a functional 3D-model for simulation and visualization.

The start-up phase was a preliminary project where time limits and packages in work breakdown structure were made in addition to a progress plan. A thorough hardware testing was carried out, as previous groups had challenges regarding them. After testing the defective hardware was identified and replaced. New components were ordered and the defective ones were replaced, including motors, encoders, IMUs and the BeagleBone Black. The new components were chosen based on a combination of availability and compatibility with existing components. Procured components are listed in Table 3.1.

The software structure of previous groups' was not possible to replicate, hence the latest version was chosen that have support for several years to come. Codes for new components were tested and executed with Arduino UNO with satisfactory results.

It was decided to keep ROS to further develop the communication platform. Through the terminal in Linux, communication was set up between IMUs and BeagleBone Black. When requesting data from the IMUs, a satisfactory response was sent to the terminal. Work has been done on C++ code directly in the Linux terminal using VIM, and ROS by reviewing tutorials in Turtlesim and Gazebo. A 3D-model has been constructed with physical measurements of the robot in Fusion 360 measurements that can further be used for visualization and simulation. Control theory was written for further work with the robot.

Akronymer

API Application Programming Interface. 30, 31

ASIC Application Specific Integrated Circuit. 21

EOL End Of Life. 31

GNU GNU's not Unix. 12, 13

LTI linear time-invariant. 51

LTS Long Time Support. 31

OS Operativsystem. 29

OSRF Open Source Robotics Foundation. 36

RBPi Raspberry Pi. 29

ROS Robot operating system. 29, 31

SBC Single Board Computer. 13

SCL Klokkelinje. 16

SDA Datalinje. 16

SDF Simulation Description Format. 38

SSI Synchronous Serial Interface. 22

URDF Unified Robot Description Format. 36

VCS Version Control System. 30

VM Virtuell Maskin. 43

XML Extensible Markup Language. 36, 38

Ordliste

DeepMind 3D-plattform innen programvare, skreddersydd for arbeid med kunstig intelligens. 38

GitHub vertsnettside for git repository, lett tilgjengelighet til kildekode. 37

parser prosess som analyserer syntax. 33

peer-to-peer nettverk som bruker distribuert arkitektur hvor alle enheter er likeverdige, i motsetning til klient-server-nettverk. 30

RPC Remote Procedure Call er en forespørsel-respons- protokoll. Initialiseres av klient som sender en forespørsel om å utføre en spesifikk prosedyre, til en kjent server. 29, 31

RS-422 en teknisk standard for differensiell signaloverføring, hvor hver datalinje er paret med en dedikert returlinje. 22

VIM Terminalbasert teksteditor. xi, xii

Innledning

0.1 Bachelorrapporten

Denne rapporten beskriver arbeidet med en underaktuert tobeinet robot gjennom våren 2022.

Roboten har også vært tema for en bacheloroppgave hvert av de tre foregående år, hvor tidligere grupper har bestått av ulike kombinasjoner av studenter fra automasjon/robotikk og elektronikk/sensorsystemer. Selve roboten er bygd i 2015, og i 2016 er det publisert en masteroppgave som omhandler forenkling av metoder for å simulere et stabilt energieffektivt ganglag for en slik robot. Videre i innledningen beskrives tilnærmingen til årets bacheloroppgave hvor det ble sett på status for robotens komponenter og programvare, hva som var anbefalt å se nærmere på fra tidligere års rapporter, i tillegg til årets oppgavetekst. Deretter beskrives valgt problemstilling og avgrensning for oppgaven.

Rapporten deles i tillegg til innledning drøfting og konklusjon inn i tre hoveddeler, disse er *kapittel 3 roboten*, *kapittel 4 programvare* og *kapittel 5 reguleringsteori*. Hver av disse delene har flere underkapitler som tar for enkeltkomponenter og programmer vi har jobbet med.

Enkelte ord og begrep er typiske å benytte på engelsk innen et emne som dette, disse er markert med kursiv eller fet skrift. Forkortelser linkes til listen over forkortelser eller ordliste, i hovedsak første gang de står i teksten.

0.1.1 Rapportens kapittelinndeling

Første hovedkapittel beskriver forprosjektet. I dette ligger arbeidspakker, tidsavgrensninger og fremdrift samt kontakt med veileder/oppdragsgiver. I andre hovedkapittel beskrives hva som var nødvendig å finne ut av, hvilken kompetanse som fantes i gruppa, og hvilke ressurser som eksisterte. Roboten og dens hardwarekomponenter beskrives i tredje hovedkapittel.

Programvare tilknyttet prosjektet, med valg av versjoner og kompatibilitet mellom disse, finnes i hovedkapittel fire. Femte hovedkapittel omhandler reguleringsteori. Arbeidsprosessen for bacheloroppgaven beskrives i sjette hovedkapittel, hvor det diskuteres alternativer og hva som fungerte bra. Sjuende hovedkapittel tar for seg drøftingen av prosjektet som en helhet, og rapporten avsluttes deretter med en konklusjon i siste og åttende hovedkapittel.

0.2 Problemstilling

Overordnet problemstilling for prosjektet er å løse hardwareutfordringene, kartlegge programvarekompatibilitet, utvikle en funksjonell 3D-modell for simulering og visualisering, og regulering tilknyttet robotens gåmønster.

0.2.1 Problemstilling beskrivelse

Det som ligger i hardwareutfordringer er å få oversikt over hvilke deler som finnes på robotlaben, både på selve roboten og på lageret. Videre kartlegge fungerende og defekte komponenter slik at nye deler kan skaffes og implementeres.

For å kunne benytte tidligere års arbeid kreves det å sette seg inn i tidligere års programvare, både Linux, C++ og ROS, via bachelorens GitHub.

Videre bruk av programvaren vil være å implementere et nytt design av en funksjonell 3D-modell, som siden kan benyttes for å finne riktig reguleringsparametere og kunne teste ulike gåmønstere, i tillegg til å benyttes for å simulere og visualisere roboten.

Kapittel 1

Forprosjektet

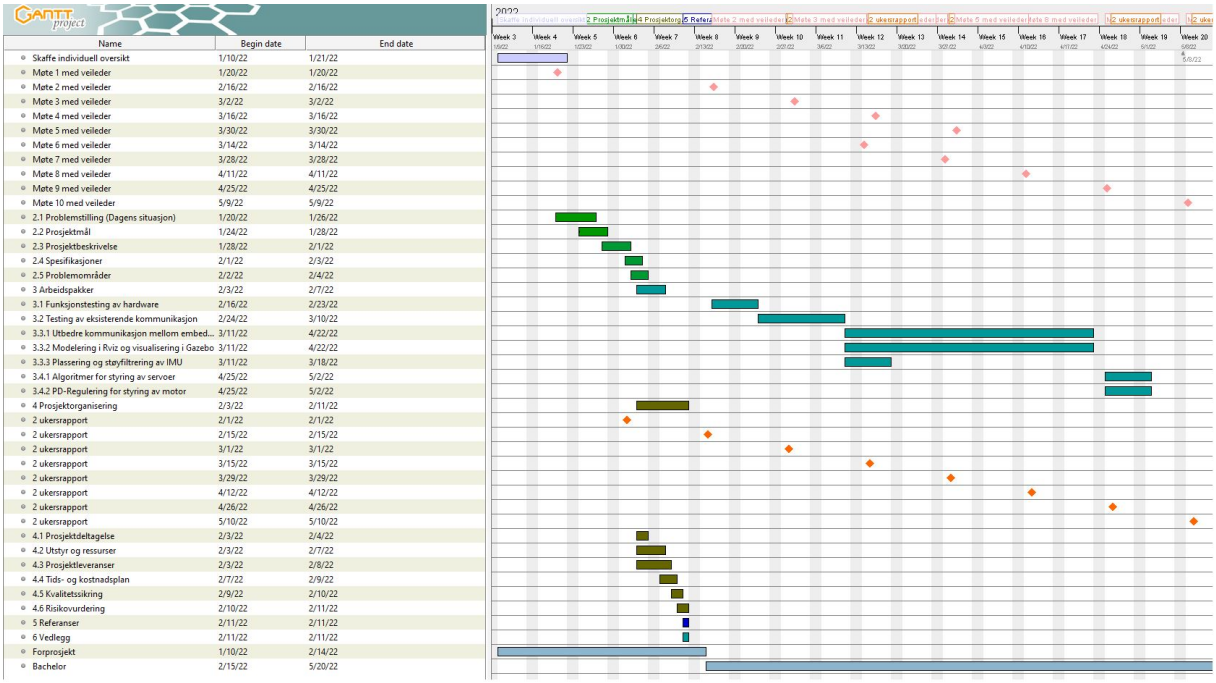
1.1 Teori forprosjekt

Hensikten med forprosjektet er å etablere et godt grunnlag om selve arbeidsoppgaven, arbeidsmetode, samarbeid og informasjonsinnhenting. Slik at forprosjektet svarer på spørsmål som "Hvilken utfordring skal gjøres/løses?, Hvordan bør oppgaven deles opp, Når skal de ulike delene gjøres?, I hvilken rekkefølge er mest hensiktsmessig å gjøre de ulike delene?, Hvilke utfordringer kan dukke opp, Hvem skal jobbe med hva?, Hvor kan man innhente informasjon som kan være til hjelp for å løse utfordringer som dukker opp?"

1.2 Resultat forprosjekt

Forprosjektet ga høyt fokus hvilke utfordringer som må løses og hvordan man bør gå frem. I dette prosjektet ble ulike utfordringer delt inn i arbeidspakker slik at det ble en oversiktlig organisering. Hvilke arbeidspakker som må gjøres, hvilken rekkefølge arbeidspakkene bør være i og hvordan arbeidspakkene henger sammen. Arbeidspakkene ble fordelt på de ulike gruppemedlemmene etter arbeidspakkens omfang og kunnskap i gruppen. Tiden per arbeidspakke ble grovt estimert etter tidligere liknende utfordringer. I forprosjektet ble det enighet innad i gruppen om hvilke tider det skal jobbes og hva som forventes av de andre gruppemedlemmene.

I den planlagte fremdriftsplanen ligger forprosjekt, arbeidspakker, tidsavgrensninger, møter med veileder og innad gruppeoppdateringer. Det er også satt hvilke gruppemedlemmer som er på de ulike arbeidspakkene. Figur 1.1 viser fremdriftsplanen i et Gantt skjema.



Name	Begin date	End date
Skaffe individuell oversikt	1/10/22	1/21/22
Møte 1 med veileder	1/20/22	1/20/22
Møte 2 med veileder	2/16/22	2/16/22
Møte 3 med veileder	3/2/22	3/2/22
Møte 4 med veileder	3/16/22	3/16/22
Møte 5 med veileder	3/30/22	3/30/22
Møte 6 med veileder	3/14/22	3/14/22
Møte 7 med veileder	3/28/22	3/28/22
Møte 8 med veileder	4/11/22	4/11/22
Møte 9 med veileder	4/25/22	4/25/22
Møte 10 med veileder	5/9/22	5/9/22
2.1 Problemstilling (Dagens situasjon)	1/20/22	1/26/22
2.2 Prosjektmål	1/24/22	1/28/22
2.3 Prosjektbeskrivelse	1/28/22	2/1/22
2.4 Spesifikasjoner	2/1/22	2/3/22
2.5 Problemområder	2/2/22	2/4/22
3 Arbeidspakker	2/3/22	2/7/22
3.1 Funksjonstesting av hardware	2/16/22	2/23/22
3.2 Testing av eksisterende kommunikasjon	2/24/22	3/10/22
3.3.1 Utbedre kommunikasjon mellom embed...	3/11/22	4/22/22
3.3.2 Modelering i Rviz og visualisering i Gazebo	3/11/22	4/22/22
3.3.3 Plassering og støyfiltrering av IMU	3/11/22	3/18/22
3.4.1 Algoritmer for styring av servoer	4/25/22	5/2/22
3.4.2 PD-Regulering for styring av motor	4/25/22	5/2/22
4 Prosjektorganisering	2/3/22	2/11/22
2 ukersrapport	2/1/22	2/1/22
2 ukersrapport	2/15/22	2/15/22
2 ukersrapport	3/1/22	3/1/22
2 ukersrapport	3/15/22	3/15/22
2 ukersrapport	3/29/22	3/29/22
2 ukersrapport	4/12/22	4/12/22
2 ukersrapport	4/26/22	4/26/22
2 ukersrapport	5/10/22	5/10/22
4.1 Prosjektdeltagelse	2/3/22	2/4/22
4.2 Utstyr og ressurser	2/3/22	2/7/22
4.3 Prosjektleveranser	2/3/22	2/8/22
4.4 Tids- og kostnadsplan	2/7/22	2/9/22
4.5 Kvalitetssikring	2/9/22	2/10/22
4.6 Risikovurdering	2/10/22	2/11/22
5 Referanser	2/11/22	2/11/22
6 Vedlegg	2/11/22	2/11/22
Forprosjekt	1/10/22	2/14/22
Bachelor	2/15/22	5/20/22

Figur 1.1: Fremdriftsplan

Kapittel 2

Behov for kunnskap og kilder

Med gitt bacheloroppgave i hånden, ble det nødvendig å sette seg inn i flere ulike programmer og systemer som var ukjente for gruppa. Eksempelvis både ROS, C++, F360 og Linux. Arbeid med ettkortsdatamaskin var også relativt ukjent.

Gjennomgang av de tidligere års Bachelorrapporter og det som fantes på GitHub fra disse, gjorde at gruppa ville undersøke om ROS kan kjøres fra BeagleBone Black og slik gjøre roboten uavhengig av stasjonær PC.

Kombinasjonen av BeagleBone Black og ROS krevde bruk av Linux og C++. Innhenting av kunnskap omkring dette ble kilder på internett, både tutorials og PDF-bøker. Enkelte av ressursene ble funnet ut fra referanselistene i de tidligere Bachelorrapportene.

Videre var de fleste av komponentene ukjente, både servocontrollere, enkodere, og IMU-er. Her ble både datablader og informasjon på nett fra andre prosjekter med slike komponenter benyttet, for å finne ut av oppkobling og nødvendige konfigurasjoner. Det var kjent at enkelte komponenter var defekte allerede i fjor og dermed måtte byttes ut.

Lang leveringstid og lave lagerbeholdninger på IMU-er og enkodere førte til at identiske komponenter ikke lot seg gjenskaffe innenfor de gitte tidsrammene for årets prosjekt. Dermed kunne ikke informasjonen og koden fra de tidligere bacheloroppgavene brukes direkte.

Gruppa var også ukjent med underaktuasjon, som vil si flere frihetsgrader enn aktuatorer.

Erfaringen i gruppa med ettkortsdatamaskin var kun bruk av en Raspberry Pi som server, i et tidligere emne. Kunnskap om oppbygningen og funksjonaliteten for BeagleBone Black, i forhold til enklere utviklingskort gruppa var kjent med, ble derfor en nødvendig del av oppgaven.

ROS hadde blitt introdusert på en labøving for to av gruppemedlemmene tidligere, så videre kunnskapsheving om ROS var en stor del i dette prosjektet. Herunder ligger valget mellom å benytte ROS eller ROS2, og programtillegg for visualisering og simulering av roboten i henholdsvis Rviz og Gazebo.

Ingen hadde direkte erfaring med C++, men alle hadde skrevet litt i C og Arduino C gjennom småprosjekter i dette studieløpet. Her ble det nødvendig å

finne ut forskjellen på C og C++, fordeler og ulemper med å benytte C++ og hvorfor det foretrekkes eller kreves i denne prosjektsammenhengen.

Linux var også helt ukjent for gruppa, så her ble det å sette seg inn i hvordan det fungerer, hvorfor det er viktig å bruke i denne sammenhengen, og vurdere alternativer om det skulle finnes.

2.1 Kunnskapsinnhenting

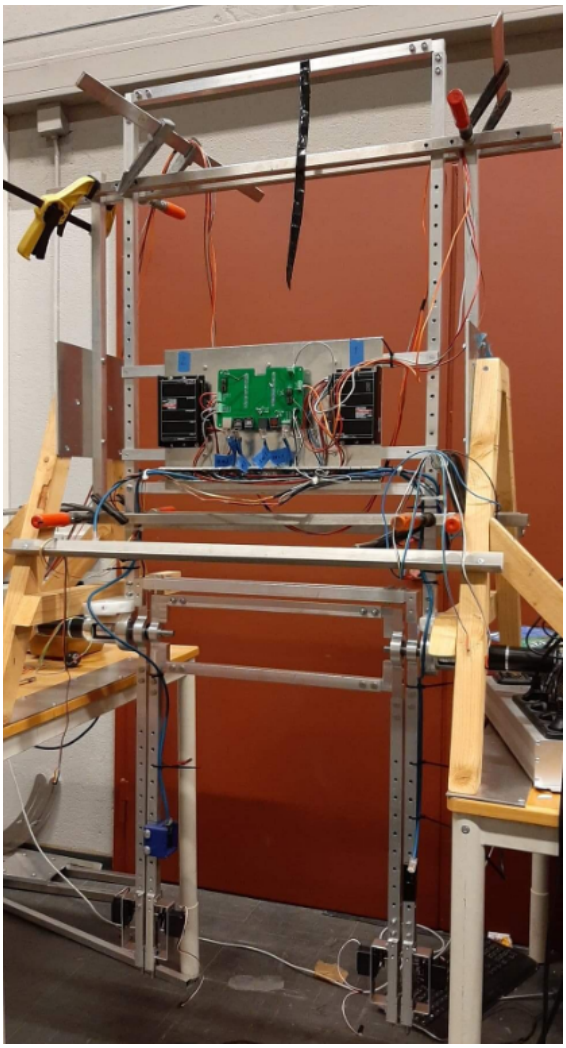
Bruk av de ulike programmene i prosjektet krevde kunnskap om siste programversjoner og oppdaterte tutorials, som gjorde at de fleste ressursene er digitale foruten de ressurspersonene vi kunne møte direkte.

Referanselisten viser hvor det ble funnet informasjon som benyttes i rapporten. I tillegg er det lest på utallige nettsider for å finne nyere oppdateringer på enkelte av elementene i prosjektet. Ettersom flere av elementene i prosjektet er åpen program- og hardvare og drives av frivillige kommer informasjonen fra utrolig mange om man ser på personnivå. Mye av kunnskapen ble funnet ved å linke seg videre fra ett sted til at annet på internett.

Gjennom å benytte robotlaben som arbeidssted kom man naturlig i kontakt med andre som jobber med lignende komponenter og var lette å spørre når sjansen bød seg. Dette ble gjort relativt uformelt, og uten spesifikke notater til hver samtale.

Kapittel 3

Roboten



Figur 3.1: Roboten i helfigur.

Roboten består i hovedsak av aluminium som vist i figur 3.1. Aluminium er et relativt lett metall, samtidig som det er robust nok til å tåle støt tilknyttet robotens gange. Aluminium er ikke magnetisk, så det genererer heller ingen støy til andre komponenter på roboten.[1] Robotens beinpar drives av hver sin motor. Den ene motoren driver det indre beinparet, og den andre motoren driver det ytre beinparet. Motorstaget er festet til et beinpar med en bevegelig akselkobling avbildet på figur 3.2. På den andre enden er beinparet festet til rammen via kulelager. På endene av hvert bein er det festet et stag som drives opp og ned av servomotorer. Overkroppen er festet til beina med kulelager avbildet i figur 3.2. Det er også kulelagre mellom innside og utside beinpar slik at de kan svinge uavhengig av hverandre. Ettersom overkroppen ikke har noen mulighet for pådrag er dette en underaktuert robot.

Roboten har mange tilkoblinger, både strøm og digitale signaler. Figur 3.3 viser en oversikt over hva som sendes hvor.

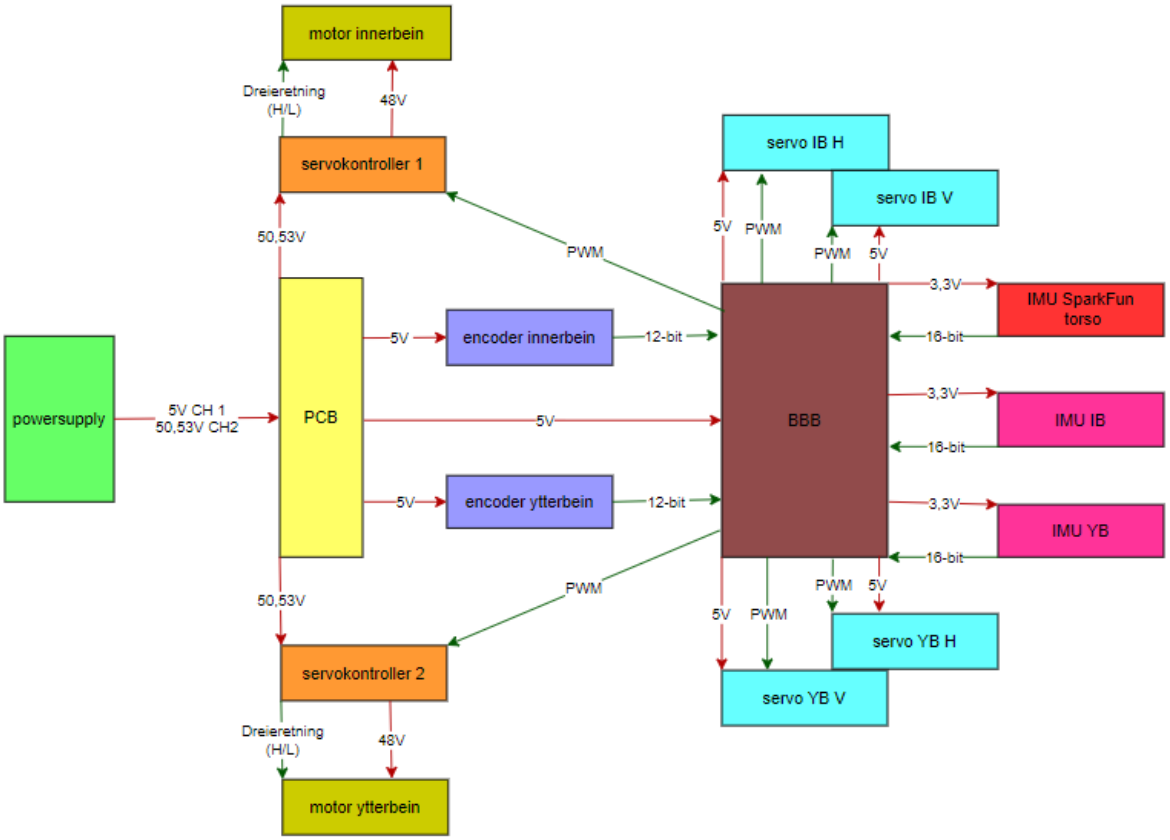
Figur 3.2: Flexledd og kulelager.

3.1 Funksjonstesting av hardware

Fungerende komponenter ble ansett som essensielt ved oppstart av prosjektet. Dermed ble det naturlig å teste eksisterende komponenter som første steg, og deretter bestille det som måtte suppleres. Enkelte av komponentene var kjent defekte, gjennom opplysninger fra tidligere bacheloroppgaver. Nye kompatible komponenter ble funnet ved å se på lagerbeholdning, leveringstid og pris.

3.1.1 Test av eksisterende enkeltkomponenter

Testing foregikk ved at den enkelte komponent ble koblet så direkte som mulig, til henholdsvis strømforsyning eller Arduino UNO, og ikke via BeagleBone Black eller servokontroller/PCB. Dette for å unngå mer enn én ukjent komponent om gangen.



Figur 3.3: Robotens signalveier, GND implisitt.

BeagleBone Black

Visuell vurdering av BeagleBone Black som var montert på roboten fra før, viste at denne hadde defekt USB-terminal ettersom den fysisk var revet av kortet.

Enkodere

Enkodere som fantes fra tidligere ble etter tur testet med en programkode og tilkobling som var identisk for hver gang, for å undersøke om det var respons. Her ble Arduino C og Arduino UNO benyttet. Kun én av fem enkodere ga ut binærverdier ved manuell rotasjon, resten var uten utslag i det hele tatt. Av disse fem tilgjengelige var det tre hunn- og to han-tilkoblinger, hvorav kun den ene han-enkoderen fungerte. Tilkobling til eksisterende motorkombinasjon er hunn-tilkobling, eventuelt bruk av adapter.

Motor-girkombinasjon

Test av motorene som var tilgjengelige på laben foregikk ved å først teste motorene med 6-til-1 girforhold som var montert på roboten, for deretter å bytte ut de med de andre motorene på laben med 26-til-1 girforhold, og teste de på samme måte.

Her ble tilgjengelig strømforsyning på laben benyttet, for å se på utslag for ulike verdier av påsatt strøm fra 0A til 3A. Spenning og strømtrekk over de ulike motorene som var tilgjengelige ble registrert og sammenlignet.

De to motorene med 6-til-1 girforhold som var montert på roboten fra før ga **ulike** utslag på beinsving. Den ene motoren ga kun et lite vinkelutslag på robotens bein uansett strømtilførsel, mens den andre motoren ga utslag som forventet i henhold til tilført strøm. Figur 3.4 viser giret som er montert på motorene.

De to motorene med 26-til-1 girforhold fungerte **likt** på begge bein, og ga fornuftig utsving i forhold til gitt strøm.

Sammenligning av tilbakesving av robotens bein med fungerende motor med 6-til-1 og motor med 26-til-1 viste at 26-til-1 ikke ga ønsket pendelbevegelse, forbi beinets utgangsposisjon før aktuering. Derimot ga motoren med 6-til-1 pendelsvingninger forbi utgangsposisjonen flere ganger før beinet stoppet opp.

Figur 3.4: Giroverføringen 6-til-1.

Uten å ha motoren koblet til roboten ble det også sett på forskjellen i strømtrekk ved maks hastighet uten belastning, kontra strømtrekk ved belastning, for å se hva som kan forventes av strømtrekk når motoren skal svinge robotens bein fremover.

Servoer

Av servoene fra Turnigy, som ble benyttet på roboten i 2021, viste det seg at tre av fire fungerte som vi ønsket. Dette ble testet ved å kjøre en kode i Arduino C og koble servo direkte til Arduino UNO. Ved å sette vinkelutslag individuelt for hver av de to som var montert på roboten lykkes det å få nok utslag til å sette leddet i lås med eksisterende montering. De to servoene fra Futuba som fantes fra tidligere bruk var defekte. Den ene var uten respons, og den andre gikk ikke fritt og ga dermed heller ikke vinkelutslag i overenstemmelse med databladet.

IMU

IMU ble testet ved bruk av I2C-kommunikasjon med kode fra eksisterende bibliotek for akkurat denne modulen i Arduino C. Tre IMU-er av modell LSM9DS1 fantes på laben men de hadde ikke sammenfallende oppsett. Det vil si at den IMU som ga fornuftige verdier ut hadde ikke brutte komponenter, til motsetning fra de to andre IMU-ene som hadde brutt SDO-jumper og CS-jumper.

Servokontrollere

Servokontrollerene ble ikke testet da det tidligere ikke er beskrevet problemer tilknyttet disse.

PCB-er

PCB-er for IMU og enkodere ble sjekket opp mot kobling på hoved-PCB, og dette stemte overens. Etter vurdering ble det bestemt at enkoder PCB-ene ikke skal benyttes ettersom tidligere gruppe har fått ødelagt flere enkodere når de har vært tilkoblet disse PCB-ene.

3.1.2 Nye komponenter

Tabell 3.1: Nye deler som ble bestilt.

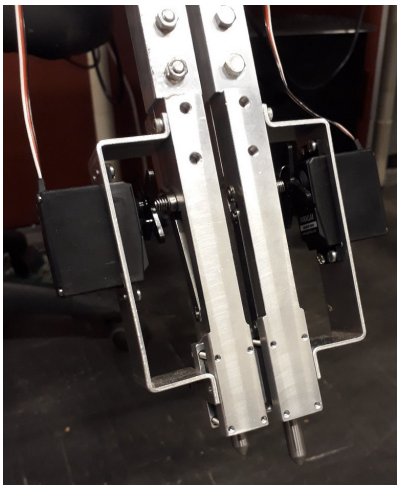
Antall	Komponent	Leverandør
1	BeagleBone Black	DigiKey
2	enkodere AMT233A-V	Mouser
2	Header til enkoder	Digikey
1	Motorkombinasjon, motor og 6:1-gir	Maxon
4	Servoer Parallax	Mouser
3	IMU-er DFrobot	Mouser

På grunn av kjente skader på eksisterende BeagleBone Black ble det bestilt en ny.

To nye enkodere var nødvendig å skaffe, ettersom det ikke fantes kompatible enkodere som ikke var ødelagte. Enkoderne som til slutt kunne leveres tidsnok ble av typen absolutt, selv om vi i utgangspunktet hadde tenkt å fortsette med inkrementell som tidligere.

Headere for enkodere ble bestilt slik at det kan festes kabler til enkoderne.

Én ny motorkombinasjon bestående av motor tilkoblet 6:1-girelement ble bestilt lik som eksisterende motorkombinasjon som fungerte på det ene beinet, for å ha lik motorkombinasjon på hvert av robotens bein. Motoren har modellnummer 148877 og er en DC-motor med grafittbørster. Girelementet har modellnummer 260551, og har 6-til-1 girforhold med keramiske tannhjul.



Figur 3.5: Tåledd, venstre inntrukket, høyre utstrukket.

Fire nye servomotorer med varenummer 900-00005 ble bestilt for å ha lik type på alle fire av robotens tåledd, ny servo fra Turnigy var umulig å oppdrive. Disse nye servomotorene har

utslag på 180 grader, som er mer enn nok til utstrekking og tilbaketrekking av robotens tær. Se bilde av tåledd i figur 3.5

Testing av IMU-er som var benyttet tidligere ga ikke samsvarende målinger for de tre IMU-ene. Kun den ene virket å gi realistiske verdier ved sammenligning av verdier fra hver av de, ved bruk av samme testkode. Ettersom samme IMU som var benyttet tidligere ikke var mulig å skaffe, ble det bestilt tre nye fra en annen produsent for kunne benytte lik type IMU på torso og hvert av robotens bein videre.

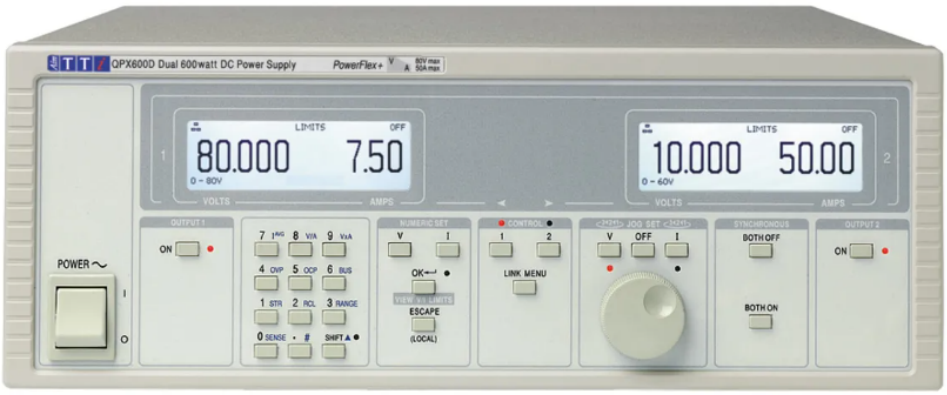
3.1.3 Implementering av de nye komponentene

Det viste seg at basen til enkoderfestet manglet noen millimeter for å få plass over motoren. Basen på motoren har en omkrets på 13 mm mens enkoderfestet har et hull på 12.6 mm. Ved å file hullet i enkoderfestet litt større passet enkoderen på motoren.

Servoene som ble bestilt var et par millimeter bredere enn de eksisterende Futuba servoene. Ved å bruke en fil ble de eksisterende festebrakettene modifisert slik at de nye servoene fikk plass.

Robotens komponenter blir videre beskrevet i respektive teoriavsnitt hvor tilkoblinger, muligheter og alternativer blir belyst.

3.2 Strømforsyning



Figur 3.6: Strømforsyning, DC Power Supply QPX600D

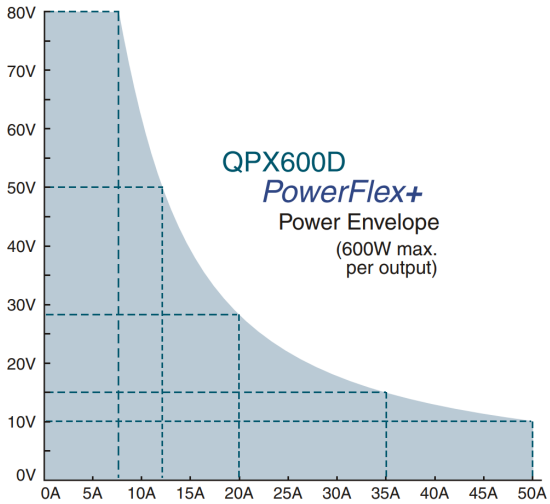
3.2.1 Teori strømforsyning

En strømforsyning leverer elektrisk energi til påkoblede laster. Strømforsyningen som benyttes er en QPX600D fra TTi avbildet i figur 3.6. Dette er en strømforsyning med to kanaler som er kapabel til å levere maksimalt 600W per kanal

med maksimal spenning på 80V, og maksimal strøm på 50A. Kurve strøm og spenningsverdier er avbildet i figur 3.7. Kanalene kan også serie- og parallellkobles for å enten oppnå en spenning på 160V ved parallellkobling eller en strøm på 100A seriekoblet. Den har også mulighet for styring av strøm og spenning via analoge spenningssignaler på 0V-5V eller 0V-10V for hele strøm- og spenningsspekteret. Strømforsyningen kan på likt vis gi tilbakemelding på hvor mye strøm og spenning den gir ut via like analoge spenningsverdier. I tillegg til analog logikk har strømforsyningen også mulighet til å ta imot og sende informasjon via Ethernet, RS232 og USB. [2]

3.2.2 Resultat strømforsyning

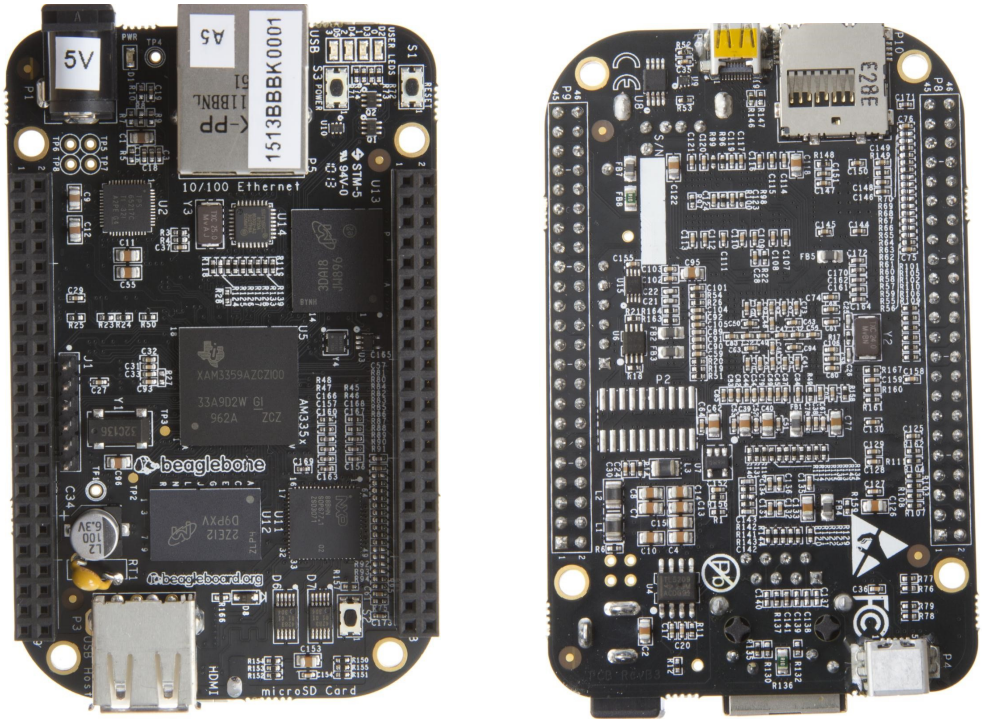
Den ene kanalen brukes for å drive servokontrollerne og motorene, mens den andre kanalen driver BeagleBone Black, IMU-ene, enkodere og servomotorene. Kanalen som driver motorene og servokontrollerne er satt til 50.53V 6,32A som tar hensyn til spenningsfallet over servokontrolleren og maksimalt nominelt strømtrekk på motorene. Den andre kanalen settes til 5V 2A som er nok til å drive enkoderne og BeagleBone Black som igjen forsyner IMU-er og servomotorer.



Figur 3.7: Kurve som viser sammenhengen mellom spenning og strøm

3.2.3 Drøfting strømforsyning

Denne strømforsyningen er mer enn kraftig nok til å drive alt av utstyr vi har behov for i dette prosjektet. Det er i hovedsak motorene som trekker strøm på roboten, og strømforsyningen har nok effekt til å drive 4 ganger så mange motorer som den gjør nå. Den har også muligheter for analog og digital kommunikasjon til andre komponenter, noe vi ikke benytter oss av. Om strømforsyningen ikke hadde vært anskaffet i forkant av overtagelsen av oppgaven hadde det ikke vært hensiktsmessig å kjøpe den da en rimeligere strømforsyning ville gjort akkurat samme nytte for prosjektet.



(a) Overside. **(b)** Underside.

Figur 3.8: BeagleBone Black.

3.3 BeagleBone Black

3.3.1 Teori BeagleBone Black

BeagleBone Black er en innebygd ettkortsdatamaskin som kjører Debian GNU/Linux operativsystem. Et annet eksempel på innebygd ettkortsdatamaskin er Raspberry Pi. Figur 3.8 viser over- og undersiden av BeagleBone Black.

BeagleBone har også utviklet en rimeligere versjon som kalles BeagleBone Green. Forskjellen er at BeagleBone Black har HDMI/audio maskinvare, som BeagleBone Green ikke har. Samme *plateavbildningsfil* kjøres på BeagleBone Green og BeagleBone Black.

BeagleBone Black har all informasjon tilgjengelig under gratis lisens (FreeBSD), både maskinvare og programvare. Alle kretsskjemaer og komponenter er offentlige slik at hvem som helst skal kunne lage ettkortsdatamaskinen.

På originale produsenters sider og informasjon som lenkes, finnes absolutt siste kommentar i august 2021, utenom dette er det ingen offisielle oppdateringer siden siste Debian versjon ble lagt ut i april 2020. Siste oppdatering på systemmanual i PDF som finnes på GitHub[3] er lagt inn i august 2021. Denne GitHuben er en av de bedre kildene for å finne de nyeste oppdateringene om BeagleBone Black fremover. Grunnen er at det ikke rekker å utgis i trykket form før det er nye

oppdateringer igjen, noe som også påpekes i flere av kildene som er benyttet i bachelorprosjektet.

Ettkortsdatamaskin, forkortes SBC, er en komplett datamaskin konstruert på ett enkelt kretskort, og har mikroprosessor, minne, og inn- og utganger, i tillegg til andre egenskaper som kreves av en funksjonell datamaskin. **Innebygd ettkortsdatamaskiner** er enheter som tilbyr alle nødvendige inn- og utganger uten behov for å koble til ekstra kretskort. Til sammenligning brukes begrepet **ettkortsdatamaskin** først og fremst om arkitektur der SBC plugges i et annet kretskort for å fungere som inn- og utgangsenheter. [4][5]

BeagleBone Black ble lansert i 2013, revisjon C i 2014. Den har 512MB RAM, 1 GHz klokkefrekvens, 32-bit ARM Cortex-AB prosessor og har 4GB eMMC flashminne. BeagleBone Black leveres med Debian GNU/Linux ferdig installert, så lenge BeagleBone Black er originalt produsert.

Prosessoren har i tillegg to programmerbare 32-bits sanntidsenheter (PRU) som kan kjøre opp mot 200MHz hastighet. På BeagleBone Black er det opp mot 69 GPIO-er å velge mellom for tilkobling vi trenger for signaler på roboten, i tillegg til en rekke andre muligheter som ligger i det å ha en komplett datamaskin på ett kort. [6] [7] [8]

Systemmanual for BeagleBone Black finnes på GitHub[9] hvor alt av fysiske elementer er beskrevet i tillegg til mulige tilkoblinger, via PC eller ikke, og det meste man ellers måtte behøve av informasjon om BeagleBone Black.

Figur 3.9 viser hvordan $beagleboard.org/Support/bone101/$ gir en ryddig oversikt over ulike inn- og utganger med tabeller for eksempelvis PRU, GPIO og I2C-pins. På denne nettsiden finnes også påpekninger om hvilke pins som benyttes til *flere* funksjoner, og dermed kan være utilgjengelige for protokoll man ønsker å bruke.

Tilkobling, oppstartsalternativer, nettilkobling og IP-adresser

Nye BeagleBone Black's kommer som nevnt med Debian ferdig installert, men om man trenger å oppdatere til Debian fordi man har en eldre Linuxversjon på BeagleBone Black finnes en steg-for-steg-instruks på siden $beagleboard.org/getting-started\#update$. Her anbefales å laste ned *IoT-plateavbildningsfil* om man ikke er avhengig av å bruke GUI, ettersom denne *plateavbildningsfilen* ikke tar like stor plass som andre. Videre anbefales installasjon av *balenaEtcher* som skriver riktig nedlastet versjon på valgt SD-minnekort.

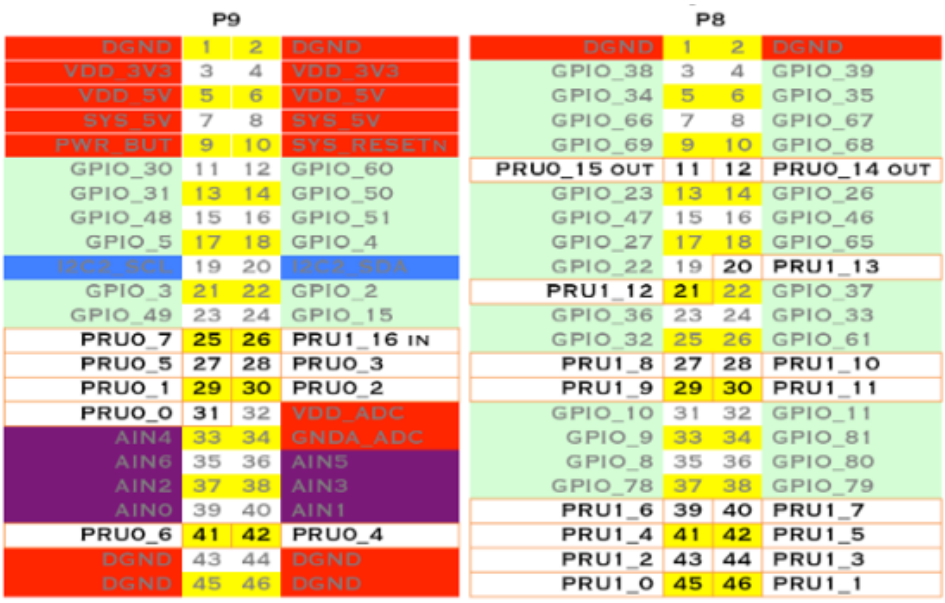
Siste tilgjengelige versjon er fra april 2020, og gjelder skriving til **eMMC** via SD-kort og uten GUI:

$AM3358Debian10.32020-04-064GBeMMCIoTFlasher$.

Alternativt **kun** via SD-kort, uten GUI:

$AM3358Debian10.32020-04-064GBSDIoT$

Siste oppdatering på $elinux.org$, en av de få plassene både Linux, BeagleBone Black, Ubuntu og ROS nevnes i samme sammenheng, er gjort i mars 2020. Herfra linkes kun til installasjon av ROS Melodic for Ubuntu Bionic(18.04), samtidig som



P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
PRU0_7	25	26	PRU1_16 IN
PRU0_5	27	28	PRU0_3
PRU0_1	29	30	PRU0_2
PRU0_0	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
PRU0_6	41	42	PRU0_4
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
PRU0_15 OUT	11	12	PRU0_14 OUT
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	PRU1_13
PRU1_12	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
PRU1_8	27	28	PRU1_10
PRU1_9	29	30	PRU1_11
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
PRU1_6	39	40	PRU1_7
PRU1_4	41	42	PRU1_5
PRU1_2	43	44	PRU1_3
PRU1_0	45	46	PRU1_1

Figur 3.9: Eksempel tabelloversikt inn- og utganger på BeagleBone Black.

det påpekes at det per den datoen finnes *ROS-builds* for Ubuntu Bionic armhf. Går man via ROS wiki istedenfor, vil man finne at siste versjon av ROS derimot er Noetic, og denne benyttes på Ubuntu Focal(20.04), hvor det fortsatt støttes arm32-arkitektur.

Før man kobler strømforsyning til BeagleBone Black settes SD-kortet inn, for deretter koble til BeagleBone Black mens man holder inne BOOT-knappen. For BeagleBone Black og enkelte andre versjoner av BeagleBoards skal det videre følges en egen instruks for å skrive *plateavbildningsfilen* til eMMC på BeagleBone Black. Se eventuelt

elinux.org/Beagleboard : BeagleBoneBlack_Debian#Flashing_eMMC

som gir seks kommandolinjer i Linux. Disse seks linjene gjør at *plateavbildningsfilen* skrives til eMMC, derfor er det viktig at SD-kortet fjernes når skrivingen til eMMC er ferdig for at re-skriving ikke skal fortsette repetitivt. Skrivingen er ferdig når alle fire USRx LEDs er avslått eller lyser konstant.

Når overføringen til eMMC fra SD-kortet er ferdig kobles strømforsyningen fra, deretter fjernes SD-kortet, og strømforsyningen kobles så til igjen.

Det kan også kjøres Debian-*plateavbildningsfil* fra SD-kortet uten å skrive det til eMMC på BeagleBone Black, SD-kortet må da stå i hele tiden og man går rett på neste steg.

Om det ikke er nødvendig å oppdatere *plateavbildningsfil* før bruk, går man rett på å koble til strømforsyning på 5V, enten via USB eller Barrel Jack. Her er det viktig å vite om man skal kjøre BeagleBone Black fra SD-kort, eller fra eMMC om *plateavbildningsfil* er skrevet dit som beskrevet over.

På en Linuxmaskin vil tilkoblet BeagleBone Black kjøre en DHCP-server og gi

maskinen en IP-adresse på 192.168.6.1, og 192.168.6.2 for seg selv.

Ved bruk av USB-kabel for internettilkobling til BeagleBone Black er ikke VM å anbefale. I det tilfellet anbefales ethernetkabel. Bruk av SSH med default debian-*plateavbildningsfil* har brukernavn **debian** og passord **temppwd**. Ved kjøring på Linux er det i utgangspunktet ikke nødvendig å legge til drivere for USB-serielltilgang til BeagleBone Black, i motsetning til Windows og Mac OS X. [10] [11] [12] [13]

3.3.2 Resultat BeagleBone Black

Tabell 3.2 viser tilkoblingspunktene på BeagleBone Black som benyttes i dette prosjektet. Selv om det var veldig få BeagleBone Black å finne, kan det være enkelt å skaffe BeagleBone Green, som er lik bortsett fra HDMI/audio-muligheter. Den nye BeagleBone Black som er merket med BBBVC som indikerer versjon C, og 202202 som indikerer produksjon i februar 2022.

3.4 Servokontroller

Figur 3.10: ESCON 70/10 servokontroller

3.4.1 Teori servokontroller

En servokontroller tar inn ett eller flere både digitale og analoge signaler, og benytter signalene til å styre en eller flere tilkoblede motorer. Servokontrolleren har også mulighet for å sende ut analoge og digitale signaler som kan gi informasjon

Tabell 3.2: Tilkoblinger til BeagleBone Black

Pin BeagleBone Black	Tilkoblet komponent	Signaltype
P8 13	Servo IB H	PWM
5V	Servo IB H	spenning
GND	Servo IB H	spenning
P8 13	Servo IB V	PWM
5V	Servo IB V	spenning
GND	Servo IB V	spenning
P8 17	Servo YB H	PWM
5V	Servo YB H	spenning
GND	Servo YB H	spenning
P8 17	Servo YB V	PWM
5V	Servo YB V	spenning
GND	Servo YB V	spenning
3.3V	IMU Torso	spenning
SDA	IMU Torso	byte
SCL	IMU Torso	byte
GND	IMU Torso	spenning
3.3V	IMU IB	spenning
SDA	IMU IB	byte
SCL	IMU IB	byte
GND	IMU IB	spenning
3.3V	IMU YB	spenning
SDA	IMU YB	byte
SCL	IMU YB	byte
GND	IMU YB	spenning
P8 19	Servokontroller 1	PWM
P9 14	Servokontroller 2	PWM
P8 11	Enkoder IB	byte
P8 12	enkoder YB	byte
P9 5	PCB	spenning

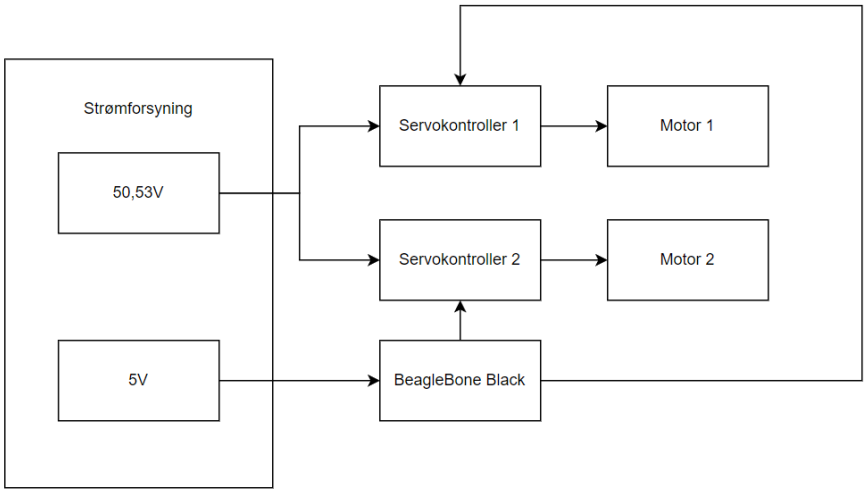
om hastighet, faktisk strømtrekk, kommunikasjonsfrekvens og temperatur. Servokontrolleren har også mulighet for å ta inn inkrementell enkoderdata og regulere med PID ved hjelp av enkoderdataene. Det er også innganger på servokontrolleren for å ta inn Hall effect sensorer, eller magnetismesensorer. [14]

3.4.2 Resultat servokontroller

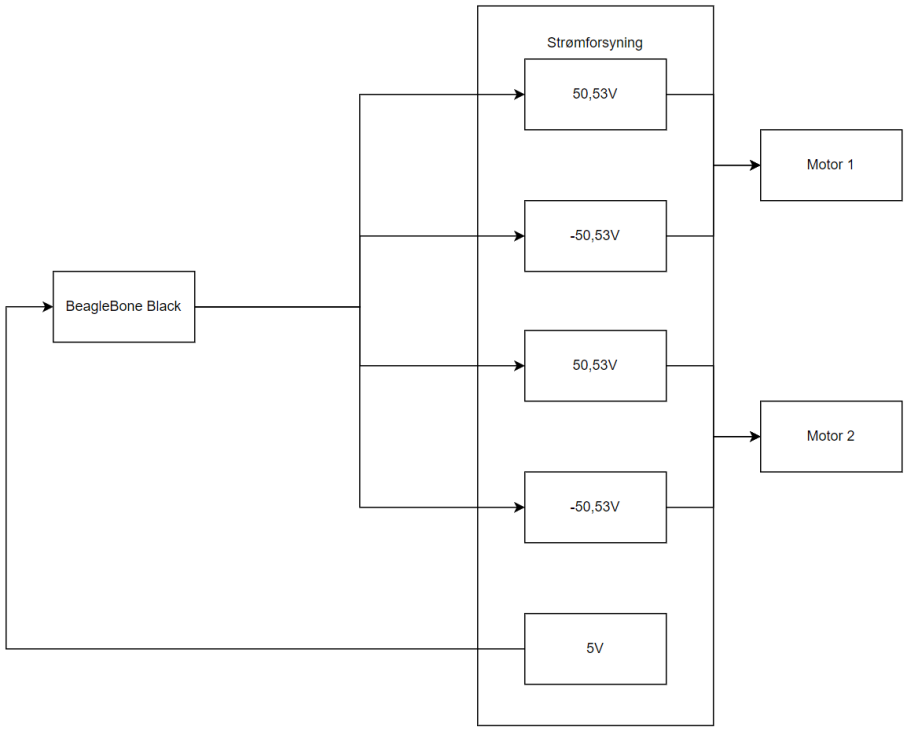
På roboten blir det benyttet Maxon sin Escon 70/10 avbildet i figur 3.10 til å styre kraft og retning til motorene. Den tar inn alt fra 10V-70V og sender ut 95% av spenningen som blir tatt inn videre til motoren. Ved å gi servokontrolleren 50.53V gir den ut 48V til motorene. De digitale og analoge inngangene og utgangene er programmerbare i Maxon sin **ESCON** programvare[14]. Servokontrollerne er satt opp til å ta imot to digitale signaler fra BeagleBone. Det ene signalet er et PWM signal som gir servokontrolleren informasjon om pådraget motoren skal ha, mens det andre signalet er et logisk lavt eller høyt signal som avgjør hvilken retning motoren roterer. Den innebygde enkoderfunksjonaliteten benyttes ikke da våre enkodere er absolutte, og det er ikke kun enkoderne som forteller noe om hva pådraget skal være.

3.4.3 Drøfting servokontroller

Det å benytte seg av en servokontroller er en elegant måte å styre motorene på, spesielt med tanke på at hver motor kan trekke opp mot 3,17A i 48V. Den åpenbare fordelen med å benytte seg av en servokontroller er at strømforsyningen kan stå på en konstant verdi. Det er servokontrolleren som tar seg av spenningsretning og strømstyrke videre til motorene. Om en servokontroller ikke hadde vært benyttet måtte BeagleBone ha styrt strømforsyningen direkte. Det hadde vært mulig da strømforsyningen kan ta imot analoge spenningssignaler som forteller strømforsyningen hvor mye strøm og spenning den skal sende ut til en hver tid, men da måtte motorene hatt to kanaler hver der kanalene hadde hatt motsatt polaritet i forhold til hverandre for å kunne styre retningen motoren skal rotere. Totalt ville det da vært fem kanaler da resten av roboten også må ha en egen kanal. Dette tallet kunne vært redusert til tre kanaler om strømforsyningen hadde hatt programmerbar polaritet på utgangen, noe den ikke har [2]. Den alternative signalflyten uten bruk av servokontroller er avbildet i figur 3.11



(a) Med servokontroller



(b) Uten servokontroller.

Figur 3.11: Forenklet signalflyt med og uten servokontroller

3.5 Motor

3.5.1 Teori motor

En DC motor er en roterende elektromotor som omgjør likespenning til mekanisk energi. En likestrømsmotor består av en stator og en rotor. Statoren består av to magneter (enten permanente magneter eller elektromagneter) som genererer et elektromagnetisk felt mellom seg. Rotoren består av en spole med en metallkjerne som det sendes strøm igjennom. Endene til spolen er festet til børster som igjen har kontakt med kommutatoren. Når det sendes strøm gjennom en spole i et magnetisk felt virker det en magnetisk kraft på spolen kalt Lorentzkraft. Kommutatoren styrer retningen til strømmen slik at rotoren fortsetter å rotere i samme retning.[15] Motor 148877 fra Maxon har 1 statorpar (1 positiv og 1 negativ magnet) og 13 kommutator segmenter.[16] Motoren er avbildet i figur 3.12.

Figur 3.12: DC-motor, modell 148877.

3.5.2 Resultat motor

Motorene er koblet opp mot servokontrolleren beskrevet i forrige delkapittel. Ved påsatt spenning beveger beinparene på seg, samtidig som beinparene svinger fritt når det ikke er påsatt spenning.

3.5.3 Drøfting motor

Motorene vi benytter oss av er på 150W og en “Stall torque” eller holdende dreiemoment på 2560 mNm. Det vil si at det er så mye kraft som kan virke på akslingen som gjør at rotasjonen blir null. [17] Det gjelder kun et lite øyeblikk da motoren kan overopphete av å holde det dreiemomentet over lengre tid. Maksimalt kontinuerlig dreiemoment er på 187 mNm. [16] Det har vært diskutert i tidligere rapporter om at motorene ikke er kraftige nok for å opprettholde et stabilt gangmønster. Tidligere simuleringer av roboten viser at for et gangmønster med skrittlengde på 15 cm og en overkroppsvinkel på -0,01 radianer trenger beinene et maksimalt pådrag på 200 mNm. [18] Motoren har påmontert en 6:1 girboks som vil si at motoren kontinuerlig kan levere et moment på 1122 mNm på beinet. Det skal være mer enn tilstrekkelig moment for å tilfredsstille det gangmønsteret. Ved påsatt spenning og strøm på motorene er det tydelig at motoren kan gi et beinutslag som tilsvarer en større steglengde enn 15 cm.

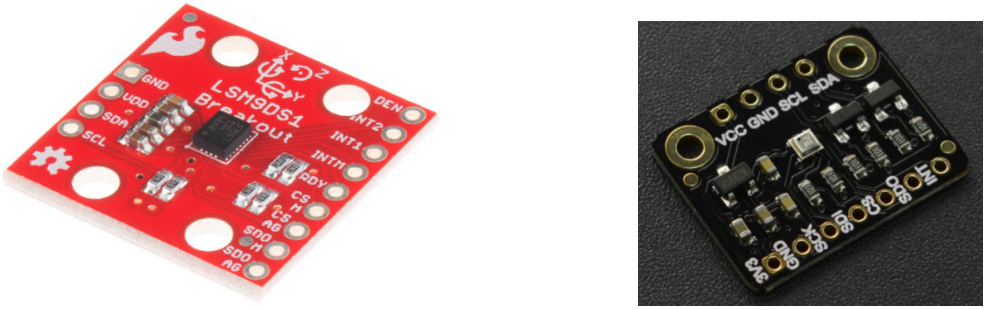
3.6 IMU

3.6.1 Teori IMU

IMU står for "inertial measurement unit". (Treghetsmålingsenhet) og er definert som en sensor med 9 frihetsgrader som måler orientering, fart, og akselerasjonskrefter ved å kombinere et akselerometer, gyroskop og et magnetometer i en og samme sensor.

For å benytte flere IMU samtidig er det viktig å kunne skille mellom hvilken sensor som gir fra seg hvilken data. Et vanlig oppsett er I2C.

Noen IMU-er har et sett med forhåndsbestemte I2C-adresser og andre kan man bestemme adressen valgfritt. Det mest vanlige er to forhåndsbestemte adresser for den enkelte IMU-modellen. Dette blir avgjørende i en I2C-sammenheng da den innhenter data fra en sensor på en fast adresse slik at systemet vet hvilken sensor den kommuniserer med. Mer om I2C-kommunikasjonen vil være beskrevet i hovedkapittel 4 programvare. IMU-ene benyttet i prosjektet er en IMU modul fra SparkFun med LSM9DS1 sensor, og en DFRobot IMU modul med BMX160 sensor. Begge er avbildet i figur 3.13



(a) IMU fra SparkFun **(b)** IMU DF Robot

Figur 3.13: IMU-er benyttet i prosjektet.

Multiplekser

En multiplekser er en enhet som velger mellom flere digitale eller analoge inngangssignaler og videresender den valgte inngangen til én enkelt utgangslinje. Valget styres av et eget sett med digitale innganger, kjent som utvalgte linjer.

Teori IMU SEL0337 BMX160

Sensoren leveres fra DFROBOT og har 9 frihetsgrader. 16-bit gyroskop, 16-bit akselerometer og geomagnetisk sensor. Selve IMU-en har to faste adresser 0x68 og 0x69. Forskjellen på disse to er en fysisk kortslutning på selve IMU-en. Med kortslutningen så vil adressen endre seg fra 0x68 til 0x69. Fra lav til høy. Disse

IMU-ene opererer med en spenning mellom 3.3V og 5.0V . Støtter både I2C og SPI protokoll. [19]

Teori IMU LSM9DS1

Sensoren leveres fra Sparkfun og har 9 frihetsgrader, to adresser for lav og to adresser for high. Magnetometer har en fast adresse ($0x1b$) og ($0x6b$) for akselerometer og gyroskopet. Denne IMU kan også kortsluttes slik at disse adressene endrer seg til ($0x1c$) og ($0x6c$). Disse går på 3.3V spenning. Både I2C- og SPI-protokoll støttes. [20]

3.7 Enkoder

For å registrere utslag på beinsving, og videre kunne beregne nødvendig motorpådrag for neste beinsving, trengs målinger som kan sammenlignes med IMU-ene sine verdier.

I prosjektet benyttes absolutte enkodere til dette, ved at beinsvingen registreres som et vinkelutslag. Hver enkoder er festet til motorakslingen på tilhørende bein. Tilgjengelighet, leveringstid og kompatibilitet med motor begrenset utvalget for dette prosjektet.

3.7.1 Teori enkoder

Rotasjonsenkoder, også kalt akslingsenkoder, kan defineres som en type posisjonssensor. Enkoderen konverterer vinkelposisjonen eller rotasjonsbevegelsen for en aksling, til et analogt eller digitalt signal. De to hovedtypene innen rotasjonsenkodere er absolutt enkoder, og inkrementell enkoder. Forskjellen på disse er at absolutt-enkodere viser den aktuelle vinkelposisjonen og dermed fungerer som en vinkeltransduser, mens inkrementell-enkodere gir informasjon om selve rotasjonsbevegelsen.

Absolutt-enkodere opprettholder posisjonsinformasjonen når strømmen slås av, dermed er enkoderverdien umiddelbart gitt når strømmen slås på. Forholdet mellom enkoderverdien og den fysiske posisjonen for enkoderen bestemmes ved montering. Derfor trenger ikke systemet tilbakestilles til noe kalibreringspunkt. Digitale absolutt-enkodere gir ut en unik digital kode for hver eneste av akslingens vinkler.

Inkrementell-enkoder gir en umiddelbar tilbakemelding om endring i posisjon, og må derfor nullstilles i en 'hjem'-posisjon for å kunne benyttes til absoluttposisjonsmålinger. Denne typen enkoder er mest brukt av alle rotasjonsenkodere, på grunn av dens evne til å gi posisjonsinformasjon i sanntid.[21]

Absolutt enkoder modell AMT233A-V

I prosjektet er det benyttet en digital seriell absolutt enkoder AMT233A-V som vist i figur 3.14 Denne benytter en patentert kapasitiv ASIC-teknologi, og kan

gi ut en 12-bit absolutt posisjon. Forsyningsspenningen anbefales til å være 5V. Nullposisjonen kan settes ved et digitalt knappetrykk via GUI-programvare(AMT Viewpoint; https://www.cuidevices.com/amt-viewpoint-mouser):

Figur 3.14: Kapasitiv enkoder AMT233A-V, med monteringssett og header.

Denne nullposisjonen lagres i enkoderens minne, og bevares selv om strømtilførselen slås av. AMT23-serien benytter 3-Wire SSI, som er en kommunikasjonsprotokoll av typen synkron seriell simplex enveis master-slave, basert på RS-422-standarder.

Den kapasitive absolutt enkoderen inneholder en asymmetrisk plate som roterer. Mellom to elektroder vil platen endre kapasitansen som dermed er målbar, og beregnes til en vinkelverdi. Det finnes også innebygde enkodere i en del permanentmagnet-børsteløse motorer brukt innen robotikk. AMT233A-V er en singel-turn enkoder og gir dermed verdier ut fra 0 til 360 grader, én omdreining. Multi-turn er enkodere som registrerer og kan lagre mer enn én omdreining.[21][22][23]

Tabell 3.3: Pinout Enkoder AMT233A-V

Nr	Funksjon
1	+5V
2	DATA
3	CLOCK
4	GND
5	MODE
6	CHIP SELECT

3.7.2 Resultat enkoder

Tabell 3.3 viser inn- og utganger for enkoderen. Når tilkoblingsplugg kables ferdig følges denne tabellen for oversikt, ved tilkobling til servokontroller. Mode pin brukes av AMT Viewpoint ved konfigurasjon av enkoderen, og settes ellers flytende ved normal bruk.[24]

3.8 Servomotor

3.8.1 Teori servomotor

Servomotorer er motorer kan deles opp i standard servomotorer, og kontinuerlige servomotorer. Standard servomotorer tar inn et PWM signal som igjen forteller servomotoren hvilken posisjon den skal stå i. En kontinuerlig servomotor tar inn et PWM signal som forteller servomotoren hvilken hastighet og hvilken retning den skal bevege seg i. Standard servomotorer har enten en enkoder eller et potmeter i seg som gjør at servomotoren alltid vet hvilken posisjon den er i. De servomotorene med potmeter kan kun settes av eller på, mens servomotorer med enkodere har muligheten til å justere hastigheten. Ofte er selve motoren i en servomotor en enkel børstet DC-motor. [25] Figur 3.15 viser servomotoren benyttet i oppgaven.

Figur 3.15: Servomotor Parallax

3.8.2 Resultat servomotor

Servomotorene som ble bestilt er standard servomotorer med et vinkelspenn på 180 grader, da det er mer enn nok vinkelutslag for å trekke inn og strekke ut tærne til motoren.[26] Ettersom at standard servomotorer kun er opptatt av posisjon blir det enklere å finne maksimalt utslag av tær da verdiene gitt til servomotor ikke er avhengig av tid.

3.8.3 Drøfting servomotor

Servomotorer passer ypperlig til formålet med å slå ut og trekke inn tær da det er posisjonen på servomotoren som er kritisk. Når PWM verdiene for utstrakt tå er funnet kan en enkelt gi samme signal til servomotoren hver gang den skal strekke ut tærne, og hver gang den skal trekke tilbake tærne. Tærne som er konstruert på roboten har også en egen låsemekanisme som gjør at servomotorene i seg selv ikke bærer noen form for vekt. All vekt hviler på robotens ramme, og dermed var det ikke nødvendig å gå til anskaffelse av servomotorer som tåler mye last. En alternativ måte å trekke inn og slå ut tærne kan være å bruke enkle DC-motorer istedenfor servomotorer. Problemet da blir at du får en funksjon som avhenger av tid. Et pådrag på x antall watt på en DC-motor over y antall tid vil i teorien gi samme vinkelutslag på motoren hver gang. I praksis vil ikke det stemme da det er mange forstyrrelser som vil virke på motoren, som for eksempel støt i bakken. Over tid vil det dannes avvik som kan bli stort nok til at tærne ikke blir fult utstrekt. Ved å benytte seg av servomotorer vil resultatet alltid bli det samme da servomotoren alltid vil forholde seg til den vinkelverdien den får.

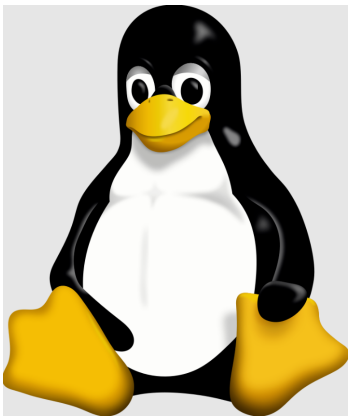
Kapittel 4

Programvare

4.1 Linux og C++

4.1.1 Teori Linux

Operativsystemet Ubuntu 20.04 ble benyttet både på Beaglebone Black og stasjonær PC på laben. Ubuntu 20.04 er et operativsystem som er en del av linuxfamilien, noe som betyr at den er bygget på toppen av den unix-liknende linuxkjernen. Den kalles *unix-liknende* av historiske grunner. Bell Labs, som var Innovasjonsavdelingen hos den amerikanske telekommunikasjonsgiganten AT&T, jobbet med et operativsystem de kalte Multiplexed information and Computing Service (Multics).[27] De trakk seg fra prosjektet, men en ansatt, Ken Thompson, bestemte seg for å lage en enklere versjon av systemet som han kalte Unix. Unix er kjent for *unix-filosofien*, som kort fortalt er *do one thing and do it well*. Dette betyr i praksis at programmer skal være så enkle som mulig. Dette gjør det enkelt for ett program å inkludere flere andre program som en *dependency*, noe som fører til økt kodegjenbruk.



Figur 4.1: Tux, logoen til Linux.

Problemet med unix-operativsystemet var at det var eid av et privat selskap, noe som gjorde at hvem som helst kunne ikke bidra til utvikling. På den tiden var det flere store aktører som jobbet med sine egne versjoner av en unix-liknende kjerne. Men kjernen vi kjenner som Linux, som ble den absolutt dominerende kjernen av den typen ble utviklet av en til da ukjent Finsk student som het Linus Torvalds. Logoen til Linux er en pingvin, og kan sees i figur 4.1.

Linuxkjernen ble utgitt av Linus som fri og åpen kildekode under GPL-lisensen, dette betyr at alle har rett til å kopiere, modifisere, og redistribuere kjernen slik de selv vil, hvor den eneste begrensningen er at all programvare som tar i bruk kildekoden må selv gi de samme rettighetene til andre. [28]

Linuxkjernen i seg selv er ikke er operativsystem, men er grunnsteinen som forskjellige operativsystemer kan bygges på, deriblant Ubuntu. Når APT-pakkebehandleren

brukes i Ubuntu, kan unix-filosofien ses i aksjon. Ved større programmer vil det være haugevis med såkalte *dependencies*, dette er andre program som det installerte programmet tar bruk av, istedenfor at det store programmet skal lage ting som allerede er kodet fra før, eller at den samme koden blir kopiert flere plasser. Når flere program bruker den samme dependency-en, vil den ikke bli installert flere ganger.

Beaglebone Black har spesifikke linuxdistribusjoner laget for å passe med seg selv, såkalte *plateavbildningsfiler*.

4.1.2 Teori C++

Programmeringsspråket som ble valgt var C++, som er et multiparadigmespråk med et statisk typesystem. Det kombinerer kontroll på lavt nivå, samt innehar abstraksjoner for å jobbe på høyt nivå, og er en av de mest populære programmeringsspråkene gjennom tidene. Det er én hovedgrunn til at vi velger C++ over Python, og det er at C++ er et mye raskere språk enn Python. I veldig mange tilfeller har ikke hurtighet så mye å si, noe som er grunnen til at Python har klart å blitt så populært, men i dette tilfellet hvor en robot skal reguleres og det tar bare en brøkdel av et sekund før den kan havne i bakken, så er hurtighet en veldig viktig del. Hvor mye kjappere C++ er i forhold til Python kommer an på hvilken test som er gjort, men til eksempel ble en såkalt **N-legemesimulering** kjørt på en PocketBeagle hvor C++ var 57 ganger kjappere enn Python. [29]

Det er flere grunner til at C++ er kjappere en Python, den største grunnen er at Python har et dynamisk typesystem. Dette betyr at datatypen til en variabel kan forandres når som helst, noe som gjør at når programmet kjøres må Python hele tiden sjekke typen til en variabel, slik den tilhørende dataen kan tolkes riktig. En annen grunn er at C++ er et kompilert språk, hvor Python benytter seg av en kommandotolk.[30] For C++ betyr det at en binærfil med alle maskinkodeinstruksene blir generert, som man da kjører. Python derimot, har et eget program som leser av hver linje i tur og lager maskinkode på sparket, noe som tar mer tid. Disse to er på engelsk kjent som henholdsvis *compiled* og *interpreted* programmeringsspråk.[31]

Sett bort fra hurtigheten er det også mange andre positive sider med C++. Det er et gammelt språk og er helt i toppen av de mest populære programmeringsspråkene i verden, det har også støtte for objekt-orientert programmering, noe som faktisk var hovedårsaken til at C++ oppsto som et språk, da den var først kjent som *C with classes*.[32]

Det er dog en del ulemper i forhold til Python. Det å skrive kode i C++ tar mye lengre tid, det er heller ingen avfallshenter, det vil si at eventuelle pekere må fjernes manuelt, noe som kan føre til minnelekkasjer og minnesegmentsfeil. C++ er også et mye større språk, det har blitt kritisert for å ha for mange forskjellige måter å gjøre en og samme ting på, slik det blir vanskeligere for forskjellige utviklere til å samarbeide med hverandre.

4.1.3 Metode og resultat Linux og C++

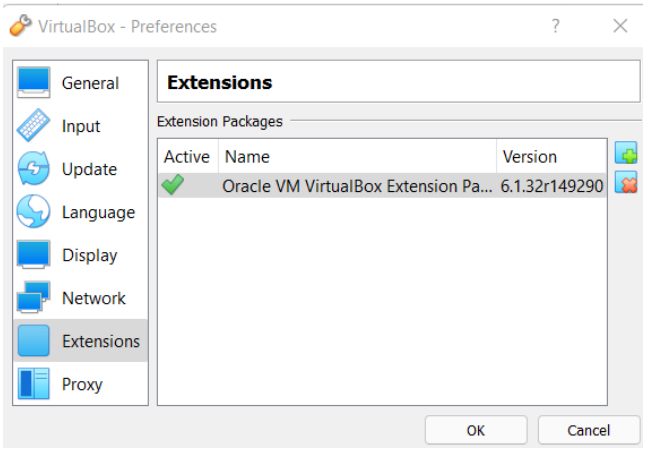
Den forrige bachelor-gruppen brukte en plateavbildningsfil basert på Ubuntu 18.04, men lenken de brukte for å laste ned denne var ikke lenger tilgjengelig, og alle spor etter en alternativ lenke til å laste ned var forsvunnet. Ubuntu 20.04 ble derfor benyttet istedenfor.

Når BeagleBone Black som ble bestilt kom, ble første steg i prosessen å flashe Ubuntu 20.04 på den. I følge brukermanualen til tidligere gruppe, og diverse brukerveiledninger på nett skulle det kun være å lage et oppstartbart minnekort med nedlastet plateavbildningsfil og holde inne noen knapper på BeagleBone. Uavhengig av hvor mange ganger dette ble gjort ville ikke flasheprosessen starte. Det ble også forsøkt med flere minnekort, uten at dette løste noe problem. Det viste seg at i plateavbildningsfilen var det en linje med kode som det måtte fjernes kommentering på. Da en på gruppen hadde en SD kort leser til USB-C ble denne benyttet for å fjerne kommentaren. Denne prosessen må også gjøres på en Linux PC.

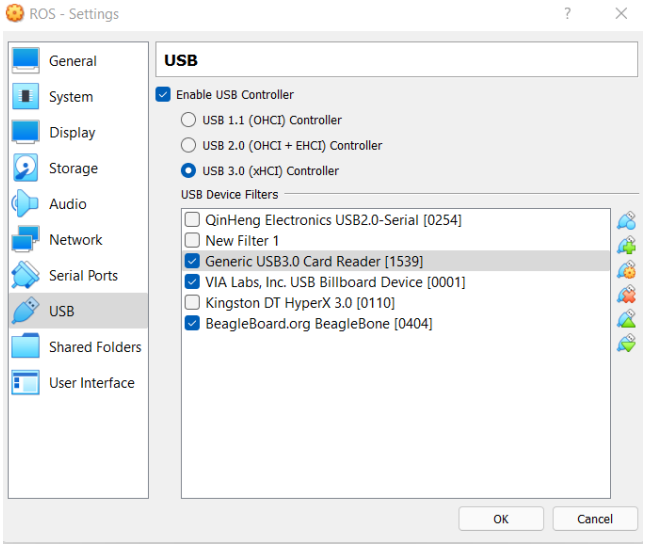
1. Lag oppstartbar minnebrikke med BalenaEcther, Rufus eller lignende.
2. Installer Oracle VM på din Windows Maskin. (Dette steget til og med steg 7 kan hoppes over om minnekort kan plugges i maskin med Linux)
3. Last ned din foretrekte Linux distribusjon og lag en virtuell maskin i Oracle VM med den plateavbildningsfilen.
4. Last ned "Oracle VM VirtualBox Extension Pack" og legg den i "file →preferences →extensions". Etter dette punktet skal vinduet se ut som figur 4.2a
5. For din virtuelle maskin, velg "settings →USB"
6. Huk av "Enable USB Controller", og velg "USB 3.0 (xHCI) Controller"
7. Trykk på "USB pluss" ikonet og velg enheten og/eller overgangen du vil benytte deg av på din VM. Figur 4.2b viser USB enheter som videresendes til den virtuelle maskinen.
8. Åpne filutforsker på din Linux PC. Minnekortet skal ha navn "rootfs"
9. Naviger deg til rootfs/boot og åpne "uEnv.txt"
10. Fjern # tegn på linje 57 slik at "#cmdline=init=/usr/sbin/init-beagle-flasher" endres til "cmdline=init=/usr/sbin/init-beagle-flasher"
11. Lagre endringene og løs ut minnebrikke. Nå vil minnebrikken flashe eMMC på BeagleBone

Når flashingsprosedyren er ferdig, opprettes kontakt med BeagleBone Black med bruk av SSH. SSH er en protokoll som gir deg kryptert tilgang til kommandolinjen til andre maskiner, som er spesielt nyttig når det kommer til "hodeløse maskiner" som BeagleBone Black. Hodeløs betyr her at det ikke er tilgang på skjerm og tastatur direkte til BeagleBone Black. Linux er kjent for at man kan gjøre så og si alt fra kommandolinjen, slike operasjoner er veldig effektive, men det er tidkrevende å bli komfortabel med slike funksjonaliteter.

Internettilgang til BeagleBone Black var nødvendig for å kunne installere ROS, og også nyttig til å laste ned diverse programmer fra Ubuntu sin pakkebehandler APT. Det skulle være mulig å få internettilgang gjennom USB-kabelen vi brukte til SSH-kommunikasjon, men ettersom dette krever noe konfigurering og vi hadde



(a) Bilde av benyttet tillegg



(b) Bilde av USB enheter som kan benyttes i VM

Figur 4.2: VM- innstillinger.

en ethernetkabel tilgjengelig ble den brukt istedenfor. To programmer vi kan anbefale er *fzf* og *tmux*. Fzf er en *Fuzzy Finder* som lar deg finne hvilken som helst fil eller mappe øyeblikkelig ved søk, veldig nyttig til tider. Tmux er en terminal multiplekser som gir deg tilgang til mange forskjellige terminaler I samme vindu. Man kan også frikoble en prosess, det vil si at vinduet ikke vises fram lenger men prosessen kjører fortsatt i bakgrunnen. Dette er fint for prosesser som skal gå i bakgrunnen hele tiden men man ikke trenger å ha oppe, som for eksempel roscore.

Koden som forrige gruppe skrev var også gjort i C++, men denne koden kunne derimot ikke så lett brukes, ettersom den var skrevet for spesifikk maskinvare, noe vi hadde byttet ut. Der kom også en del problemer. IMU-en som var bestilt

var en DFRobot_BMX160, som har egne biblioteker til Arduino og Raspberry Pi, ettersom disse ikke ble brukt kunne ikke bibliotekene direkte bli tatt nytte av. En arduino var tilgjengelig, så en gitt eksempelkode gitt for arduino ble prøvekjørt. Med den ble målingene akkurat slik de skulle, men å gjøre det på BeagleBone Black ble en utfordring. Sensoren bruker i2C-tilkobling, og fra kommandolinjen ble dermed kommandoen *i2cdetect* benyttet for å detektere sensoren. Herfra kjørtes kommandoen *i2cdump* for å lese av verdiene til alle registrene. Databladet til BMX160 viser hva alle de forskjellige verdiene skal bety, på den kan man se at registrene for akselerasjon, gyro og magnetometer er fra 0x04 til 0x17, men før man begynner å lese av verdier skal sensorene settes i *normal power mode* og ikke i *low power mode*. For å sette akselerometer, gyroskop og magnetometer til *normal power mode* må kommandoen **i2cset** brukes på 0x7e adressen med henholdsvis verdiene 0x11, 0x15, 0x18 slik som beskrevet på side 90 i databladet til BMX160.

Dessverre var fortsatt ikke de forventede verdiene fra IMU-ene til stede etter sistnevnte kommandoer ble gjort. Det ble deretter forsøkt å bruke arduinobiblioteket direkte, ved å forandre filnavnet på arduinofilen fra .ino til .cpp, og å kompilere koden på BeagleBone Black. Alle headerfilene til biblioteket måtte inn, noe som var krevende ettersom det var mange nestede headerfiler. Når koden skulle kompileres måtte I-flagget brukes for å peke på mappene til hvor alle headerfilene ligger. Da alle headerfilene ble inkludert kom vi over ett nytt problem, som var at det ble navnekonflikter av variabler på mange av de forskjellige headerfilene. Hvorfor det ble med navnekonflikter når headerfilene ble importert manuelt men ikke når arduinofilen kjørte ble ikke funnet ut av.

Ett forsøk til ble gjort med Raspberry Pi biblioteket, hvor det ble forsøkt å gå inn i de importerte bibliotekene der for å finne ut hva som blir gjort i bakgrunnen for å hente data fra IMU-ene, det lyktes heller ikke, på grunn av en blanding av at den objektorienterte formen til biblioteket, og manglende importfiler.

4.1.4 Drøfting Linux og C++

Linux er det åpenbare valget til innebygde ettkortsdatamaskiner. Dette ettersom slike systemer har gjerne begrensede ressurser og unike IO-muligheter, som bare et fritt og åpent operativsystem kan tilpasses. Valget av en tilpasset Ubuntu 20.04 fungerte til å laste ned ROS og slik som forventet, men det kunne være vanskelig å finne informasjon til bruk av IO-operasjoner som var kritisk for oss. En mer populær løsning til ettkortsdatamaskin er Raspberry PI og dets tilhørende operativsystem raspbian. I startfasen ble Raspberry Pi vurdert pga. støtte til ROS 2 og mer tilgjengelig informasjon og biblioteker.

Angående C++ er det enighet at det er det språket som passer oppgaven best, selv om steget hvor det kunne nyttes skikkelig ikke ble nådd. Hadde arduino blitt brukt kunne et bibliotek til sensoren vært tatt i bruk og det ville vært stor sannsynlighet at et forsøk på regulering kunne ha blitt oppnådd, ettersom med arduino fungerte kommunikasjonen rett "ut av boksenslik den skulle.

4.2 ROS, Rviz, Gazebo

Robotens styringskommunikasjon og signalprosessering foregår ved bruk av programvaren ROS. Det vil si at alle algoritmer har sin plass inne i ROS-filsystemet, eksempelvis motorstyring, sensoravlesninger, filtrering og regulering.

ROS brukes i prosjektet i kombinasjon med C++ og Linux, ettersom denne kombinasjonen anbefales for . Anbefalinger for andre kombinasjoner av OS og programmeringsspråk finnes ut fra hvilken type innebygd enhet man ønsker å jobbe med (RBPi el.l.).

Påfølgende teori tar utgangspunkt i ROS-versjonen som brukes våren 2022, *ROS Noetic Ninjemys*. Versjonen *ROS Melodic Morenia* som tidligere B.Sc.-grupper har benyttet, har ikke støtte mye lengre fram i tid, og ulike ROS- versjoner er ikke alltid kompatible med en og samme linuxversjon.

Først introduseres litt overordnet om grunnleggende elementer i ROS og samspillet mellom de, etterfulgt av noe mer utfyllende beskrivelser. Sist i teoridelen beskrives ROS-verktøyet *Rviz* og ROS-simulator *Gazebo*, som begge relateres til prosjektet.

4.2.1 Teori ROS

ROS er et sett med programvare-rammeverk som inneholder biblioteker og verktøy for å hjelpe programvare-utviklere med å lage robotapplikasjoner. ROS er en forkortelse for **Robot Operating System**, og er lisensiert under åpen kildekode. Grunnleggende informasjon, installeringsinstruksjoner og opplæringseksempler finnes på nettsiden wiki.ros.org. En rask oversikt gis også på wikipedia. [33] [34]

Ulike protokoller benyttes i ROS, hvor **services** er en synkron RPC-kommunikasjon, **topics** er asynkron dataoverføring og **parameter server** lagrer data.[35] Konseptnivåene i ROS er *Filesystem*, *Computation Graph* og *Community*, og i tillegg defineres to typer **namespaces** *Package Resource* og *Graph Resources*.

Filesystem

Under Filesystem ligger Packages, Metapackages, Package Manifests, Repositories, Message (msg) types og Service (srv) types.

Packages er hovedenheten for organisering av programvare i ROS. Den kan inneholde *noder*, *ROS-avhengige biblioteker*, *datasett*, *konfigurasjonsfiler*, og annet som er praktisk å organisere sammen. Packages er den minste konstruksjonen man kan opprette i ROS.

Metapackages er spesialiserte packages som kun skal representere en gruppe av andre packages som har en relasjon til hverandre. Oftest er metapackages benyttet som bakoverkompatible plassholdere for konverterte rosbuild-stakker. Rosbuild er erstattet med *catkin* i nyere ROS-versjoner, og er *byggesystemet* for pakkene.

Package Manifests er .xml- filer som gir metadata om en pakke. Filen inneholder *navn*, *versjon*, *beskrivelse*, *lisensinformasjon*, *avhengigheter*, og annen metadata

som eksempelvis eksporterte pakker.

Repositories en samling av pakker som deler felles VCS.

Message (msg) types er beskrivelse av beskjeder som sendes ved bruk av *messages* i ROS, og definerer beskjedenes datastruktur.

Service (srv) types er beskrivelse og definisjon av forespørsel/respons- datastrukturene når *services* brukes i ROS.

Computation Graph

Dette nivået består av et peer-to-peer nettverk med ROS-prosesser som kjøres samtidig. De grunnleggende Computation Graph konseptene i ROS er *nodes*, *Master*, *Parameter Server*, *messages*, *services*, *topics* og *bags*, som alle gir data til nettverket på ulike måter.

Nodes er prosesser som utfører beregninger. ROS er designet til å være modulær i en fininndelt skala, og et robotkontroll-system er vanligvis sammensatt av mange noder. Hver node skrives ved bruk av et ROS *client library*, eksempelvis *roscpp* eller *rospy*

Master gir navneregister og -oppslag til resten av Computation Graph. Uten masteren vil ikke nodene være i stand til å verken finne hverandre, utveksle beskjeder eller sette i gang tjenester. Den kjøres vanligvis ved bruk av *roscore*-kommandoen, som laster ROS-masteren sammen med andre essensielle komponenter. Etter at masteren har satt individuelle noder i stand til å lokalisere hverandre kjører disse videre peer-to-peer utenfor masteren.

Parameter Server tillater sentral datalagring. Serveren kjører på innsiden av ROS-masteren og er en delt multi-varierende *dictionary*, som er tilgjengelig via nettverks-APIer. Noder benytter denne serveren til å lagre og hente ut parametere ved kjøring. Den er best egnet til statiske ikke-binære data slik som konfigurasjonsparametere, ettersom den ikke er designet for høy ytelse. For at dataverktøy enkelt kan inspisere konfigurasjonstilstander og modifisere disse om nødvendig, er parameterserveren globalt synlig.

Parametere gis navn ved å bruke vanlig navnekonvensjon, som innebærer at ROS-parameterne har et hierarki som passer med *namespaces* som benyttes for *topics* og *nodes*. Parameter datatyper er *32-bit integer*, *bool*, *string*, *double*, *iso8601-datoer*, *lister* og *64-basekodede binærdata*.

Navnekonvensjonen refererer til *~name* som et *privat* navn. Disse er primært for parametere som er spesifikke for én enkelt node. Parameterverktøy som lar parametere endres og settes med kommandolinjen *rosparam* er *set*, *get*, *load*, *dump*, *delete* og *list*.

Klientbiblioteker som støttes er *rospy*, *roscpp* og *roslisp*.

Messages er måten noder kommuniserer på ved å sende beskjeder til hverandre. *Message* er en datastruktur hvor standard datatyper og arrays støttes, i tillegg til at beskjedene kan inneholde vilkårlig nøstede strukturer og arrays. *Messages* routes via et transportsystem oppbygd med publisering og abonnering. En node sender ut en *message* ved å publisere den under et gitt *topic*.

Topics er et *navn* som brukes for å identifisere innholdet i en *message*. En node som er satt opp til å omhandle en bestemt type data vil abonnere på et gitt *topic*. Det kan være flere samtidige publiseringsnoder og abonneringsnoder for et og samme *topic*. Generelt er ikke publiseringsnodene og abonneringsnodene klar over hverandres eksistens. Hensikten med det er å frigjøre produksjonen av informasjon fra bruken av den. Hvilken som helst publiseringsnode og abonneringsnode kan kommunisere med respektive *topic* når som helst.

Services er, i motsetning til topics, kommunikasjon basert på forespørsel og respons. *Services* defineres av en beskjedstruktur med et bestemt par av beskjeder, én beskjed for forespørsel og én for respons. En services-node tilbyr denne tjenesten under et gitt navn, og en klient benytter denne tjenesten ved å sende forespørsel og avvente respons. ROS-klientbiblioteket, eksempelvis roscpp, viser dette som et funksjonskall (RPC).

Bags er en viktig datalagringsmekanisme eksempelvis for sensordata, som kan være vanskelige å samle inn men nødvendige for utvikling og testing av algoritmer. Bagsformatet lagrer og spiller av igjen *ROS message data*.

Community Level

Konseptet med Community Level er at ulike ROS-ressurser gjør det mulig å fritt utveksle programvare og kunnskap under åpne lisenser. Disse ressursene er *Distributions*, *Repositories*, *The ROS Wiki*, *Bug Ticket System* og *Mailing Lists*.

Distributions er samlinger av versjonsbestemte stakker som kan installeres, på lignende måte som Linuxdistribusjoner. Dette gjør installeringen av en samling med programvare enklere, og vedvarende versjoner vedlikeholdes på tvers av et sett med programvare.

I dette prosjektet er **ROS Noetic Ninjemys** benyttet, som har LTS til mai 2025, til forskjell fra ROS Melodic Morenia fra tidligere B.Sc.-oppgaver som har EOL i mai 2023. Nye ROS-utgivelser er ikke nødvendigvis kompatible med *tidligere* ROS-utgivelser. Figur 4.3 viser logoen til ROS Noetic Ninjemys.

ROS lanserer en ny versjon i mai hvert år, som følger Ubuntu LTS-versjonene(Linux).

ROS2 derimot, gir ut en ny versjon hver sjette måned, som støttes kun ett år om gangen enn så lenge.

Figur 4.3: ROS Noetic Ninjemys' logo.

ROS2 er en større revisjon av ROS API, som vil kunne dra fordel av moderne biblioteker og teknologier for kjernefunksjonene i ROS. Støtte for både sanntidskode og innebygd system maskinvare legges også til.

Både Repositories, The ROS Wiki og Mailing Lists oppdateres jevnlig med nytt om ROS, og her finnes også fora for spørsmål og svar. I Bug Ticket System kan problemer og forespørsler angående ROS og ROS-relatert programvare meldes inn.

Graph Resource Names

Navnene i Graph Resource gir en hierarkisk struktur som benyttes for alle *ressursene* slik som *nodes*, *parameters*, *topics* og *services*, i en ROS Computation Graph. Disse navnene er avgjørende for hvordan mer kompliserte systemer komponeres i ROS.

Graph Resource Names er viktig for innkapsling innad i ROS. Hver av ressursene (noder, parametere, emner og tjenester) defineres innenfor en *namespace*, som den kan dele med andre ressurser. Generelt kan ressurser lage andre ressurser innenfor sitt navneområde, og de har tilgang til ressurser innenfor eller over sitt eget navneområde. Koblinger kan lages mellom ressurser i bestemte navneområder, men gjøres oftest ved å integrere de i en kode, på et nivå *over* begge navneområdene.

Visualisering av dette omhandles i avsnitt 4.2.6, se figur 4.9, som handler om rqt_graph-verktøyet i ROS.

De fire ulike typene Graph Resource Names innenfor ROS er *base*, *relative*, *global* og *private*, og har hver sin syntax.

Package Resource Names

For å forenkle prosessen med å referere til filer og datatyper benyttes Package Resource Names innenfor Filesystem-Level- konseptet. Disse navnene er navnet på den *Package* som ressursen er inne i, i tillegg til ressursens navn. Filer som refereres til ved å bruke Package Resource Names inkluderer *Message (msg) types*, *Service (srv) types* og *Node types*. [36] [37] [38] [39] [34]

4.2.2 ROS-konsepter på et høyere nivå

Kjernen i ROS-plattformen gir mange ulike måter å kommunisere data på, men setter ingen føringer på hvordan de brukes eller hvordan de er navngitt. Dette gjør at ROS enkelt kan integreres med en mengde ulike arkitekturer, men for å bygge større systemer over ROS er det nødvendig med flere konsepter på et høyere nivå. Her finnes i tillegg til stakkene *common_msg* og *geometry* også *plugins*, *filters* og *robot model*.

common_msg inneholder *messages* som stadig brukes av andre ROS pakker. Disse er beskjeder for *actions*, *diagnostics*, *geometric primitives*, *robot navigation* og *common sensors*.

geometry inneholder matematikkpakken *angles*, og geometripakkene *bullet*, *kdl*, *tf2* og *tf_conversions*.

Her er eksempelvis biblioteket tf2 relevant for bachelorprosjektet. Tf2 gir en standard måte for å følge med på koordinatrammer og transformere data innenfor et helt system, som for en robot ofte bygger på flere ulike koordinatrammer samtidig, uten at man er avhengig av å vite alt om alle koordinatrammene. Dette lar brukeren av biblioteket transformere punkter og vektorer mellom hvilke som helst av ulike koordinatrammer i systemet, til hvilket som helst tidspunkt.[40]

Plugins

Pluginlib er et bibliotek som dynamisk laster biblioteker i C++-kode, slik at man kan utvide og modifisere applikasjonens oppførsel uten at man trenger applikasjonens kildekode. Pluginlib laster slik både opp og ned plugins fra innsiden av en ROS-pakke, uten at applikasjonen er klar over biblioteket eller header-fila som inneholder class- definisjonene.

Filters

Filters-pakken er nært knyttet til ovennevnte pluginlib, og inneholder et C++-bibliotek for å prosessere data. Filtre konfigureres fra parametere på Parameter Server. Implementerte filtre per nå i ROS er *MeanFilter*, *MedianFilter*, *IncrementFilter* og *TransferFunctionFilter*.

Robot Model

URDF-pakken definerer en .xml-fil for å fremstille en robotmodell. Pakken inneholder en C++ -parser. Mer om URDF og 3D-modellering av roboten finnes i avsnitt 4.3.5 *Simulering av roboten, Gazebo*, og 4.4 *Modellering av robot i Fusion 360*. [41]

ROS client library

ROS klientbibliotek er en samling av kode som letter jobben med å programmere i ROS. Disse tar mange av ROS-konseptene og gjør de tilgjengelige via ferdigstilt kode. Bibliotekene tillater at det skrives ROS-noder, at det publiseres og abonneres til et topic, skrives og calles services, og at Parameter Server brukes. Slike biblioteker kan implementeres i ethvert programmeringsspråk, men per nå jobbes det primært med støtte for C++ og Python.

Det mest brukte klientbiblioteket er **Roscpp**, og er designet for å være et høyytelsesbibliotek for ROS. [42]

For å skrive egne klientbiblioteker eller integrere andre systemer med ROS kreves mer kunnskap på detaljnivå om implementeringen av ROS. Med noen av de grunnleggende konseptene i ROS nå forklart, slik som arkitekturen i Computation Graph som beskrevet tidligere inkludert rollene til ROS Masteren og nodene, sees det dermed videre på noen av detaljene.

Teknisk oversikt

Master

Masteren implementeres via XMLRCP som er en tilstandsløs HTTP-basert protokoll. XMLRCP er tilgjengelig i mange ulike programmeringsspråk slik at det er enkelt å interagere med masteren. Den har registrerings-APIer som tillater noder å registreres seg som *publisher*, *subscriber* og *service provider*. I variabelen ROS_MASTER_URI er masterens URI lagret, og denne korresponderer med vertsporten på XMLRPC-serveren den kjører. Default port for masteren er 11311.

Parameter Serveren er som nevnt tidligere en del av ROS Masteren, og dermed er også Parameter Server APIen implementert via XMLRPC. Bruken av XMLRPC muliggjør smidig integrering med ROS klientbibliotekene og gir stor fleksibilitet

når serveren lagrer og henter data. Har man tilgang til et XMLRPC klientbibliotek kan kall til Parameter Server gjøres direkte uten å benytte et ROS klientbibliotek, ettersom XMLRPC APIene gjør det enkelt å integrere disse kallene.

Node

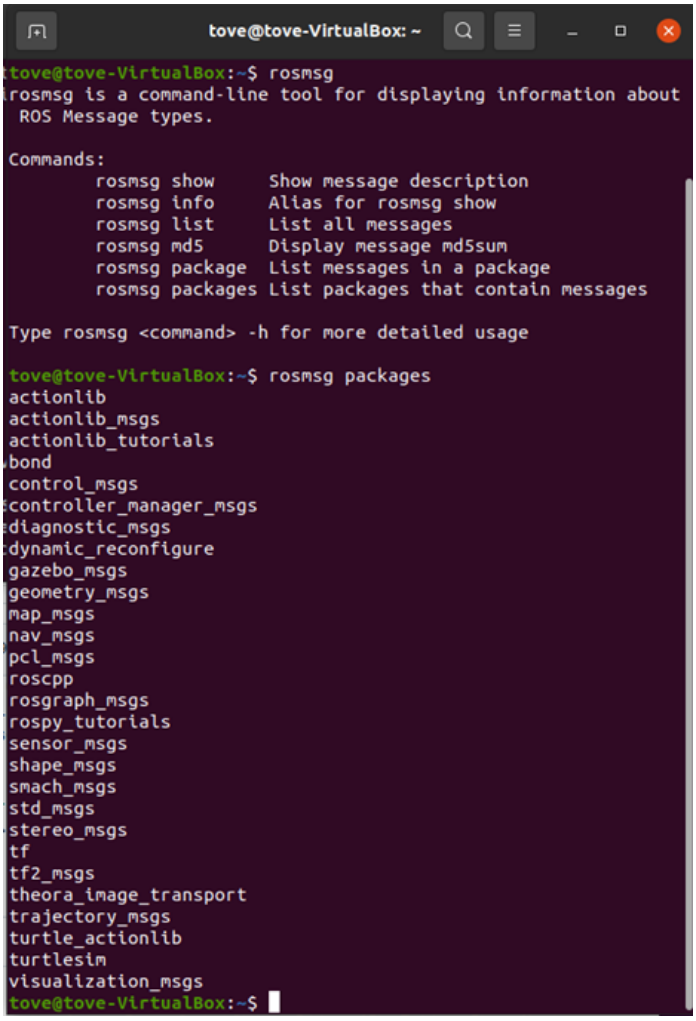
En ROS-node har flere APIer. *Slave-APIen* mottar callback fra Master, og forhandler koblinger med andre noder. *Transportprotokoll (TCPROS)*, hvor noder etablerer kobling mellom topics ved å bruke en felles protokoll. *Kommandolinje-API* som muliggjør at navn innenfor en node kan bli konfigurert under kjøring, ved at hver node støtter kommandolinje remapping-argumenter. Hver node har også en URI som korresponderer til vertsporten på XMLRPC- serveren den kjører. Denne serveren brukes kun til å opprette koblinger med andre noder og kommunisere med masteren, den benyttes ikke til overføring av topics- eller servicedata.

TCP benyttes ofte i forhold til UDP ettersom det gir en pålitelig og stabil kommunikasjonsflyt hvor datapakkene alltid ankommer i riktig rekkefølge, og resendes helt til de ankommer. Ved dårlig WiFi eller modemtilkobling kan denne resendingen opptre som feil, mens dette er uproblematisk for kablet ethernet. UDP kan i slike tilfeller være mer egnet og effektivt.

Noder kommuniserer altså direkte med hverandre, over en egnet overføringsmekanisme. Data routes ikke via masteren. [43]

Som nevnt tidligere under Package Resource Names finner vi Message(msg) *types* og Service(srv) *types*. For å beskrive verdiene på dataene som ROS nodene publiserer bruker et forenklet *message description language*, nemlig **messages**. Denne beskrivelsen gjør det enkelt for ROS-verktøyene å generere kildekode for gitt type beskjed, som kan benyttes i mange mottaker-kodespråk.

Beskrivelsen av beskjeden lagres i .msg-filer i undermappen *msg/* tilhørende en ROS-pakke. En slik .msg-fil har to deler, *fields* og *constants*. Feltene er dataene som



Figur 4.4: Kommando *rosmsgs packages* i ROS.

er sendt i respektive beskjed. Konstantene definerer nyttige verdier som kan brukes til å tolke de feltene. Beskjed-*type* refereres til ved å benytte Package Resource navnene. Kommandolinjen *rosmsg* brukes for å skrive ut informasjonen og kan finne kildefilene som bruker en gitt beskjed-type. Se eksempel i figur 4.4. [44]

Service-*typene* beskrives tilsvarende med et forenklet *services description language*, srv. Dette bygger direkte på ROS msg-formatet, for å muliggjøre kommunikasjonen mellom nodene ved bruk av en forespørsel-respons- protokoll. Beskrivelsene lagres i .srv-filer i en pakkes undermappe, srv/, og refereres til ved å bruke navnet fra Package Resource Names. Utskrift av beskrivelse, pakker som inneholder .srv-filer og kildefiler som benytter typen service, oppnås ved bruk av kommandolinjen *rossrv*, se figur 4.5 [45] Det finnes i tillegg en stakk som heter *common_msgs*, som inneholder messages som er svært utbredt brukt av andre ROS-pakker. Hensikten med stakken er å forhindre sirkulær avhengighet, ved å gi en delt avhengighet for de mest vanlige *messages* som benyttes mellom flere stakker. [46]

```
tove@tove-VirtualBox:~$ rossrv list
control_msgs/QueryCalibrationState
control_msgs/QueryTrajectoryState
control_toolbox/SetPidGains
controller_manager_msgs/ListControllerTypes
controller_manager_msgs/ListControllers
controller_manager_msgs/LoadController
controller_manager_msgs/ReloadControllerLibraries
controller_manager_msgs/SwitchController
controller_manager_msgs/UnloadController
diagnostic_msgs/AddDiagnostics
diagnostic_msgs/SelfTest
dynamic_reconfigure/Reconfigure
gazebo_msgs/ApplyBodyWrench
gazebo_msgs/ApplyJointEffort
gazebo_msgs/BodyRequest
gazebo_msgs/DeleteLight
```

Figur 4.5: Kommando *rossrv list* i ROS.

4.2.3 Dataverktøy i ROS

Ulike verktøy er nyttig til forskjellige prosjekter, her nevnes noen som er mer relevante enn andre for dette robotprosjektet.

Rviz er et dataverktøy organisert i pakker i ROS som andre algoritmer, og kan 3D-visualisere roboter og sensordata.

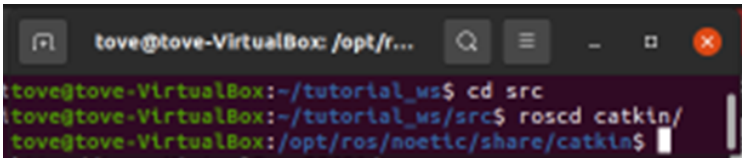
Rosbag er en kommandolinje som benytter filformatet bags som er nevnes i avsnittet om Computation Graph. Bags logger ROS-beskjeder ved å lytte til emner og lagre beskjeder når de ankommer emnet. Når bags frigir beskjeden igjen vil det være likt som om den originale noden produserte dataene til Computation Graph, som gjør at bags er et nyttig verktøy for å lagre data som skal brukes i utvikling på et senere tidspunkt. Rqt_bag gir en GUI om nødvendig, i utgangspunktet er rosbag kun et kommandolinje-verktøy.

Catkin er selve oppbygningssystemet for ROS, og erstatter tidligere *rosbuild*. Ettersom det baseres på CMake fungerer det på tvers av ulike plattformer, har åpen kildekode, og er uavhengig av programmeringsspråk.

Rosbash- pakken gir en rekke verktøy som augmenterer funksjonaliteten til Linux' terminalvindu, *bash shell*. Eksempler er rosls, roscd og roscp som har samme funksjonalitet som respektive ls, cd og cp. Dette muliggjør bruk av pakkenavn direkte istedenfor hele filplasseringsadressen for pakken. I tillegg fås tab-fullføring

når man skriver, *rosed* som gjør det mulig å redigere i valgte default teksteditor, og *rosrun* som kjører kjørbar kode.

Se eksempel i figur 4.6 hvor man kan se resultatet av cd(change directory) kontra roscd(ROS change directory). Her ser man at man flytter seg lengre i ett steg ved bruk av *roscd catkin* som går fra mappen *tutorial_ws/src til /opt/ros...*, mens i linjen over går det ikke å skrive *cd catkin* for catkin finnes ikke i det området man står i.



Figur 4.6: Linux-kommando *cd* vs. ROS- kommando *roscd*.

Roslaunch brukes for å sette i gang mange ROS-noder, og setter parameterne i ROS Parameter Server. Konfigurasjonsfilene i roslaunch skrives ved bruk av XML og kan automatisere en kompleks oppstart- og konfigurasjonsprosess inn i én enkelt kommando. Roslaunch-scripts kan inkludere andre roslaunch-scripts, starte opp noder på spesifikke maskiner, og restarte prosesser som eventuelt dør i løpet av kjøringen. [47]

4.2.4 Systemvisualisering rqt_graph og rqt_dep

For å se hvordan de ulike *nodes* og *topics* interagerer finnes det et verktøy som visualiserer dette kalt rqt_graph. Her fremstilles en oversikt over prosessene som kjører i ROS, og tilkoblingene deres. Ettersom dette er et grafisk verktøy kreves skjerm tilkoblet enheten som kjører ROS.

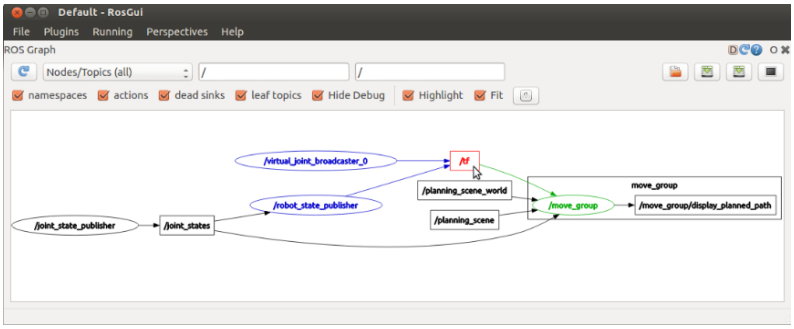
Videre kan også avhengighetene fremstilles grafisk på tilsvarende måte, ved bruk av rqt_dep. De ulike fargene i rqt_dep- figuren viser direkte avhengigheter i grønt i forhold til blokken som pekeren er på, og i blått den avhengigheten man søkte i kommandolinjen, her **rqt_dep sensor_msgs**, se figur 4.7b.

I figur 4.7a ser man nedtrekksmenyen hvor kan det velges i hvilke deler av systemet man vil vise. Her ses visualisering av *Nodes/topics(all)* hvor det videre kan krysses av like under nedtrekksmenyen om man vil vise namespaces, som er den første svarte ellipsen, og actions som er den grønne ellipsen. Ellers ser man her noder i blå ellipser, tillegg sees i røde rektangler og emner i sorte rektangler.

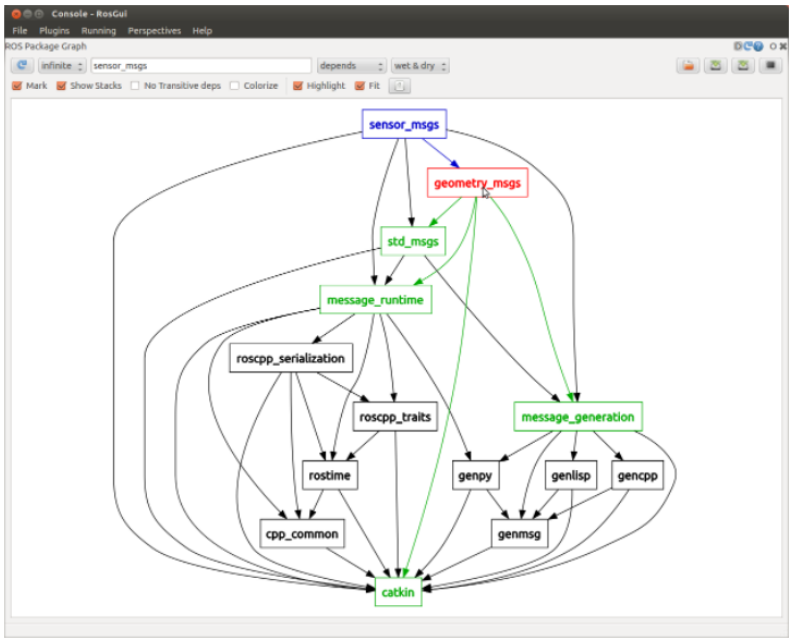
4.2.5 Simulering av roboten, Gazebo

Gazebo er en robotsimuleringsprogram, en dynamisk 3D-simulator, som ble utviklet i 2002. Eksisterende versjon er Gazebo 11.0 som ble lansert i januar 2020 og har EOL i januar 2025. ROS ble integrert i Gazebo i en periode fra 2009, men ROS har i dag ingen distribusjoner som inneholder Gazebo direkte.

Gazebo har åpen kildekode, og drives av OSRF. For å benytte URDF-filer fra vår 3D-modell kreves bruk av Gazebo-*plugins* for ROS Noetic. Siste versjon er som nevnt Gazebo 11.0 og fungerer best på Ubuntu.



(a) Eksempelbilde rqt_graph, hentet fra ROS.wiki



(b) Eksempelbilde rqt_dep, hentet fra ROS.wiki

Figur 4.7: Systemvisualisering i ROS.

Med tanke på krav til HW kommenteres det at enhetens GPU må være bra nok, og her sies at Nvidiakort fungerer bra med Ubuntu. CPU må tilsvare Intel i5 eller nyere. I tillegg kreves minst 500MB ledig plass på disk (dette inkluderer installering av Ubuntu).

En av fordelene med å bruke Gazebo med ROS er at vekslingen mellom den reelle robotmodellen og den simulerte gjøres enkelt. I Gazebo kan man logge simuleringen og spille den av igjen fra egne loggfiler. [48] [49] [50] [51] Gazebo drar nytte av *dependencies* (eksisterende biblioteker) for å utføre spesifikke oppgaver, og støtter noen *physics engines*. Eksempler på disse er **Ogre** og **Qt**, **ODE** og **Bullet**. Disse bidrar blant annet til god grafikk og dynamikk. Ettersom alt foregår under åpen kildekodelisens ligger kjernebibliotekene tilgjengelige på GitHub ved bruk av . Disse kjernebibliotekene er SDFormat, Ignition Math, Ignition Transport

og Ignition Messages. [52]

Simuleringsspesifikke tags må legges til URDF-filen for at den skal fungere tilstrekkelig. Det finnes en tutorial på nettsiden gazebosim.org som forklarer hvordan URDF konverteres til nødvendig SDF-fil. Grunnen til at konvertering er nødvendig ligger i at ROS ikke oppdateres i samme tempo som utviklingen innenfor robotikk. URDF er et standard format i ROS, mens SDF dekker opp manglene i URDF når det gjelder alt fra *world*-nivået til robot-nivået. Herunder posisjonene til roboten i en *world*, og andre egenskaper som parallell kobling, friksjon m.m.

SDF beskrives ved bruk av XML, og på gazebosim.org finnes beskrivelse av stegene i å få en URDF-robot til å fungere som ønsket i Gazebo. [53] Benyttes F360 til 3D-modellering kan man få ut SDF, som beskrevet i etterfølgende avsnitt *4.3 modellering av robot i Fusion 360*, og er ikke avhengig av å gå gjennom alle stegene som beskrevet over.

For å få ROS til å interagere med Gazebo må det som ønskes gjort i Gazebo dynamisk linkes til ROS- bibliotekene. Dette gjøres ved å spesifisere Gazebo-plugin i URDF-filen.[54]

Det finnes alternative simuleringsprogram, og et som gir mye og oversiktlig informasjon er MuJoCo. Fra oktober 2021 er MuJoCo antatt av DeepMind, og kommer videre til å ha åpen kildekode på GitHub. [55]

Historikk, oversikt og steg-for-steg fremgangsmåter for både 3D-modellering, visualisering og simulering finnes på nylig oppdaterte nettsider for MuJoCo's dokumentasjon. Her benyttes også URDF-filer, men de vil bli konvertert til MuJoCo's egen MJCF-filer ettersom de selv ser på URDF-filer som mer begrensede. [56]

Fra 2016 finnes en kommentar på hvor kompatibelt MuJoCo er mot Gazebo, og slik nyttig i ROS. Her sies at det i prinsippet er mulig å integrere Gazebo med MuJoCo, men at det frem til da ikke hadde vært forsøkt.

Siden 2016 har både MuJoCo og Gazebo utviklet seg og ligner mer på hverandre nå enn tidligere. I diskusjonsfora synes MuJoCo å være på oppadgående trend og Gazebo kommenteres å være på vei til å bli utdatert. I et tilsvar sies det at det finnes ROS-bridges for som gjør ulike simuleringsprogram kompatible for bruk med ROS. [57]

I en studie gjennomført i Tyskland publisert i 2021, tar de for seg en sammenligning av ulike simuleringsprogram. Her ser de på hastigheten til programmene for å undersøke kvaliteten som kreves for pålitelig simulering. Fire ulike simuleringsprogrammer blir undersøkt i studien som kjøres på flere ulike enheter for å sammenligne på ulike prosessorer. De fire ulike programmene er Gazebo, PyBullet, Webots og MuJoCo. Ved store tidsintervall i kjøringen kommer Gazebo dårligere ut enn de andre, hvor Webots har bra stabilitet opp mot tidsintervaller på 48ms. Ulempen med Webots er færre konfigurasjonsmetoder sammenlignet med andre. MuJoCo omtales som egnet for erfarne utviklere på grunn av eget programmeringsspråk og omfattende settings-muligheter. PyBullet sies å være enkelt å lære og enkelt i bruk. Gazebo omtales i kombinasjon med ROS som veletablert. Det finnes forskjeller på hvordan de fire løser parallelle prosesser og hvor mye minne det bruker.

Interessant i dette forsøket er at de bruker Ubuntu 20.04 og ROS Noetic Ninjemys, og oppgir både kjerne, Dockerversjon og Pythonversjon, og publiseringen er kun ett år gammel. Deres konklusjon er at valget av simuleringsprogram vil avhenge av hva man vil simulere, og de har sammenlignet to ulike scenarioer hvorav den ene var en robotarm. Samtidig sier de at alle de fire fungerer likt og bra, så lenge tidsintervallet ikke er for høyt. Er tidsintervallet høyt vil nøyaktigheten svekkes for alle programmene, men mer for Gazebo. I tillegg nevnes flere simuleringsprogram som ennå ikke er lansert, men like om hjørnet, og viser til flere studier i referanselisten, som sammenligner ulike simuleringsprogram i ulike settinger.

Utgangspunktet for artikkelen er å se på dette i forhold til komponentene, og forsøket kjøres derfor på ulike enheter som har ulike prosessorer. En vanlig måte å øke responsiviteten i simuleringen på for å få denne til å gå så raskt som mulig i forhold til sanntid, er å redusere tidsintervallet i kjøringen. Dette er negativt proporsjonalt med presisjonen i simuleringen. Her brukes en real-time factor RTF som tar forholdet mellom simuleringstid og reell tid. Høy RTF vil ha redusert presisjon som en konsekvens. De fire simuleringsprogrammene som sammenlignes er Gazebo, MuJoCo, PyBullet og Webots. Simuleringsprogrammer som anses som lovende og er under utvikling er Ignition, en etterfølger til Gazebo, Isaac Sim som støtter ROS API og kan være et mulig alternativ til Gazebo, og ytterligere nevnes Isaac Gym, og RaiSim. For de fire programmene i studien trekkes Gazebo og PyBullet frem med fordel om at det finnes mye informasjon og støtte og at de er vel etablerte. [58]

4.2.6 Resultat ROS, Rviz, Gazebo

I prosjektet vårt benytter vi oss av ROS, som står for robot operating system og er ett rammeverk for å utvikle programvaren som skal kontrollere roboter. I dag er det to ROS-versjoner ute, ROS 1 og ROS 2. Vi benytter oss av ROS 1 ettersom ROS 2 krever 64-bit arkitektur, men beaglebone har bare 32-bit.

ROS er en mellomvare som er laget for at forskjellige programvarerammeverk for robotutvikling skal kunne kommunisere med hverandre. Det er basert på fri og åpen kildekode, noe som er grunnen til at den er laget for Linux, ettersom den tar I bruk veldig mange pakker som er av fri og åpen kildekode. Designfilosofien bak ROS er at man lager et likemannsnettverk av prosesser som har lav koblingsgrad med hverandre. Prosesser kan være for eksempel en som leser inn sensorverdier og tolker de, eller noe som styrer en motor. Disse prosessene følger en abonner/publiser-modell. Dette vil si at de individuelle prosessene kan publisere en verdi, for eksempel en sensorprosess vil så klart publisere verdien den får fra sensoren, da kan andre prosesser som vil ta i bruk denne verdien kunne abonnere på sensorprosessen, slik at den får brukt sensorverdien internt i sin logikk. Det med at proseessene har lav koblingsgrad betyr at du kan forandre på den indre logikken til de forskjellige prosessene uten at det går utover ander prosesser, med mindre det går utover abonnering/publiserings-verdiene.

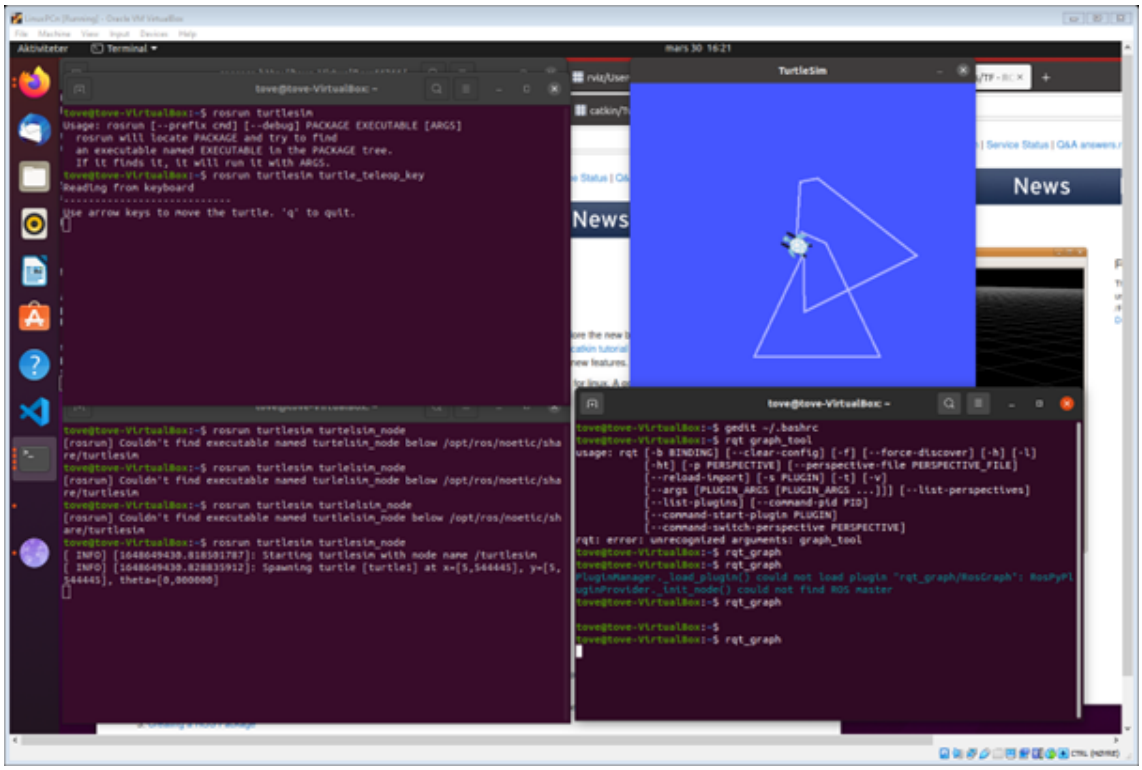
Etter vi hadde flashet BeagleBone Black med Ubuntu, kunne vi da laste ned

ROS direkte gjennom ethernetkabelen. Valget av ROS-distribusjon falt på Noetic ettersom dette var den eneste som støttet Ubuntu 20.04 med ROS 1, vi stilte deretter med en annen versjon enn fjorårets gruppe som brukte Melodic.

Under installasjonen av Noetic kunne man velge mellom tre versjoner. Desktop-full, Desktop, og ROS-Base. Ettersom BeagleBone Black har lite plass, og har lite bruk for tilleggsfunksjonene til de mer komplette installasjonene, ettersom de handler om visualisering som ikke er relevant i et hodeløst system.

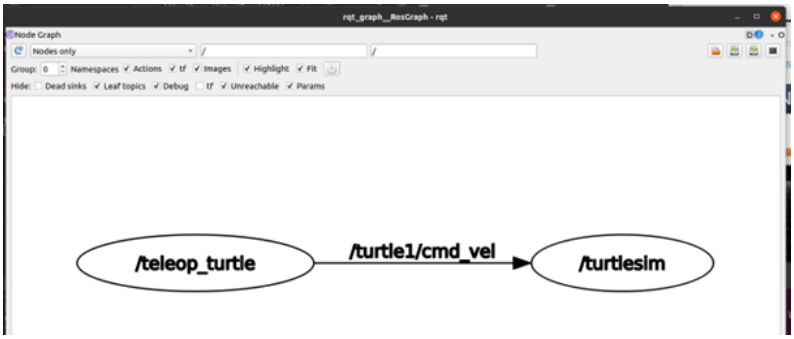
Under installasjonen gikk vi tom for plass på BeagleBone Black slik at vi ikke fikk installert hele ROS-Base, i etterkant kom vi fram til at vi sannsynligvis hadde takket ja til å installere mer enn nødvendig under første installasjon, ettersom det gikk fint senere. Før vi fant ut av det gikk vi inn i en blindvei ved å prøve å finne måter å utvide minnekapasiteten til BeagleBone Black på.

Utvidelse av minnekapasiteten var svært krevende, ettersom mesteparten av minnet ikke ble brukt i home-mappen, men hovedsakelig /usr/, en systemmappe hvor deler av selve operativsystemet ble brukt. Derfor var det ikke så enkelt å flytte denne mappen til et minnekort eller minnepenn, spesielt siden denne mappen blir brukt under oppstartsprosessen. Når det ikke gikk ble det også prøvd å kjøre hele operativsystemet fra en minnepenn koblet til BeagleBone Black, også uten hell. Dette problemet ble en blindvei ettersom vi senere fant ut det ikke var et problem å installere ROS når man er forsiktig med hvilke pakker man takker ja til å installere.

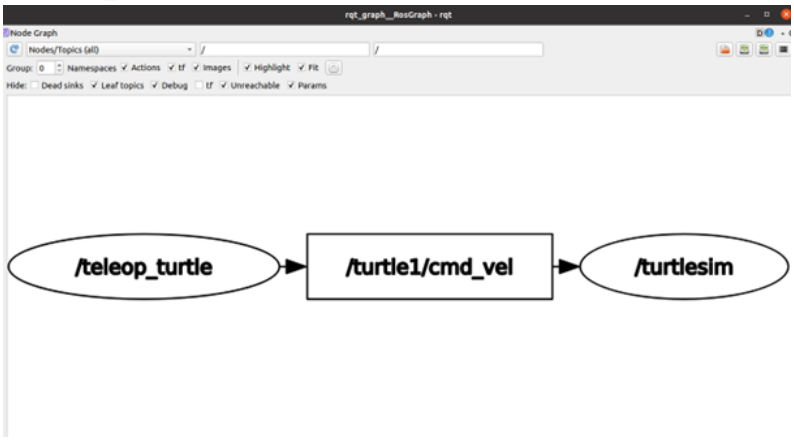


Figur 4.8: Illustrasjon over multiple terminaler i ROS.

I figur 4.8 fremstilles hvordan ROS kan øves på, med simulering av et objekts bevegelser og sporing av disse i testprogrammet Turtlesim. Her er det ulike terminaler som har hver sin funksjon, og man kan ha mange terminaler samtidig. Det blå vinduet viser grafisk de styringskommandoene som skrives i en av terminalene, ved at man har startet en terminal med en kommando som muliggjør styring av objektet i grafikken ved hjelp av piltaster. Her kan man velge å gi koordinater som objektet skal bevege seg til som kommando i en terminal, istedenfor å aktivere og bruke piltaster. Viktig er at den *første* terminalen **kun** benyttes til å starte ROS ved bruke av kommandoen **roscore**.



(a) Rqt_graph hvor det er valgt å kun vise tilkobling mellom noder.



(b) Rqt_graph for samme system, men her med alle koblinger valgt.

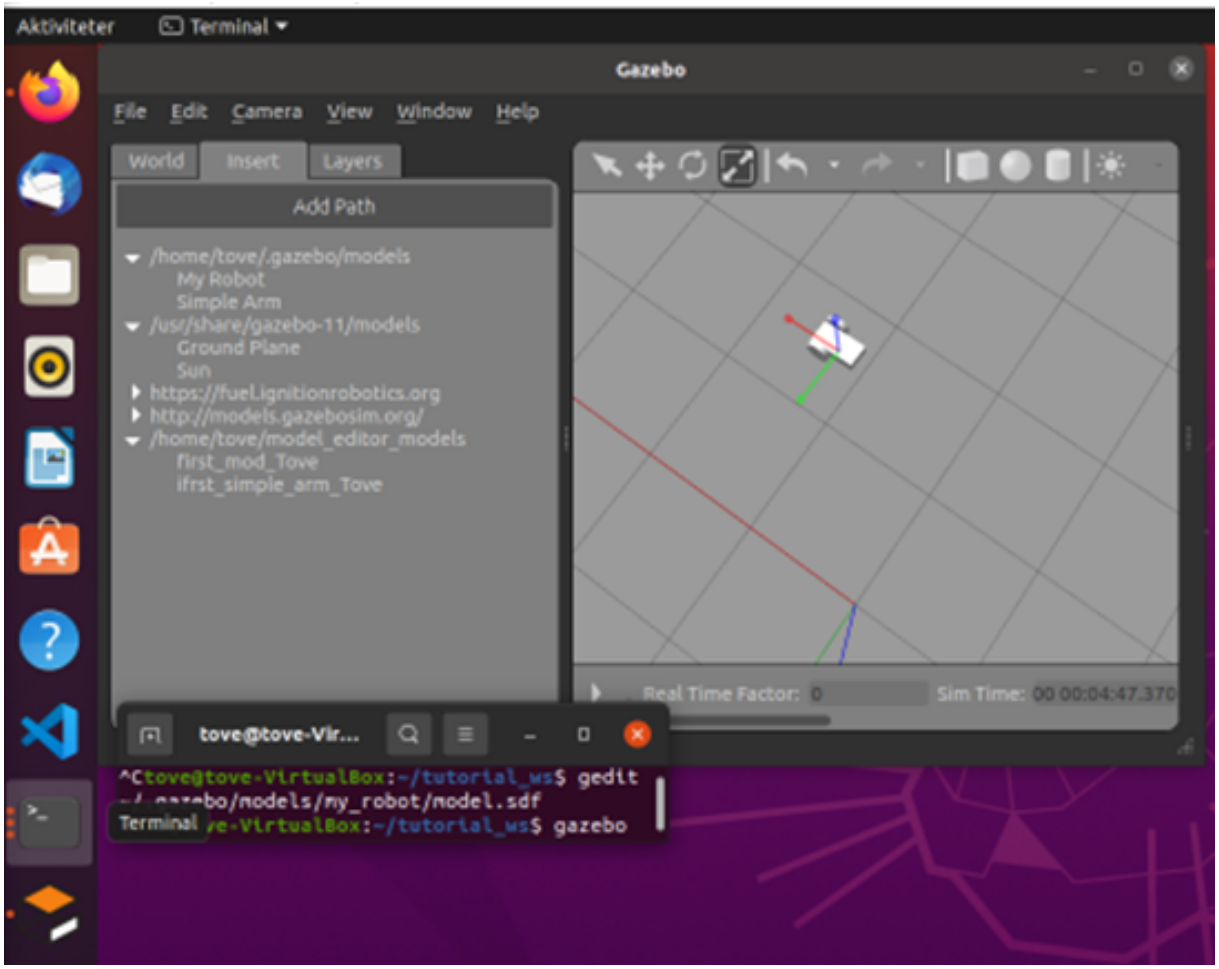
Figur 4.9: Ulike varianter av rqt_graph-visualisering fra test i Turtlesim.

Valgmenyene i rqt_graph gjør at man kan få vist den varianten av visualisering som gir mest informasjon etter eget behov. Fra testprogrammet Turtlesim ble det også testet noen av tilleggene i ROS. Figur 4.9a visualiserer kun tilkoblingen mellom noder, hvor topics er vist som informasjon langs signalveien. I figur 4.9b er det valgt å vise alle tilkoblinger, og da finnes topics inne i et rektangel og man ser forholdet mellom elementene på et annet vis.

Rviz

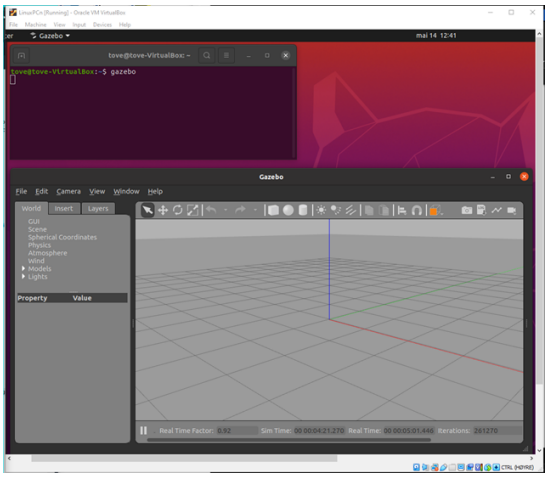
For å komme i gang med Rviz finnes en fremgangsmåte på ros.org, som tar seg alt fra installering , oppsett og kjøring av Rviz i Noetic ved bruk av kommandolinjer i terminal. Her beskrives hvordan GUI-en til Rviz er oppbygd og hva som kan konfigureres hvor av sensorer og tilsvarende elementer. [59]

Resultat Gazebo



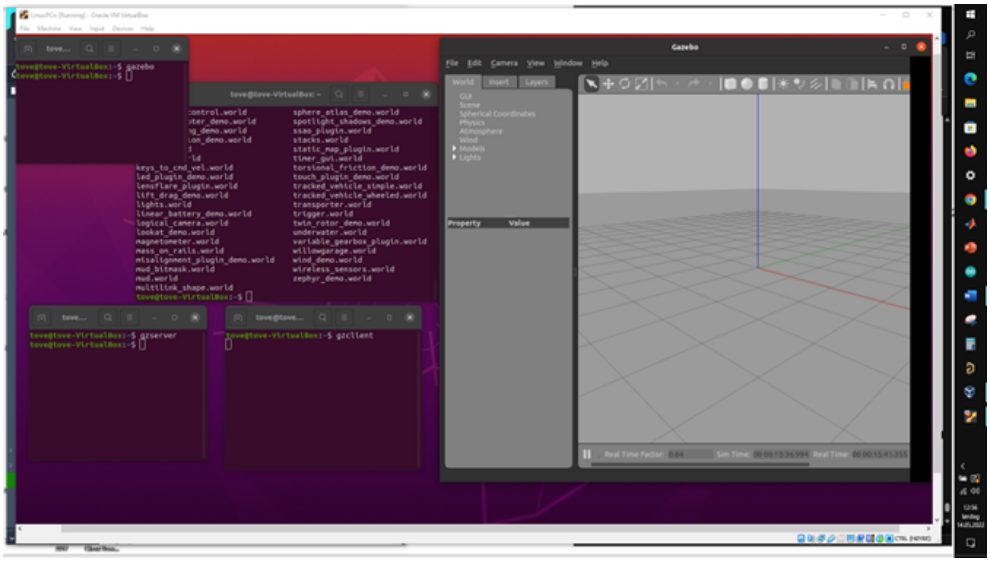
Figur 4.10: Robot i Gazebo, testmodell.

Installering av Gazebo gjøres ved å følge instruks på classic.gazebo.org, der finnes også fine introduksjonsprosjekter av ulike vanskelighetsgrader oversiktlig organisert. Her gis alt fra eksempler på modellering av robot direkte i Gazebo, til bruk av SDF-fil (se avsnittet om modellering av roboten) som vi har fra F360.



Figur 4.11: Gazebo11.0 installert på VM.

Gazebo11.0 i Ubuntu på en Linux VM sees i figur 4.11, hvor **Gazebo** skrives i kommandolinjen og trykker enter for å åpne GUI-en for Gazebo. Figur 4.12 viser en alternativ måte hvor det åpnes en *master* i én terminal og deretter en *client* hvor det skrives **gzclient** som dermed åpner Gazebo. Videre finnes det detaljerte forklaringer på alt av verktøy i panelene på samme side som nevnt over. Her er det også detaljforklaringer på hvordan man setter opp grensesnittet mellom ROS og Gazebo, med egne tutorials for både ROS, ROS2, hvilke kombinasjoner av ROS og Gazebo som kan kombineres. [60]

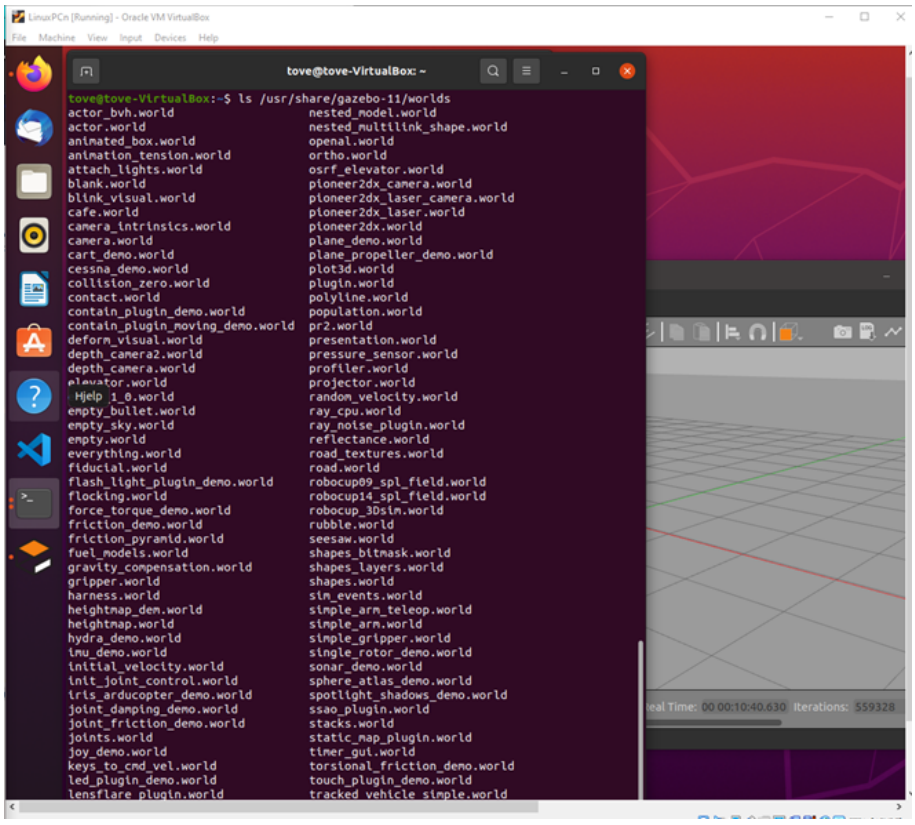


Figur 4.12: Oppstart av GUI Gazebo fra klient.

Kombinasjonen av ROS og Gazebo gir muligheten til å bruke kommandolinjer for å interagere med Gazebo. Eksempelvis i figur 4.13 ser man bruk av **ls** inne i mappen */usr/share/gazebo-11/worlds* som gir ut en liste over tilgjengelige *worlds* man kan bruke for miljøet rundt roboten man legger inn.

Modellering av roboten i F360 viste seg å kunne gi SDF som kan benyttes direkte i Gazebo, slik at konvertering fra URDF ved hjelp av plugin er egentlig overflødig når robot modelleres i F360.

Tutorials på classic.gazebosim.org er greie å følge. I figur 4.10 sees en liten robot med to hjul som er laget og testet i Gazebo på Linux VM ved å trinnvis redigere en SDF-fil fra ROS' kommandovindu, ved å følge en slik tutorial.

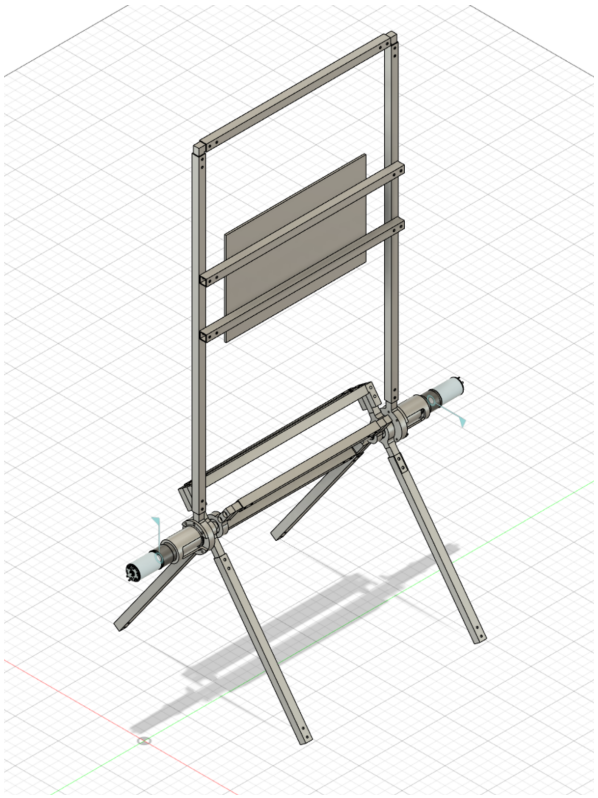


Figur 4.13: List of worlds i Gazebo.

Drøfting Gazebo

Selve prosessen med å få Gazebo til å kjøre fra ROS og legge inn en enkel modell lykkes greit. Det gjenstår derimot å få satt opp ROS-systemet rundt den nye 3D-modellen som er laget i dette bachelorprosjektet. MuJoCo lar både modellering og simulering skje i samme program, til sammenligning med ROS kombinert med Gazebo. Påfølgende avsnitt om modellering i Fusion360 viser at man kan ta inn modeller fra andre programmer også her. Dette viser at det finnes flere muligheter å gjennomføre en simulering på, for roboten.

4.3 Modellering av robot i Fusion 360



Figur 4.14: 3D-modell av roboten i Fusion 360.

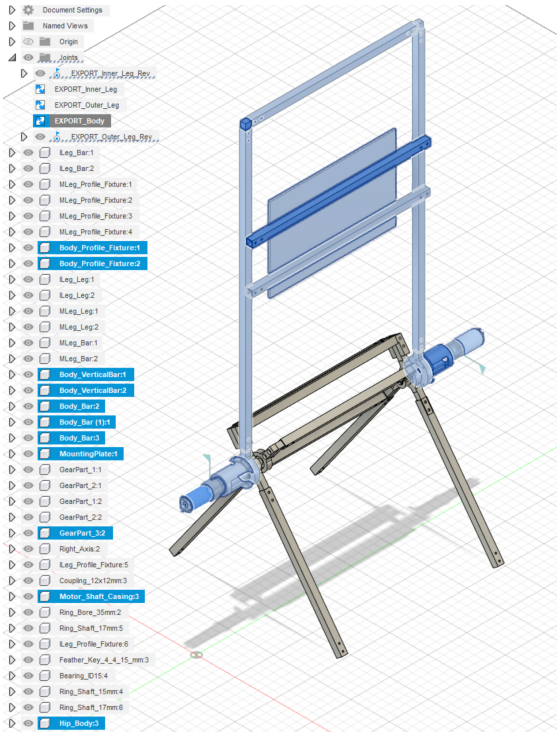
For å fortelle ROS hvordan roboten vår fysisk er og ser ut bruker vi en URDF Xacro fil. URDF står for Universal Robot Description Format og som navnet tilsier beskriver den roboten. I tillegg til å fortelle noe om de fysiske dimensjonene og hvilke ledd som henger sammen med hverandre, kan URDF filen også gi informasjon om materialet roboten består av og dermed også fortelle noe om vekt og tyngdepunkt til roboten. Ferdig 3D-modell sees i figur 4.14

4.3.1 Resultat modellering av robot i F360

For å lage en god modell som kan benyttes til simuleringer i Gazebo er det brukt Fusion360. Ved å eksportere roboten fra Fusion360 vil roboten beholde fysiske egenskaper som materiale, noe som gjør at fysiske egenskaper som vekt og tyngdepunkt blir riktig i Gazebo. Main Assembly filen som finnes fra før av er trolig lagd i SolidWorks og har ikke definert ledd og materiale. Ved importering av denne til Fusion360 var det noen deler som var plassert feil i forhold til hvor de skulle være, og motor/gir var manglende. Ved importering av CAD filer til motor og gir fra Maxon viser det seg at både motor og gir kun er enkle hele blokker. For å

lage en del som hadde mulighet for rotasjon ble gir delen delt opp i to deler slik at staget kan bevege seg uavhengig av girhuset. CAD filene manglet også monteringsmateriell for å feste motor og gir sammen. Det gir noen millimeter feil på størrelsen til motoren da monteringsmateriellet er noen millimeter tykt. Når det gjelder feilplassering av deler etter importering var for eksempel det indre beinparet flytende i løse luften og ikke inntil andre deler. Når en del ble flyttet på, bevegde den seg uavhengig av de andre delene den var festet til. Denne logikken fører til at Fusion360 behandler hver eneste del som en enkelt del uten bindinger til andre deler.

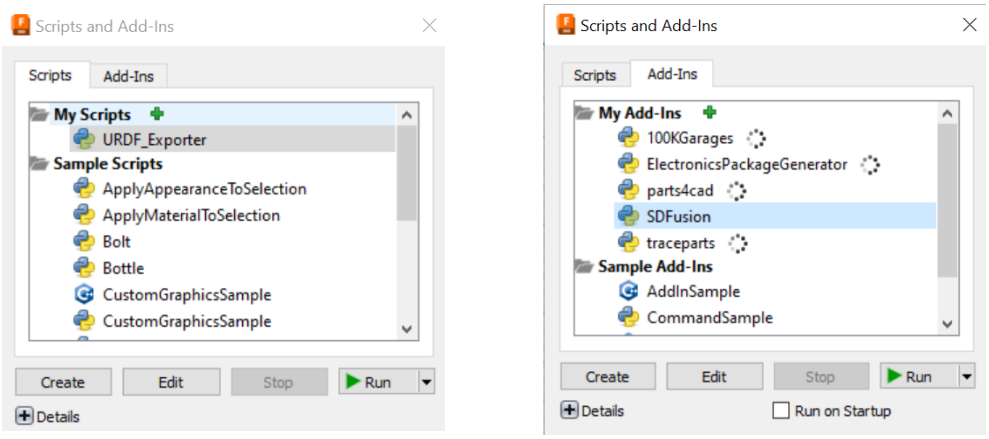
For å fikse dette må ledd eller ledgrupper defineres. Faste leddgrupper brukes for å binde sammen alle deler slik som de er plassert i tidspunktet leddgruppen defineres. Ettersom at noen deler var feilplassert ble det først lagd enkle faste ledd mellom to deler, da dette fører det ene leddet inntil det andre. På denne måten kan feilplasserte ledd flyttes til resten av roboten med millimeters nøyaktighet, kontra å måtte flytte delen med musepekeren. For ryddighetens skyld er først enkle faste ledd definert slik at alle delene på roboten er plassert riktig i forhold til hverandre, deretter er det lagd faste leddgrupper mellom alle ledd som sitter fast i hverandre før hvert enkelt ledd igjen er blitt slettet. Til slutt er de roterende leddene blitt lagd. I Fusion360 har roboten blitt definert til å være lagd av aluminium, og alle koblinger mellom hver enkelt del er enten lagd som et roterende ledd, eller en fast leddgruppe. Se figur 4.15



Figur 4.15: Leddgruppe for overkroppen.

Roboten har nå tre faste leddgrupper og to roterende ledd. Leddgruppene er Indre Beinpar, Ytre Beinpar og overkropp. De roterende leddene er begge den roterende delen i hver sin motor/gir kombinasjon.

Ved eksportering til URDF fil ble fusion2urdf skript benyttet. Dette skriptet støtter ikke "nested components" eller komponenter bestående av flere deler. Det var en del slike "nested components" i tegningen, så disse måtte slettes og alle delene måtte legges inn igjen på nytt, og alle ledd og leddgrupper måtte igjen bli definert. Etter å ha definert alt av ledd på nytt uten "nested components" ble modellen eksportert til URDF. [61] Ettersom at ROS bruker URDF og Gazebo bruker SDF er roboten også eksportert til SDF med SDFusion Add-In.[62] Forskjellen på en Add-In og skript i Fusion 360 er at et skript kun kjøres en gang også er den ferdig kjørt, mens en Add-In kan kjøre kontinuerlig. I dette tilfellet er SDFusion Add-In brukt som et skript for å eksportere til SDF. Se figur 4.16



Figur 4.16: Lokasjon for Add-Ins og Skript

4.3.2 Drøfting modellering av robot i F360

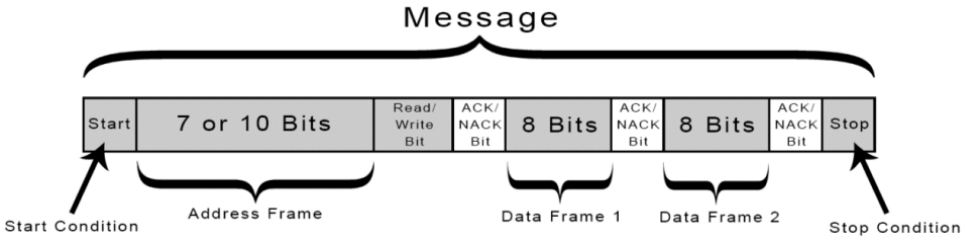
For simulering og visualisering av roboten i Rviz og Gazebo måtte det lages URDF og SDF filer som beskriver robotens egenskaper. Slike filer er kompliserte å skrive fra bunnen av. Etter samtaler med ressurspersonell og utforsking av muligheter ble det oppdaget at det eksisterer skript og Add-Ins til Fusion 360 som genererer disse filene for deg fra en 3d modell med definerte komponenter/deler og led(grupper). Slike tilleggsskript og -Add-Ins finnes også i SolidWorks som plugin for å konvertere fra SolidWorks filer til URDF.[63]. Både SolidWorks og Fusion 360 kunne ha blitt benyttet. Da det tidlig i prosjektet ble nevnt at Fusion 360 hadde disse funksjonalitetene, og det har fungert for andre studenter tidligere, landet beslutningen på å bruke Fusion 360 for modelleringen.

4.4 Kommunikasjonsprotokoll og maskinvare

4.4.1 Teori I2C

I2C også kjent som IIC og står for inter-intergrated circuit er en kommunikasjonsprotokoll. Det er en synkron multikontroller/multimål switchet, ensidig, seriell kommunikasjonsbuss. Det er en protokoll som er mye brukt for å koble perifere I2C'er med lavere hastighet til prosessorer og mikrokontrollere i kortdistansekommunikasjon.

Hvordan ulike enheter skal kommunisere og hvem som skal bestemme hvordan kommunikasjonen skal foregå. Det som menes med multikontroller og multimål er et ubestemt antall mastere/kontrollere/sendere og slaver/mål/mottakere. Kommunikasjonen kan foregå mellom ulike sett av mastere og slaver. Det kan være flere mastere enn slaver, og motsatt.



Figur 4.17: I2C kommunikasjonsBIT

I2C definerer ulike typer transaksjoner der hver av dem begynner med *start* og slutter med *stopp*. Det kan være en enkel melding der en master skriver data til en slave. Det kan være en enkel melding der en master leser data fra en slave. Det kan være et kombinert format der en master etterspør flere lesinger eller skriver data til en eller flere slaver.

For å sende en I2C-melding deles dataene inn i ulike deler der slaveadressen er i begynnelsen slik master(ene) vet hvilken slave som sendte meldingen. Før og etter del legges det til et bekreftelsessegment. Dette tillater systemet å oppdage om dataene er akseptert av mottakeren eller ikke. Dette er for å vite om meldingen må sendes på nytt fra slaven. Se figur 4.17 for en illustrasjon av disse bekreftelsessegmentene.

Det er to signaler knyttet til I2C-bussen: den serielle klokkelinjen (SCL) og den serielle datalinjen (SDA). Sistnevnte er en toveislinje som brukes til å sende og motta dataene til/fra grensesnittet.

4.4.2 Resultat I2C og IMU

Lodding og montering av nye komponenter var nødvendig for å kunne starte kommunikasjonsprosessen mellom innebygd maskinvare og komponentene. Testing av nye komponentene av alle tre IMU-er (SEL0337 BMX160) gjort i Arduino IDE som ga tilfredsstillende resultater. Videre for å få inn data fra alle tre sensorer

samtidig kreves det at de har forskjellige adresser. Disse IMU-ene har kun to forskjellige adresser. Fra den standard adressen 0x68 ble det loddet en kortslutning. Da fikk den adressen 0x69. For å få den tredje individuelle adressen ble det benyttet IMU LSM9DS1 som har adressen 0x1C. En løsning videre for å kunne avlese alle tre IMU-ene hver for seg var via Master-slave protokoll.

Kommunikasjonen mellom Ubuntu og BeagleBone Black ble testet med en enkel kobling med et LED-lys. For at dette skulle kunne gjennomføres måtte retning deklareres og verdi. echo out > direction, echo 1 > value. Her deklarertes retningen som ut, og verdien konstant som 1. For å velge hvilken utgangspinne kreves det at den valgte pinnen eksporteres. sudo echo 30 > export.

4.4.3 Drøfting I2C og IMU

Det kunne ha vært aktuelt å benytte IMU som ikke har geomagnetisk sensor da denne roboten kun beveger seg i to retninger. Både kostnad og leveringstid for slike IMU-er var mindre gunstige enn de som ble valgt.

Det kunne ha benyttes en multiplexer som kunne håndtere flere IMU av samme type og fortsatt klart å skille mellom hvilken IMU som sender hvilken informasjon og til hvilket tidspunkt. Det ble valgt å se bort i fra dette og heller benytte de komponentene som var tilgjengelig da det ga tilfredsstillende resultater.

I2C-protokoll er som regel raskere enn SPI Valgt I2C-protokoll da andre komponenter på selve roboten ikke støttet SPI-protokoll.

Kapittel 5

Reguleringsteori

5.1 Gåmønsteret til robot

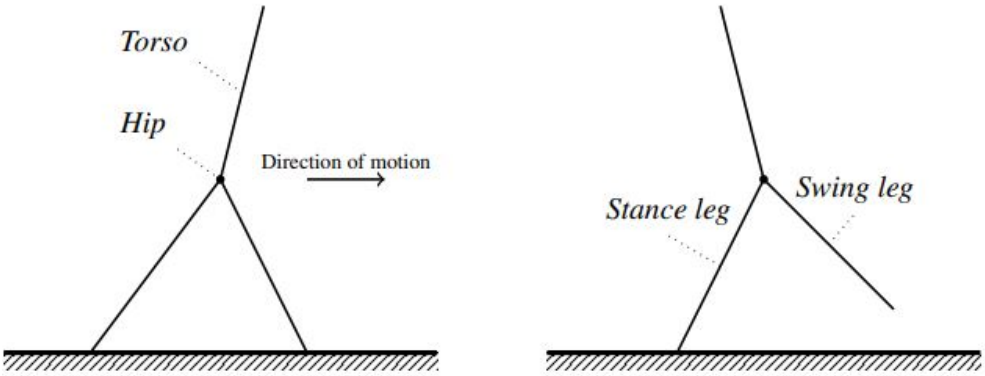
Roboten som blir beskrevet i denne teksten er en åpen kinematisk kjede som består av to bein sammenkoblet med overkroppen i et felles ledd, hoften. Når roboten skal gå vil den bytte mellom to faser. Èn fase der begge beina er i kontakt med bakken, og en annen fase når det er kun ett av beina som har bakkekontakt, der sistnevnte blir det beinet som har kontakt med bakken kalt det stående bein og det andre svingbeinet. Roboten er altså et hybridsystem hvor dynamikken i svingfasen kan modelleres ved hjelp av Euler-Lagrange-likningen.

Selve gåmønsteret er illustrert i figur 5.1. Under fasen der kun ett av beina har bakkekontakt fungerer det stående beinet som et dreieledd fra tuppen av beinet som har bakkekontakt. Overgangen til neste fase blir når svingbeinet treffer bakken. Dette støtet betraktes som perfekt uelastisk med andre ord ingen glidning mot underlaget. Dette svingbeinet blir det nye stående beinet. Ved dette støtet blir det gamle stående bein det nye svingbeinet. Gåmønsteret blir betraktet som symmetrisk og gjentagende som vil si at konfigurasjonen er identisk i begynnelsen av hvert påfølgende steg med unntak av at det stående beinet og svingbeinet byttes mellom hvert av beinene. Roboten vil kun bevege seg rett fremover langs horisontalplanet. [64]

5.1.1 Teori hengende pendel

Hengende pendel er en masse som henger ned under festepunktet og har mulighet til å dreie rundt dette punktet. Ved en start utenfor likevektspunktet vil pendelen bevege seg periodisk, og energien vil omdannes mellom potensiell energi og bevegelsesenergi. Gravitasjonskreftene vil trekke tyngdepunktet nedover og starte den periodiske bevegelsen. Friksjon og varmeutvikling vil sørge for at pendelen mister energi og kommer til likevektspunktet sitt, som er når pendelen henger stille rett ned. Tiden per periode, med andre ord tiden per svingning, er avhengig av lengden på pendelen.

For å oppsummere fysikken til pendelen vil en masse m henge langs en uelas-



Figur 5.1: Stå- og svingebein, illustrasjon hentet fra Masteroppgave *Stable Gaits for an Underactuated Compass Biped Robot with a Torso* av C.Sætre, side 35

tisk lenge l på fra pendelens dreiepunkt d. Gravitasjonskreftene g vil virke med en kraft lik$m \cdot g$ på pendelens masse m. Svingetiden T vil ha denne sammenhengen $T = 2 \cdot \pi \cdot \sqrt{\frac{l}{g}}$. [65]

5.1.2 Teori invertert pendel

En invertert pendel er en pendel som har sitt tyngdepunkt høyere enn sitt dreiepunkt. Systemet har et likevektspunkt, men er svært ustabilt og uten krefter utenifra vil den inverterte pendelen falle. Dette vil si at polene til systemet vil ligge i høyre halvplan. Polene til en overføringsfunksjonsrepresentasjon av et LTI-system er de samme som egenverdiene til systemmatrisen til en tilstandsrommodellrepresentasjon av det samme LTI-systemet.

Systemet kan holdes ved like ved hjelp av et aktivt lukket tilbakekoblet kontrollsystem som overvåker systemet og gir tilbakemelding om statusen til systemet og tilpasser seg deretter. Systemet prøver å tilpasse sin input til å optimalisere selve outputen til systemet. For å få til dette trengs det sensorer som overvåker selve systemet, og det trengs noen aktuatorer som eksempel til den inverterte pendelen er motorer som kan forandre vinkelen. Forandrer så vinkel for å så flytte dreiepunktet horisontalt tilbake under pendelens tyngdepunkt slik at den inverterte pendelen ikke faller, og heller holder selve balansen. Ved en invertert pendel som overkroppen til roboten som jobbes med i denne bacheloroppgaven kan pendelen falle kun to retninger, fremover og bakover.[66][67]

5.1.3 Teori regulering av robot

PD-Regulator

PD-regulator er en type regulator i et kontrollsystem der outputen varierer i forhold til feilsignalet, også kjent som proporsjonal deriverte kontroller. Denne type

regulator beregner output, pådrag, med hjelp av matematiske operasjoner proporsjonal forsterkning (P) og derivatvirkning (D) brukt på inngangssignal. Derivatdelen i PD-regulatoren har som oppgave å lage et tilleggssignal til pådraget som skal motvirke endringer i den målte prosessverdien. Se illustrasjon av blokkdiagram for PD-regulator i figur 5.2.

Den matematiske operasjonen mellom input og output for PD-regulatoren kan skrives som: $u = K_p \cdot (e + T_d \cdot \frac{de}{dt}) + u_0$

(5.1)

Hvor

$$e = r - y, \tag{5.2}$$

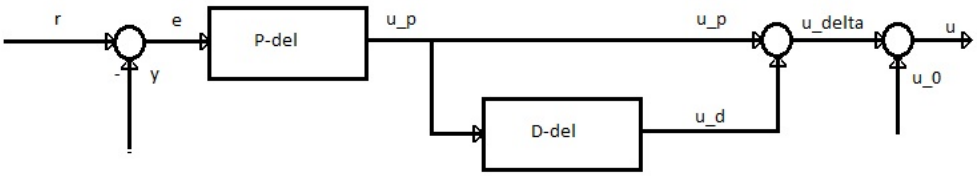
$$u = u_P + u_D + u_0, \tag{5.3}$$

$$u_P = K_p \cdot e \tag{5.4}$$

$$u_D = K_P \cdot T_d \cdot \frac{de}{dt} \tag{5.5}$$

Der u_P er P-pådraget, u_D er D-pådrag, u_0 er nominelt pådrag. e er reguleringsavviket, de/dt altså e' er den tidsderivert av reguleringsavviket. T_d er derivasjonstiden. K_p er proporsjonalforsterkning og K_d er derivatforsterkningen.

PD-reguleringen har et ledd som er proporsjonalt med den deriverte av avviket. Dette leddet vil gi et tillegg til pådraget så lenge den deriverte av avviket er forskjellig fra null. Det betyr at når vi har stasjonær tilstand i sløyfa så blir bidraget fra D-delen lik null. Da er den målte prosessverdien konstant og den deriverte av denne konstanten blir null. [68]



Figur 5.2: Blokkdiagram for modell med PD-regulator

Hensikten med tilbakekoblingen er å gjøre $e = y_0 - y$ så liten som mulig. Dette for å gjøre outputen så nærme den ønskede outputen som mulig.

Fremgangsmåten som ville vært benyttet hadde vært å bruke Ziegler-Nichols metode for grovinnstilling av PD-regulatoren. Ved avlesing fra Ziegler-Nichols tabell for PD-regulator. $K_p = 0,65 \cdot K_k og T_d = 0,12 \cdot T_k$

Deretter finjusteringer. Dersom signalet hadde vært for urolig ville en reduksjon av K_p, eller justering av T_d på følgende vis: Reduksjon av T_d til null og økning av T_d gradvis inntil stabiliteten er god nok. Ved et for stort dynamisk avvik, så kan T_d økes. Ved et for stort stasjonært avvik, så kan K_p økes. [68]

Teori Kalmanfilter

For statistikk og kontrollteori er Kalman-filtrering, også kjent som lineær kvadratisk estimering (LQE), en algoritme som bruker en serie målinger observert over tid, inkludert statistisk støy og andre unøyaktigheter, og produserer estimater av ukjente variabler som har en tendens til å være mer nøyaktige enn de som er basert på en enkelt måling alene, ved å estimere en felles sannsynlighetsfordeling over variablene for hver tidsramme.

Kalmanfilter benyttes for å estimere tilstanden til et dynamisk system. Kalmanfilter kan deles inn i to deler. En forutsigende del, og en korrigerende del. Den forutsigende delen tar styringsdataen i betraktning for å kunne estimere gjeldende tilstandsvariabler, sammen med deres usikkerheter. Når utfallet av neste måling er observert, oppdateres disse estimatene ved å bruke et vektet gjennomsnitt, med større vekt til estimater med større sikkerhet. Algoritmen er rekursiv, ettersom den baserer estimeringen på tidligere estimeringer. Den kan operere i sanntid, kun ved å bruke de nåværende inngangsmålingene og tilstanden beregnet tidligere og dens usikkerhetsmatrise.

Kalmanfiltrering har to antagelser som ligger til grunn. Den første er at alt er Gaussisk, og den andre er at alle modeller er lineære. Hvis systemet befinner seg i en Gaussian lineær verden så vil Kalmanfilter være den optimale estimeringen og tilstandsberegningen. Ved et ikke-lineært system som er tilfelle for de aller fleste systemer må kalmanfilteret utvides. Dette gjøres ved å linearisere de ikke lineære systemene ved hjelp av Taylor approksimering. [69] [70]

Drøfting av regulering

Videre kunne det blitt sammenlignet PD-regulering opp mot Kalman filtreringen. Simulering og utprøvinger med justeringer for å se hvilke av metodene som optimaliserer gangen til roboten. Kalman filtrering kan også benyttes sammen med en PD-regulator for optimale målinger og reguleringer. [71][72]

Kapittel 6

Arbeidsprosessen

6.1 Teori arbeidsprosessen

Arbeidsprosessen er en beskrivelse av arbeidsmetode og selve prosessen til gruppen opp mot problemstillingen. Dette inneholder hva som jobbes med, tidsbruk og av hvem. Hvordan forventningene til samarbeid og individuelt arbeid.

6.1.1 Resultat arbeidsprosessen

Arbeidsprosessen og fremgangsmåten ble satt i forprosjektet som ble fulgt med noen modifikasjoner på tidsbruk på noen av de ulike arbeidspakkene. Fossefallmetoden per individ på de ulike arbeidspakkene. Noen arbeidspakker ble det satt flere personer på, andre arbeidspakker ble fordelt individuelt.

Det ble loggført arbeidstimer der de fulle arbeidsdagene ble oppført som satte fokus på arbeidspakkene og skapte tillit til hverandres kapabilitet og pågangsmot. Noe sykdom og karantenetid inntraff som vurdert i risikoanalysen. Har hatt etter planen regelmessige møter med veileder. Et møte annenhver uke, med noen unntak rundt påskeferien. Vi hadde jevnlige oppdateringen internt i gruppen for å vite hva alle gjør og hvordan arbeidet går. Dette for å samkjøre arbeidet og for å se an behovet for ressursallokering.

Den planlagte tiden per arbeidspakke inneholdt en stor del av estimert tid for å løse uforutsette ekstra utfordringer. Her gikk de første arbeidspakkene som planlagt. Noen arbeidspakker ble mer hensiktsmessig at en person jobbet med da det var kun en datamaskin som kunne jobbe med fremgangen samtidig. Da ble det tre personer på parallelle pakker.

Det ble møtt store utfordringer på kommunikasjonsoppsettet med BBB og ROS for å få kjørt enkle koder. Innhentet info fra ressurspersonell og andre utenfor tidligere nevnte ressurspersonell, men ingen som hadde en løsning på det. Tok kontakt med tidligere prosjektgruppe, men resurspersonell hadde heller ingen løsning. Dette tok mye lengre tid enn estimert i forprosjektet og arbeidspakker som var avhengig av kommunikasjonsoppsettet ble forskjøvet. Videre ble det besluttet med hensyn på tiden som var igjen av prosjektet å fokusere på ting som

lot seg gjennomføres i prosjektet. Arbeid som har med gangen og selve regulering balansen til overkroppen blir til eventuelle fremtidige grupper.

Drøfting arbeidsprosessen

Kunne kontaktet tidligere gruppemedlemmer fra første dag bacheloroppgaven startet og bedt tidligere gruppe demonstrere alt for å sjekke hva som faktisk fungerte til forskjell på hva som stod i rapporten. Dette kunne ha kortet ned prosessen med feilsøking. Fått se hva som fungerer og hvordan. Samtidig fått kartlagt faktiske problem som ikke kom fram i tidligere rapport.

Kapittel 7

Drøfting av prosjektet

Oppgaven var rettet mot våre spesialiseringer, men utfordringene og problemene underveis havnet utenfor våre fordypinger innfor studiet. Her hadde det vært nyttig med en person som gikk data med mer kunnskap rettet mot Linux.

Dette medførte at vår problemstilling fra forprosjektet ble endret for å tilpasses mot det arbeidet som ble utført. Her fikk vi ikke jobbet med og lært av hverandres fordypinger i like stor grad som ønsket. Til tross for utfordringene som var rettet mot Linux ble fortsatt flere av problemene løst, det var dog meget tidkrevende.

Planen var å teste alle enkeltkomponenter hver for seg for å luke ut alle defekte komponenter. Etter dette skulle alt av eksisterende kode og oppsett kjøres på allerede eksisterende BeagleBone Black. Ettersom mini USB-porten på den eksisterende Beaglebone Black var revet av var dette dessverre ikke mulig.

Ved oppsett på ny BeagleBone Black ble det oppdaget at Ubuntu 18.04 ikke lengre var tilgjengelig å laste ned. Under installasjon av 20.04 oppsto det en del problemer og det samme med installasjon av ROS. Dermed ble fokus satt på å løse problemene med oppsett framfor å kjøre eksisterende C++ kode fra 2021.

Det var mye nytt å sette seg inn i, hvor vi brukte mye tid på å tilegne oss kunnskap fra diverse plasser på internett angående hvordan ROS henger sammen og grensesnittet mellom den fysiske I/O-delen til BeagleBone Black og Ubuntu-operativsystemet. Det hadde vært flott med et arveprosjekt som dette om det ble organisert en overgivelse med demonstrasjon fra tidligere gruppe til den nye. Fra første dag i arbeidsperioden kunne man bedt tidligere gruppe demonstrere alt, for å få sett hva som fungerer og hvordan. Dette kan gjøre en overtakelse mer effektiv. Samtidig fått kartlagt faktiske problemområder som ikke kommer frem i tidligere rapport.

Tidligere erfaring av I/O-kontroll var med IDE-er som tar seg av grensesnittet til de tilhørende mikrokontrollerne. Ettersom BeagleBone Black er en innebygd ettkortsdatamaskin hvor I/O styres direkte i selve datamaskinen var det en utfordring å skjønne hvordan kommunikasjon kunne opprettes sett i sammenheng av tidligere mikrokontrollerkunnskap.

Beaglebone Black er relativt lite brukt og dermed var det vanskelig å finne relevant informasjon som kunne hjulpet med forståelsen. Det ble ofte søkt opp

informasjon om hvordan ting gjøres på Raspberry Pi i håp om å kunne overføre kunnskapen til Beaglebone Black.

Hovedproblemet er som tidligere nevnt mangelen på bibliotek til valgt maskinvare, det å skrive egne bibliotek til valgt oppsett ble for tidkrevende da det samtidig var kunnskapsmessige konseptuelle hull.

På maskinvarefronten ble det gjort mye framgang ettersom det ble kjøpt inn nye komponenter av det som var defekt fra forrige gruppe. Både de nye komponentene, og de som ikke var regnet som defekt ble funksjonstestet med tilfredsstillende resultat, noe som gjør at neste gruppe kan fokusere mer på programmeringsbiten.

Modell av roboten ble også ferdigstilt noe som var en betydelig arbeidsprosess, dette vil fjerne mye last til neste gruppe.

Planer til simulering og regulering krever nøyaktig informasjon om robotens tilstand, dette skulle oppnås med hjelp av de tre IMU-ene. Ettersom en IMU har innebygd tre forskjellige sensorer som kan brukes for å oppnå nøyaktig posisjon og orientering, kreves en måte å tolke disse sensoravlesningene på en måte som kan virkeliggjøre dette målet. Det å slå sammen flere sensordata kalles sensorfusjon, og en fornuftig metode vil være et kalmanfilter.

Kapittel 8

Konklusjon og fremtidig arbeid

8.1 Konklusjon

Hovedmålet for prosjektet har vært å få en fungerende underaktuert robotprototype. Tidligere arbeid er dokumentert, men omfanget av hva som skulle gjøres var enda ikke helt klart. Det som derimot var klart var at alle deler bør funksjonstestes, og at tidligere 3D modell ikke har vært tilstrekkelig god.

Alle de fysiske komponentene er nå fungerende ettersom defekte komponenter er byttet ut med nye. Enkodere er nå absolutte, og ikke inkrementelle, så det er mindre sannsynlighet for avvik tilknyttet samplingstid på hofteledd. 3D-modell er også blitt lagd med definert materiale og ledd. Denne 3D-modellen kan enkelt konverteres både til URDF og SDF fil som skal gjøre integrering til Rviz og Gazebo så sømløst som mulig og med riktig fysikk.

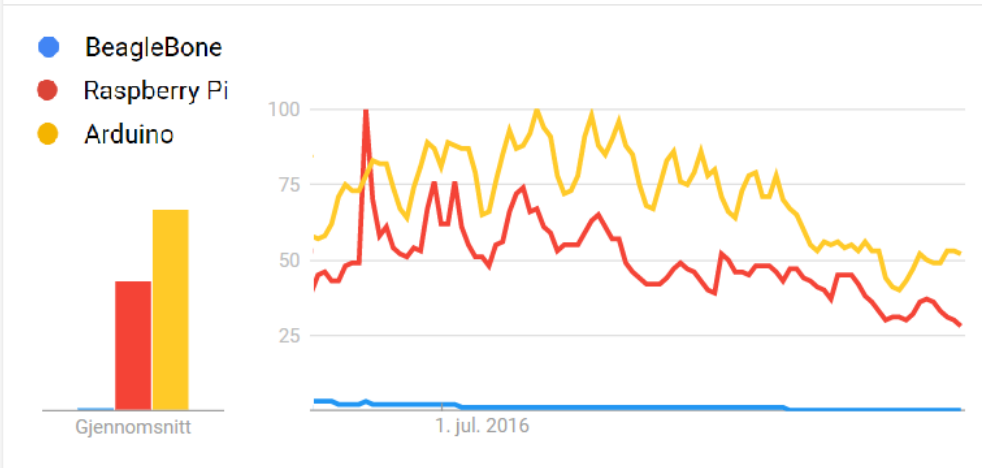
Fjorårets oppsett med ROS-kode viste seg å ikke kunne testes på grunn av komplikasjoner knyttet til BeagleBone Black og IMU-kompatibilitet med C++. Fokus for prosjektet ble derfor tilpasset disse utfordringene. Mye av tiden gikk til å lete etter svar på utfordringene som har oppsto, forsøke å løse de, og deretter enten forsøke å fikse problemet med en annen metode, eller gå videre til neste problem. Ressurser tilknyttet BeagleBone Black var også noe utfordrende å oppdrive da offisiell støtte ble stoppet fra april 2020. All videreutvikling og støtte av BeagleBone Black drives nå av frivillighet. Figur 8.1 viser at populariteten knyttet til Raspberry Pi er vesentlig større, som tilsier at det er mer støtte og ressurser knyttet til den.

Samarbeidet i gruppen har fungert bra og informasjonsdeling har i hovedsak foregått fysisk i form av samtaler med hverandre, og fildeling på Teams. Medlemmene i gruppa har jobbet målrettet med sine ansvarlige arbeidsområder samtidig som terskelen for spørsmål, diskusjoner, idemyldring og samarbeid har vært lav.

Alle på gruppen har tilegnet seg masse ny kunnskap. Ved oppstarten av oppgaven var Linux, C++, ROS og 3D-modellering i stor grad ukjent for gruppen. I retrospekt ser alle medlemmene at de sitter igjen med langt mer kunnskap innenfor disse temaene enn for kun noen måneder siden.

Uten muligheter for gjenskaping av nøyaktig samme arbeid som tidligere er

roboten på et vis lengre vekk fra å kunne gå da programvare ikke er helt klart, samtidig som den er nærmere med å kunne gå da roboten fysisk ikke har skader. Tanker rundt videreutvikling av roboten står det om i kapittelet om framtidig arbeid.



Figur 8.1: Maskinvaretrend ved Googlesøk fra 2015 til 2022

8.2 Fremtidig Arbeid

Testing av roboten ble dessverre ikke gjennomført. Dermed ligger en del framtidig arbeid fra fjorårets gruppe også med videre. Punktene som følger er dermed vår anbefaling.

8.2.1 Enkodere

Ettersom enkoderne nå er absolutte og ikke inkrementelle må ny kode for disse skrives. I tillegg til ny kode må neste gruppe også sette nullpunktsverdi for enkoderne slik at enkoderne har et bestemt utslag ved likevektspunkt (for eksempel 0).

8.2.2 Servoalgoritme

Algoritmer for kontrollering av servomotorer må lages slik at det frie beinet til roboten kan svinge uten å treffe bakken. Tanken er at servomotorene på det frie beinet strekker ut tærne etter passering av det stående beinet, og det stående beinet blir det frie beinet når det skifter retning fra en bakoverbevegelse til en framoverbevegelse. I det vektskiftet utføres kan det nye frie beinet trekke inn tærne tilknyttet servomotorene.

8.2.3 Valg av hovedenhet

Mye av problemene som har oppstått kunne vært unngått om en mer populær hovedenhet hadde blitt brukt. De nye IMU-ene fungerer fint på Arduino og har biblioteker til Raspberry Pi. Ettersom det ikke er offisiell støtte for BeagleBone Black lengre kan det være et alternativ å gå over til en mer populær løsning. Om neste gruppe velger å gå for en Raspberry Pi 3 eller nyere har de også mulighet til å gå over til ROS2 om ønskelig.

8.2.4 IMU

Hovedproblemet er som tidligere nevnt utfordringen med å få til I2C-kommunikasjon direkte mellom valgte IMU sensorer og BeagleBone Black. Dette kommer av manglende biblioteker som passer med valgt maskinvare. Om neste gruppe velger å beholde BeagleBone Black kan det være lurt å investere i IMU-er med biblioteker til C++. Tidligere grupper har hatt problemer med støy knyttet til IMU-ene, spesielt ved støt mot bakken. Det bør derfor implementeres filtre og løsninger som kan ta fatt på det problemet, som for eksempel kalmanfilter som er beskrevet tidligere i oppgaven.

8.2.5 ROS oppsett

ROS oppsettet fra fjorårets gruppe har ikke blitt benyttet da dette er en gammel versjon av ROS og Ubuntu. Det er ikke undersøkt hvor komplisert det vil være å få dette oppsettet til å fungere på nyere versjoner av ROS og Ubuntu. Eventuelt kan framtidig gruppe vurdere å gå over til ROS2. Siden flere og flere går over til ROS2 blir det mer og mer kunnskap, ressurser og pakker knyttet til det.

8.2.6 PD-kontroller

Ettersom roboten ikke har blitt operativ har det heller ikke blitt implementert noen form for regulering. Implementasjon av PD-regulator skal være optimalt for en robot som dette.

Bibliografi

[1] Wikipedia.com, *Aluminium*, Available at https://en.wikipedia.org/wiki/Aluminium(2022/05/12).

[2] rs-online.com, *TTi*, Available at https://docs.rs-online.com/c650/0900766b80f74584.pdf(2022/05/12).

[3] github.com, *SRMintro*, Available at https://github.com/beagleboard/beaglebone-black/wiki/System-Reference-Manual#introduction (2022/05/10).

[4] wikipedia, *SBC*, Available at https://en.wikipedia.org/wiki/Single-board_computer (2022/05/09).

[5] github.com, *bsdBBB*, Available at https://wiki.freebsd.org/arm/BeagleBoneBlack (2022/05/10).

[6] wikipedia, *BBB*, Available at https://en.wikipedia.org/wiki/BeagleBoard#BeagleBone_Black (2022/05/09).

[7] ti.com, *CPU*, Available at https://www.ti.com/lit/ds/symlink/am3358.pdf?ts=1652163597655&ref_url=https%253A%252F%252Fwww.google.de%252F (2022/05/10).

[8] elinux.org, *BBB*, Available at https://elinux.org/Beagleboard:BeagleBoneBlack (2022/05/09).

[9] github.com, *SRMpdf*, Available at https://github.com/beagleboard/beaglebone-black/blob/master/BBB_SRM.pdf (2022/05/10).

[10] beagleboard.com, *getStartUpdate*, Available at https://beagleboard.org/getting-started#update (2022/05/10).

[11] github.com, *LatestImage26*, Available at https://github.com/beagleboard/Latest-Images/issues/26 (2022/05/10).

[12] elinux.org, *BBBubuntu*, Available at https://elinux.org/BeagleBoardUbuntu (2022/05/10).

[13] ros.org, *reps*, Available at https://ros.org/reps/rep-0003.html (2022/05/10).

[14] maxongroup.com, *Maxon Group*, Available at https://www.maxongroup.com/maxon/view/product/control/4-Q-Servokontroller/422969(2022/05/12).

[15] Wikipedia.com, *Motorteori*, Available at https://en.wikipedia.org/wiki/DC_motor(2022/05/12).

[16] maxongroup.com, *Maxon Group*, Available at https://www.maxongroup.com/medias/sys_master/root/8881624186910/EN-21-151.pdf(2022/05/12).

[17] Wikipedia.org, *StallTorque*, Available at https://en.wikipedia.org/wiki/Stall_torque(2022/05/14).

[18] C. F. Sætre, «Stable Gaits for an Underactuated Compass Biped Robot with a Torso,» ph.d.-avh., NTNU, 2016, s. 63–66.

[19] dfrobot, *I2C Hardware Hookup*, Available at https://wiki.dfrobot.com/BMX160_9_Axis_Sensor_Module_SKU_SEN0373 (2022/04/21).

[20] Sparkfun, *I2C Hardware Hookup*, Available at https://learn.sparkfun.com/tutorials/lsm9ds1-breakout-hookup-guide#hardware-hookup (2022/03/29).

[21] Wikipedia, *Rotary encoder*, Available at https://en.wikipedia.org/wiki/Rotary_encoder (2022/04/09).

[22] Heidenhain, *Single-turn*, Available at https://www.heidenhain.us/resources-and-news/difference-between-single-turn-and-multi-turn-encoders/ (2022/04/11).

[23] Digikey, *Single-turn*, Available at https://www.digikey.no/en/product-highlight/c/cui/amt-absolute-encoders (2022/04/21).

[24] Mouser, *Absloute encoder AMT233A-V*, Available at https://no.mouser.com/datasheet/2/670/amt23_v-1776596.pdf (2022/04/09).

[25] Wikipedia.com, *ServoTeori*, Available at https://en.wikipedia.org/wiki/Servomotor#:~:text=A%20servomotor%20(or%20servo%20motor,a%20sensor%20for%20position%20feedback.(2022/05/12).

[26] Parallax.com, *Servo Datablad*, Available at https://no.mouser.com/datasheet/2/321/900-00005-Standard-Servo-Product-Documentation-v2.-462659.pdf(2022/05/12).

[27] ROS.org, *Multics*, Available at https://multicians.org/history.html(2022/05/16).

[28] gnu.org, *GPL*, Available at https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html(2022/05/16).

[29] D. Molloy, *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*, 2nd ed. Hoboken, NJ, USA: Wiley, 2019.

[30] technopedia.com, *kommandotolk*, Available at https://www.techopedia.com/definition/7793/interpreter(2022/05/19).

[31] Albatross, *C++*, Available at https://www.cplusplus.com/info/description/(2022/05/18).

[32] S. Sharma, *C with classes*, Available at https://www.linkedin.com/pulse/c-classes-sunil-sharma/(2022/05/19).

[33] ROS.org, *Basics*, Available at https://wiki.ros.org/ (2022/04/11).

[34] wikipedia.org, *ROS2*, Available at https://en.wikipedia.org/wiki/Robot_Operating_System#ROS_2 (2022/05/01).

[35] ROS.org, *Introduksjon*, Available at https://wiki.ros.org/ROS/Introduction (2022/04/22).

[36] ROS.org, *Konsepter*, Available at https://wiki.ros.org/ROS/Concepts (2022/04/22).

[37] ROS.org, *Parameter Server*, Available at https://wiki.ros.org/Parameter%20Server (2022/04/22).

[38] ROS.org, *Master*, Available at https://wiki.ros.org/Master (2022/04/22).

[39] ROS.org, *Bug Ticket System*, Available at https://wiki.ros.org/Tickets (2022/04/22).

[40] wiki.org, *tf2*, Available at http://wiki.ros.org/tf2 (2022/05/01).

[41] ROS.org, *Høyere nivå konsepter*, Available at https://wiki.ros.org/ROS/Higher-Level%20Concepts#actions (2022/04/23).

[42] ROS.org, *cpp*, Available at https://wiki.ros.org/Client%20Libraries (2022/05/02).

[43] ROS.org, *teknisk*, Available at https://wiki.ros.org/ROS/Technical%20Overview (2022/05/02).

[44] ROS.org, *msg*, Available at https://wiki.ros.org/msg (2022/05/02).

[45] ROS.org, *srv*, Available at https://wiki.ros.org/srv (2022/05/02).

[46] ROS.org, *commonMessages*, Available at https://wiki.ros.org/common_msgs (2022/05/02).

[47] ROS.org, *tools, Rviz++*, Available at https://en.wikipedia.org/wiki/Robot_Operating_System#Tools (2022/05/03).

[48] ROS.org, *gazeboPlugins*, Available at https://wiki.ros.org/gazebo_plugins?distro=groovy (2022/05/05).

[49] gazebosim.org, *gazebosim*, Available at https://classic.gazebosim.org/ (2022/05/05).

[50] gazebosim.org, *rosGzPlugins*, Available at https://classic.gazebosim.org/tutorials?tut=ros_gzplugins (2022/05/05).

[51] gazebosim.org, *tutorial*, Available at https://classic.gazebosim.org/tutorials?cat=guided_b&tut=guided_b1 (2022/05/15).

[52] gazebosim.org, *tutorial*, Available at https://classic.gazebosim.org/tutorials?cat=guided_a&tut=guided_a1 (2022/05/15).

[53] gazebosim.org, *connectROSgazebo*, Available at https://classic.gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros (2022/05/05).

[54] gazebosim.org, *tutorial*, Available at http://wiki.ros.org/urdf/Tutorials/Using%20a%20URDF%20in%20Gazebo (2022/05/15).

[55] roboti.us, *alternativSimulering*, Available at https://www.roboti.us/index.html (2022/05/05).

[56] mujoco, *mujocoLatest*, Available at https://mujoco.readthedocs.io/en/latest/overview.html (2022/05/05).

[57] reddit, *alternativSimulering*, Available at https://www.reddit.com/r/robotics/comments/s9e9fc/alternative_simulation_environments_to_gazebo/ (2022/05/05).

[58] arxiv.org, *Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning*, Available at https://arxiv.org/pdf/2103.04616.pdf (2022/05/05).

[59] gazebosim.org, *userguide*, Available at https://wiki.ros.org/rviz/UserGuide (2022/05/05).

[60] gazebosim.org, *connectROSgazebo*, Available at https://classic.gazebosim.org/tutorials?cat=connect_ros (2022/05/05).

[61] Syuntoku14, *Fusion2URDF*, Available at https://github.com/syuntoku14/fusion2urdf(2022/05/12).

[62] Roboy, *SDFusion*, Available at https://github.com/Roboy/SDFusion(2022/05/12).

[63] ROS.org, *SolidWorks*, Available at http://wiki.ros.org/sw_urdf_exporter/Tutorials/Export%20an%20Assembly(2022/05/12).

[64] C. F. Sætre, «Stable Gaits for an Underactuated Compass Biped Robot with a Torso,» ph.d.-avh., NTNU, 2016, s. 35–45.

[65] Wikipedia, *Hengende-pendel*, Available at https://en.wikipedia.org/wiki/Pendulum(2022/04/22).

[66] Wikipedia, *Invertert-pendel*, Available at https://en.wikipedia.org/wiki/Inverted_pendulum (2022/04/22).

[67] youtube.com, *tilbakekobling*, Available at https://www.youtube.com/watch?v=Pi7l8mMjYVE&list=PLMrJAkhIeNNR20Mz-VpzgfQs5zrYi085m(2022/05/12).

[68] K. Bjørvik, *Reguleringsteknikk*, 2016.

[69] KalmanWiki, *Kalman*, Available at https://en.wikipedia.org/wiki/Kalman_filter(16/04/22).

[70] KalmanVideo, *Kalman*, Available at https://www.youtube.com/watch?v=o_HW6GnLqvg(16/04/22).

[71] S. Yuliawan, *Kalman*, Available at https://journal.ugm.ac.id/ijitee/article/download/64511/32205(16/05/22).

[72] S. Wakitani, H. Nakanishi og Y. Ashida, *Kalman*, Available at https://folk.ntnu.no/skoge/prost/proceedings/PID-2018/0064.PDF(16/05/22).

Vedlegg

A Plakat

VIDEREUTVIKLING OG FEILSØKING AV UNDERAKTUERT ROBOTPROTOTYPE

Knut Harald Gabow Au, Karl Magnus Sundelin, Tove Ingeborg Løvø, Tor Berge Sæbjørnsen

NTNU
Kunnskap for en bedre verden

Introduksjon

Oppgaven omhandler videreutvikling og feilsøking av en underaktuert tobeinet robotprototype, og bygger på tre tidligere bacheloroppgaver som alle har videreutviklet roboten. Overordnet problemstilling for prosjektet er å løse hardwareutfordringene, kartlegge programvarekompatibilitet, og utvikle en funksjonell 3D-modell for simulering og visualisering. Gruppen har god tverrfaglighet med to studenter fra Elektronikk og Sensorsystemer, og to studenter fra Automasjon og Robotikk.

Programvare

Tidligere gruppers oppsett på programvaredelen var ikke mulig å gjenskape og det ble valgt å benytte programvarenes siste versjoner som har støtte flere år fremover. Koder til nye komponenter ble testet og utført med Arduino UNO og Arduino C med tilfredsstillende resultat. Det ble besluttet å beholde ROS for å videreutvikle kommunikasjonsplattformen. Det er blitt arbeidet med C++ kode direkte i Linux-terminalen og ROS ved gjennomgang av tutorials i Turtlesim og Gazebo.

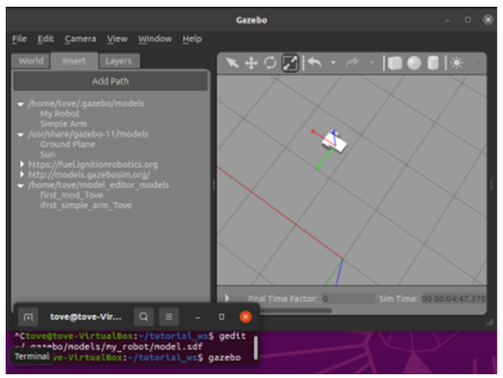


Fig. 1: tohjulet robot i Gazebo

Enkel Gazebosimulering av tohjulet robot figur 1 ble lagd for å bli bedre kjent med Gazebo. En funksjonell 3D-modell av roboten er konstruert i Fusion 360 og kan benyttes videre i sanntidsvisualisering og simulering.

Roboten

Rammeverket til roboten består i hovedsak av aluminium. Ellers på roboten finnes det 3 IMU-er, 2 motorer med gir, 2 enkodere, 4 servomotorer, 2 servokontrollere og en BeagleBone Black. En to-kanals strømforsyning forsyner resten av komponentene med strøm. Roboten har 3 ledd (indre ben, ytre ben og overkropp), men kun to aktuatorer, noe som betyr at roboten er underaktuert. Underaktuasjon er vanskelig å regulere da et (eller flere) ledd er i fri bevegelse og ikke kan aktueres.

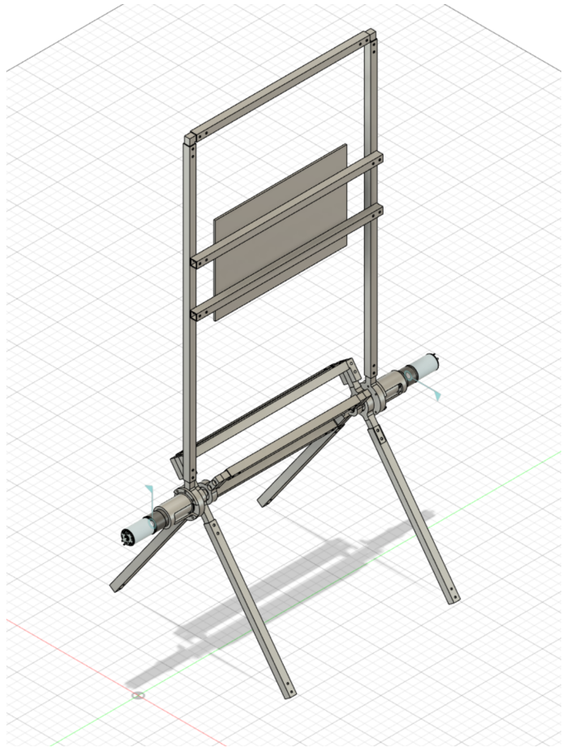


Fig. 2: 3D-modell av roboten i helfigur

3D-modell av roboten er lagd i Fusion 360 med definerte ledd og faste leddgrupper. 3d-modell vises i figur 2.

Sammendrag

Det er utført en hardwareundersøkelse av alle robotens komponenter da tidligere gruppers arbeid pekte på en del utfordringer knyttet til de. Her ble det testet hvilke komponenter som var fungerende og hvilke som var defekt. Det ble bestilt nye komponenter og de defekte ble byttet ut. Det har vært en større utfordring enn tiltenkt å få de nye komponentene til å fungere. Nøyaktig 3D-modell er utformet slik at framtidige grupper enkelt kan foreta simuleringer. Reguleringsteori ble utformet for videre arbeid med roboten. Det er i stor grad jobbet med oppsett og kommunikasjon mellom komponenter via BeagleBone Black der SSH kommunikasjon har vært brukt for å kommunisere med en datamaskin og BeagleBone Black.

Konklusjon

Hovedmålet for prosjektet har vært å få en fungerende underaktuert robotprototype. Tidligere arbeid er dokumentert, men omfanget av hva som skulle gjøres var enda ikke helt klart. Det som derimot var klart var at alle deler bør funksjonstestes, og at tidligere 3D modell ikke har vært tilstrekkelig god. Alle defekte komponenter ble kartlagt og byttet. Fjorårets oppsett med ROS-kode viste seg å ikke kunne testes på grunn av komplikasjoner knyttet til BeagleBone Black og IMU-kompatibilitet med C++. Fokus for prosjektet ble derfor tilpasset disse utfordringene. Alle på gruppen har tilegnet seg masse ny kunnskap. Ved oppstarten av oppgaven var Linux, C++, ROS og 3D-modellering i stor grad ukjent for gruppen.

B Bestillingsliste

Innkjøp:

Komponent:	Antall:	Navn/modell	Leverandør	Pris pr.25.02.22		totalpris
BBB	1	102110420	Digikey	589,31	pr.stk NOK	589,31
Encoder	2	AMT233A-V	Mouser	418,1	pr.encoder NOK, ved bestilling av to	836,2
Header encoder	2	5025780600	Digikey	4,15	pr.stk NOK	8,3
Motor/6 Gir	1	148877-260551	Maxon	6170,9	pr.stk (kombo)	6170,9
Servo	4	900-00005	Mouser	157,1	pr.stk NOK	628,4
IMU	3	Sen0373	Mouser	121,6	pr.stk NOK	364,8

TOTAL 8597,91

Omregning pris motorkombinasjon:		
394,09€(motor), 223€(gir)		394,09
		223
		617,09
Valutaomregning € til NOK pr.03.03.22		6170,9

C Inventarliste

Inverntarliste

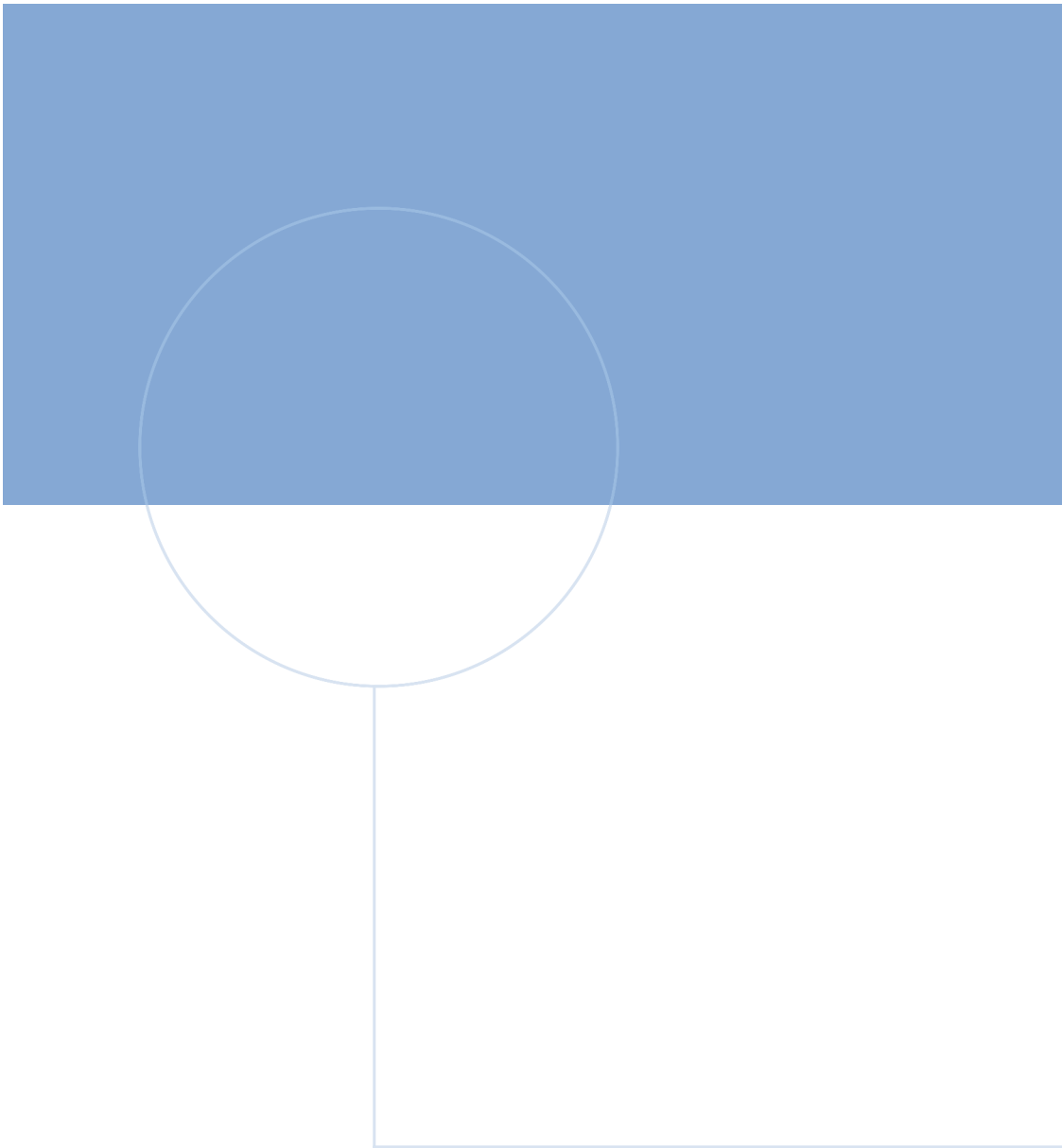
På lager

komponent	antall
Dspace maskin og kataloger	
Dspace ledninger	6
analog bro (national instrument 9237)	1
Nettverksledninger	4
umbrakoset	1
skrutrekkere div størrelser	3
strømkabel til stasjonær	
National instrument daq	1
boks med regnbukabler både M og F	2
passiv varmefordeler forran vifte	1
stabilisator stang for roboten	1
rull med multileder	1
Dspace motor controll	1
IMU-cabel arduino	1
lastcelle	1
encoder moutning to motor	4
encoder mounting directry to shaft	2
skruer til (motor mounting to house)	4
strips	30
micro SD-adapter	1
pose med skiver og skruer	1
motorer MAXON 241318 swiss made 1559291	3
beaglebone black	1
kulelagrer 6002	1
kulelager 6004-C	9
kulelager 6003 (tomme esker)	6
Flexible coupler	4
Prismatisk (pull og release) motor 19mm 12V	2
tom encodereske (scancon industrail encoder)	3
kabelstrømpe 1m	1
beaglebone case 3D printet	1
IMU cases	3
motor controller 422969	1
RJ 45 loddekontakter	4
arduino mega	1
3D case til arduino mega	1

Defekt encoder scandon encoder (2M 2F)	4
breadboard	1
IMU kretskort	1
encoderkretskort	2
Main controllboardpar	2
Servo futuba s9254	2
Servo tgy-sc340v	3
Defekt Servo tgy-sc340v	1
div små ledninger	xx
vektskive stor	1
vektskive liten	2
boks med servokomponenter	2
sparkfun levelconverter	4
IMU	1
rød boks med "servo stuff"	1
rød boks med "old elektronik"	1
sockets	2
switching power supply 5V 2A	1
stifemaskin	1
kopp med div penner	1
Defekt motor MAXON motor 148877 med gir 260551	2
Defekt BeagleBone Black	1
DFRobot SEN0373 IMU	3
Enkoder header 5025780600	2

På robot

robotrammen	
MAXON motor controll servorcontroler	2
BeagleBone Black	1
selvlaget kretskort	1
motor MAXON motor 148877 med gir 260551	2
encoderkretskort	2
IMU kretskort	2
Paralllax 900-00005 servomotor	4
IMU case	1
flexible coupler	2
Div ledninger mellom komponenter	xx
Enkoder AMT233A-V	2



NTNU

Kunnskap for en bedre verden