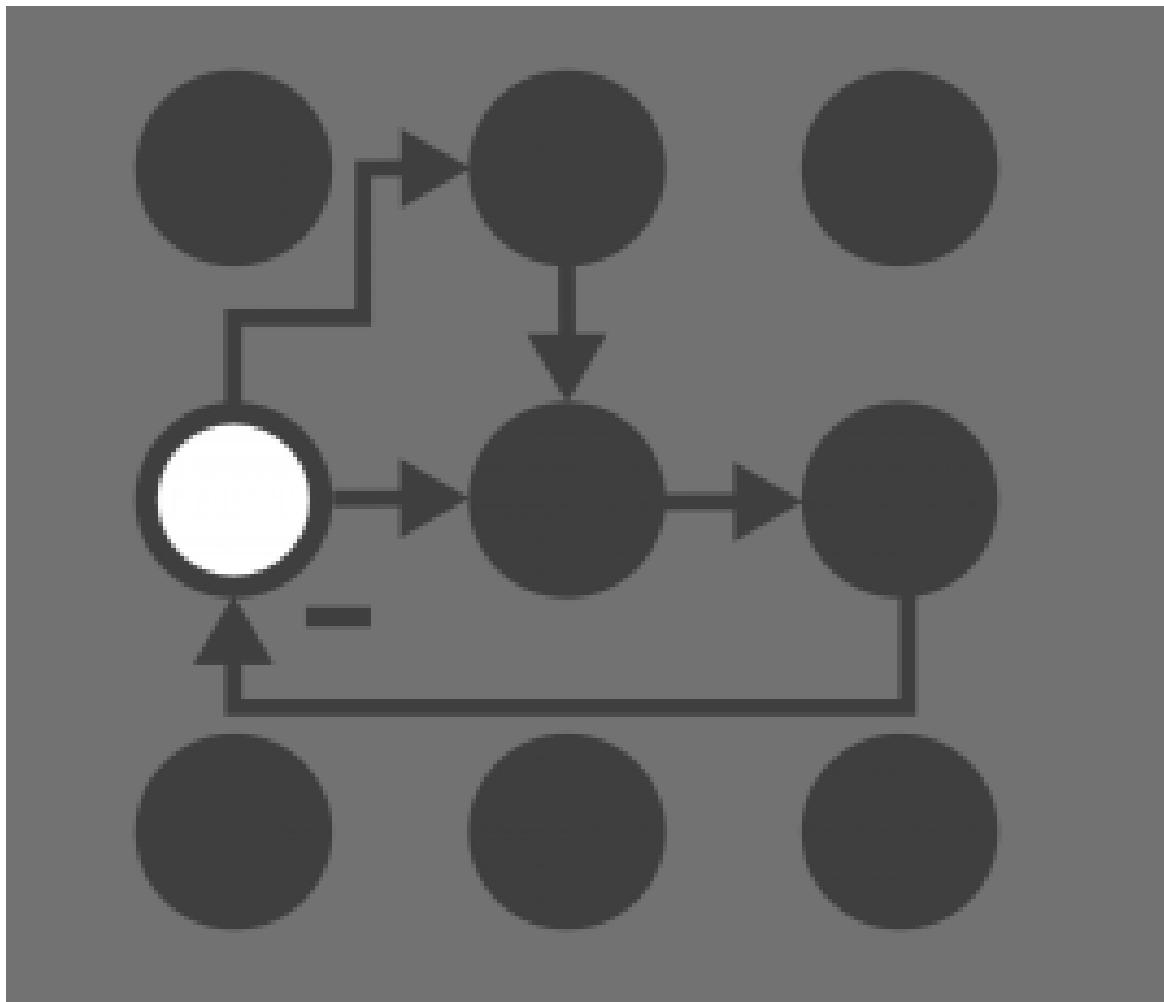


## Ros2 Robot - 3. Semesters Eksamens



*"There is a saying that any problem can be solved with enough layers of abstraction and so just like an onion or an ogre. That's what we are going to be doing with this problem. We are going to treat it as a bunch of simple smaller layers of problems that together let us solve the bigger problem more easily"*

Studerende: Henrik B

Hold: EAA\_ITEK24\_EMB

Undervisere: Kaj Norman Nielsen (KNN), Caspar Facius (CAFA) og Jonas Wehding (JWEH).

Dato: 02-08-2025

Antal tegn: 23119

Antal sider: 9,63

---

<sup>1</sup> Newans, Josh (2022) <https://www.youtube.com/watch?v=-PCuDnpqiew>

**Abstract:**

Denne rapport dokumenterer udviklingen af en ROS2-baseret robotprototype til bekæmpelse af dræbersnegle i danske haver. Systemet er bygget på en Raspberry Pi 5 og implementerer både digital og analog sensorfeedback, aktuatorstyring og signalbehandling. Projektet fokuserer på opbygning af et closed-loop kontrolsystem med brug af en PCA9685 PWM-driver, en MCP3008 ADC og en custom current measuring sensor til feedback. Rapporten gennemgår design, implementering og test af både hardware og software, med integration af ROS2 Jazzy Jalisco, hvor der arbejdes med pub/sub-kommunikation mellem noder. Der anvendes værktøjer som Docker, GitHub, SSH og Visual Studio Code. Resultatet er et funktionelt og modulært system, som danner grundlag for videreudvikling mod en fuldautomatisk og mobil løsning.

## Indholdsfortegnelse

<b>Indholdsfortegnelse.....</b>	<b>3</b>
<b>1. Indledning.....</b>	<b>5</b>
<b>2. Problemformulering.....</b>	<b>5</b>
<b>3. Indledende research.....</b>	<b>5</b>
<b>4. Blokdiagram.....</b>	<b>6</b>
<b>5. Design.....</b>	<b>7</b>
5.1. Digital Input.....	7
5.2. I <sup>2</sup> C Data og PWM.....	8
5.3. Digitalt Input (Feedback).....	9
5.4. Motorstrøm og Analogt Input.....	9
5.5. Digitalt Data - SPI (Feedback).....	10
5.6. Digitalt Output.....	11
5.7. Færdigt design.....	11
<b>6. Implementering.....</b>	<b>12</b>
6.1. 3D print.....	12
6.2. Forsyning.....	13
6.3. Digitalt input og Digitalt output.....	14
6.4. I <sup>2</sup> C Data og PWM.....	15
6.5. Digitalt Input (Feedback).....	15
6.6. Motorstrøm, Analogt Input og Digitalt Data - SPI (Feedback).....	16
6.7. Endelig Implementering.....	18
<b>7. Test.....</b>	<b>19</b>
7.1. Forsyning.....	19
7.2. Digitalt input og Digitalt output.....	19
7.3. I <sup>2</sup> C data og PWM.....	20
7.4. Digitalt Input (Feedback).....	21
7.5. Motorstrøm og analogt input.....	21
7.6. Digitalt Data - SPI (Feedback).....	26
<b>8. Kode og Algoritme Flow.....</b>	<b>27</b>
8.1. SSH.....	28
8.2 Robot Controller - Ros2 Jazzy.....	28
<b>9. Fremtidige Udvidelser.....</b>	<b>30</b>
<b>10. Konklusion.....</b>	<b>32</b>
<b>11. Materialeliste.....</b>	<b>33</b>
<b>12. Referencer.....</b>	<b>34</b>
<b>13. Bilag.....</b>	<b>35</b>
13.1. Prerequisites.....	35
13.2. Wiring RPI.....	36
13.3. Testkode.....	37
13.3.1. Testkode - Digital Input og Digital Output.....	37
13.3.2. Testkode - I <sup>2</sup> C Data og PWM.....	38

13.3.3. Testkode - Digitalt Input (Feedback).....	39
13.3.4. Testkode - Motorstrøm og Analogt Input og Digitalt Data - SPI (Feedback)..	40
13.4. SSH.....	41
13.5. Ros2 Jazzy Jalisco.....	41
13.5. Implementeret Ros2 kode.....	43
13.5.1. Feedback_pkg.....	43
13.5.1.1. Signal_node.....	44
13.5.1.2. Switch_sensor_node.....	46
13.5.1.3. Voltage_node.....	47
13.5.2. Motor_pkg.....	48
13.5.1.2. linear_actuator_node.....	50
13.5.1.2. grab_node.....	52
13.5.3. Database_pkg.....	55
13.5.1.2. indicator_node.....	55
13.5.4. Launch file.....	56
13.6. Rqt_graph install.....	57
13.7. Github.....	57
13.8. Docker.....	58
13.9. Ros2 DDS test.....	58
13.10. FastDDS Monitor.....	60
13.11. Implementerings problematik.....	60

## 1. Indledning

Denne tekniske rapport er udarbejdet som 3. semester eksamensprojekt på Aarhus Erhvervsakademi i 2025. Projektet undersøger, om en embedded løsning kan bruges til at bekæmpe dræbersnegle i danske haver. Rapporten beskriver design, implementering og test af en ROS2-baseret grib-arms robot samt anvendelsen af værktøjer som DockerHub, GitHub og SSH til kommunikation mellem SBC og PC. Derudover gennemgås, hvordan systemet bruger closed-loop feedback fra en high-side current sensor via ADC.

## 2. Problemformulering

Motivationen for en embedded løsning udspringer af egne og andres erfaringer med dræbersnegle, hvilket førte til idéen om en mere effektiv, hurtig og skånsom måde uden brug af gift og med hensyn til miljø og dyrevelfærd.

- Hvilke krav stilles der til at udvikle en robot, der kan løse denne opgave?
- Er det muligt at skabe en del af en embedded løsning, der automatisk, hurtigt og skånsomt kan aflyve dræbersnegle?

## 3. Indledende research

I løbet af et længere forløb på 3. semester blev der arbejdet intensivt med at finde en egnet løsning på problemstillingen. Undervejs skiftede den interne debat mellem at udvikle en statisk lokkefælde eller en mobil enhed. Til sidst blev beslutningen at satse på en mobil løsning, hvilket er den langsigtede plan. Den fulde implementering ligger dog udenfor dette projekts scope.

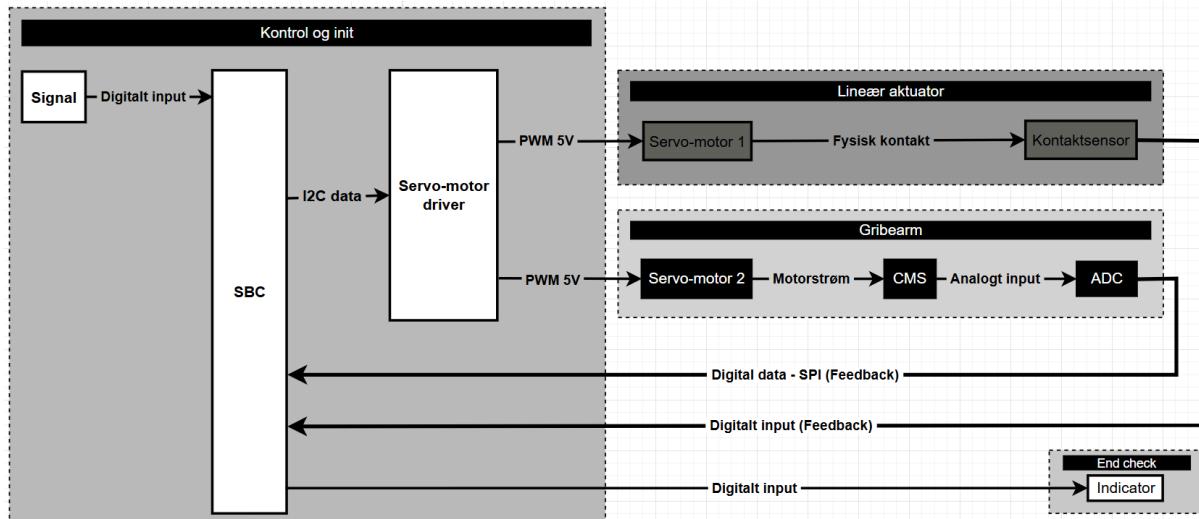
Ved et besøg hos Camro ApS, et firma der arbejder med skadedyrsbekæmpelse, oplyste CEO Torben Villadsen, at de aldrig tidligere havde udviklet eller solgt en embedded løsning på området. Han understregede dog, at en sådan løsning bestemt kunne være relevant, og at den præsenterede idé også ville kunne anvendes i andre sammenhænge – for eksempel til bekæmpelse af væggelus på hoteller, til rottekæmpelse eller andre dyr, hvor en genkendelse af dem kunne være relevant.

Undervejs blev der undersøgt forskellige muligheder, herunder simulering i Webots. Flere faglige communities og ressourcer pegede imidlertid på ROS2 som et relevant middleware framework. ROS2 tilbyder desuden integration med simuleringens miljøet Gazebo (som ligger uden for scope), hvilket gør det muligt hurtigt at teste robot-funktionalitet og opbygge relevante kompetencer inden for robotudvikling.

Under arbejdet med projektet blev begreber som closed-loop control og robot-controller identificeret som centrale faglige elementer. Dette førte til et fokus på at opnå dybdegående viden om feedback-sensing, DDS (Data Distribution Service) og package/node-strukturen i systemet og med inspiration og umiddelbar test kommer system oversigten til at indeholde følgende:

## 4. Blokdiagram

I dette afsnit beskrives systemarkitekturen og de enkelte kommunikationsveje for et automatiseret snegle bekæmpelses-system, der er baseret på en SBC, en servo-motor driver og to servo-motorer i hver deres closed-loop control system.



Figur 1: Blokdiagram af det overordnede design i udvikling af en del af en snegledræber robot

### Kontrol og init

Signal sender et digitalt input til SBC (Single-board computer) med ønske om aktivering af backend kodet applikation, hvorefter SBC sender I2C data til servo-motor driveren om at aktivere motorer, der styre i to "closed loop control" systemer - Lineær aktuator og Gribearm.

### Lineær Aktuator

Servo-motor 1 bevæger sig og aktiverer en kontaktsensor for positionerings feedback.

### Gribearm

Servo-motor 2 sender strøm til en CMS (current measuring sensor), som sender analogt signal til ADC, som derefter sender digitalt signal til SBC.

### End check

Til slut sendes en digitalt output til indikator fra SBC for at signalere at opgaven er fuldført.

Komponent	Input	Output
Signal	Digital input 3.3V til SBC	
SBC	Digital input 3.3V fra Signal Digital input 3.3V fra kontaktsensor	I <sup>2</sup> C til servomotor driver Digital output 3.3V til Indikator

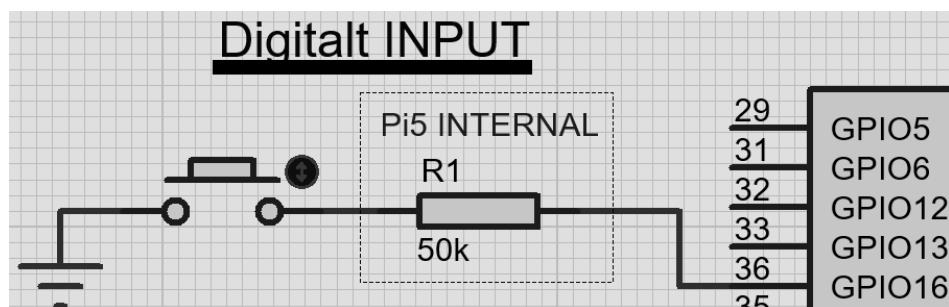
	Digital data (SPI) fra ADC	
Servo-motor Driver	I <sup>2</sup> C fra SBC	Motorstrøm - PWM-moduleret, 5V til servo-motor 1 og 2.
Servomotor 1	PWM 5V moduleret fra servomotor driver	Fysisk kontakt med kontaktsensor
Servomotor 2	PWM 5V moduleret fra servomotor driver	Motorstrøm til CMS
CMS	Motorstrøm fra servo-motor	Analog feedback til ADC
ADC	Analog feedback fra CMS	Digital data (SPI) til SBC
Indicator	Digital output 3.3V fra SBC	

## 5. Design

### 5.1. Digital Input

Til at generere startsignalet for systemet anvendes en simpel push button, som er direkte forbundet til en GPIO-pin på en Raspberry Pi 5B (Pi5 fremover). Når knappen aktiveres, sendes et digitalt signal til Pi5, hvilket fortsætter den igangværende applikationen på Pi5. Trykknappen i sig selv sender en spænding på 0 V eller 3,3 V.

- Når knappen er trykket ind: GPIO-pin får forbindelse til GND (signal = LOW, 0).
- Når knappen ikke er trykket: GPIO-pin er høj (signal = HIGH, 1).

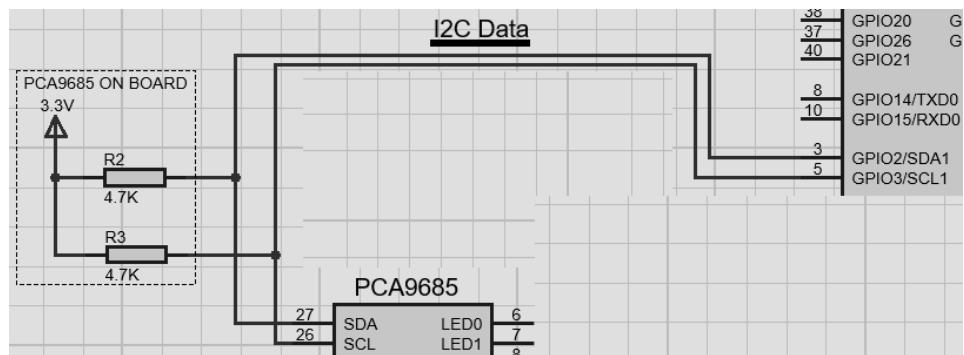


Figur 2: Elektrisk diagram af digital input signal til Pi5

Push button-løsningen er valgt som simulation i denne sammenhæng. Denne signal-metode er let at udskifte eller udvide, hvis systemet senere skal modtage signal fra eksempelvis sensorer eller kameraer.

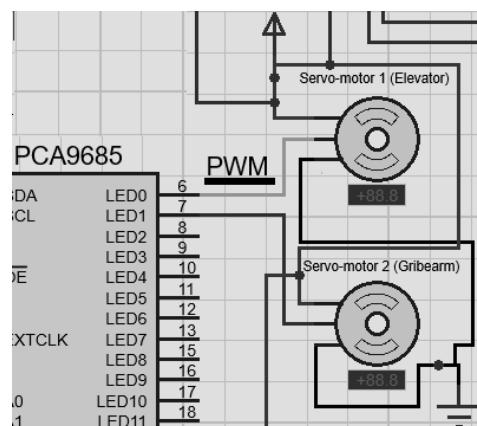
## 5.2. I<sup>2</sup>C Data og PWM

I systemet blev I<sup>2</sup>C valgt som kommunikationsform til styring af servomotorer via en ekstern PWM-controller PCA9685 (fremover PCA). Selvom Pi5 tilbyder hardware PWM, var oplevelse efter noget tids eksperimenter at denne løsning havde begrænsede og ustabile software biblioteker til den relativt nye Pi5 (2023), hvilket gjorde det vanskeligt at opnå styring direkte, da overlayet ikke fungerede som de undersøgte biblioteker lagde op til, til forskel fra tidligere modeller som Pi4, som kører et simpel overlay i config.txt filen. PCA løser dette ved at overtage genereringen af hardware PWM-signaler, hvilket sikrer stabil og præcis bevægelse af servoerne. Det aflaster Pi5, minimerer jitter, som ses tydeligt ved software PWM.



Figur 3: Elektrisk diagram af I<sup>2</sup>C fra Pi5's SDA og SCL til PCA.

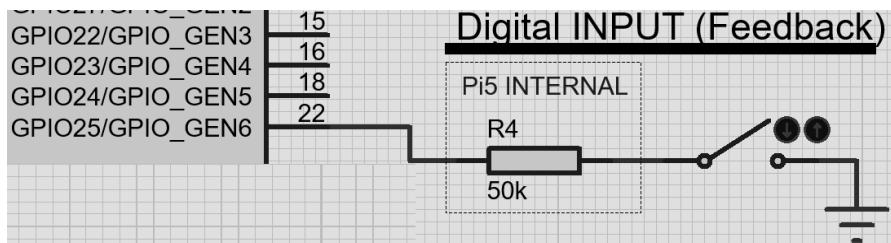
PCA-modulet tilbyder 16 uafhængige PWM-kanaler med 12-bit opløsning (4096 tællerskridt) og en programmerbar udgangsfrekvens fra  $\approx 24$  Hz til  $> 1,5$  kHz, hvilket komfortabelt omfatter de 50 Hz, som systemets valgte servo-motor en MG995-servoen kræver. Det tilhørende Circuit Python-bibliotek abstraherer I<sup>2</sup>C-kommunikationen og muliggør præcis konfiguration af prescale-register, puls bredde og duty-cycle med minimal kode-overhead. Samlet set giver PCA en robust, skalerbar og veldokumenteret platform til servo-drift på Pi5.



Figur 4: Elektrisk diagram af PWM fra PCA9685 til servo-motorer

### 5.3. Digitalt Input (Feedback)

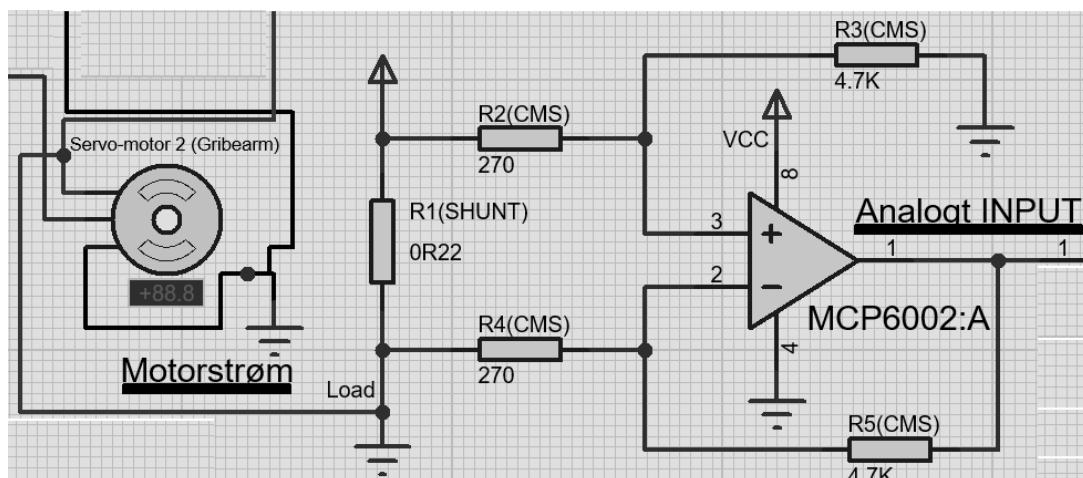
I systemet anvendes en simpel kontaktsensor som reference sensor for lineær aktuator servo-motor (Servo 1). Kontakten placeres i toppen af lineær aktuators bevægelsesområde, således, at når servo'en løfter den lineære aktuator op og rammer kontakten, sendes et digitalt signal til Pi5. Dette signal bruges som referencepunkt for servo'en, så systemet altid ved hvor top position er. Når kontakten aktiveres, stoppes servo-motoren øjeblikkeligt for at undgå mekanisk overbelastning og for at sikre præcis positionering.



Figur 5: Elektrisk diagram af den fysiske kontakt og det digitale input feedback.

### 5.4. Motorstrøm og Analogt Input

Motorstrøm i servo-motor 2 er direkte relateret til den modstand, motoren oplever under drift. Når belastningen overstiger en foruddefineret tærskel, skal servo-motoren stoppes for at undgå skade på både motoren og mekaniske komponenter, som den påvirker. Endvidere vil modstand ligeledes betyde, at der er opfangeret et objekt.



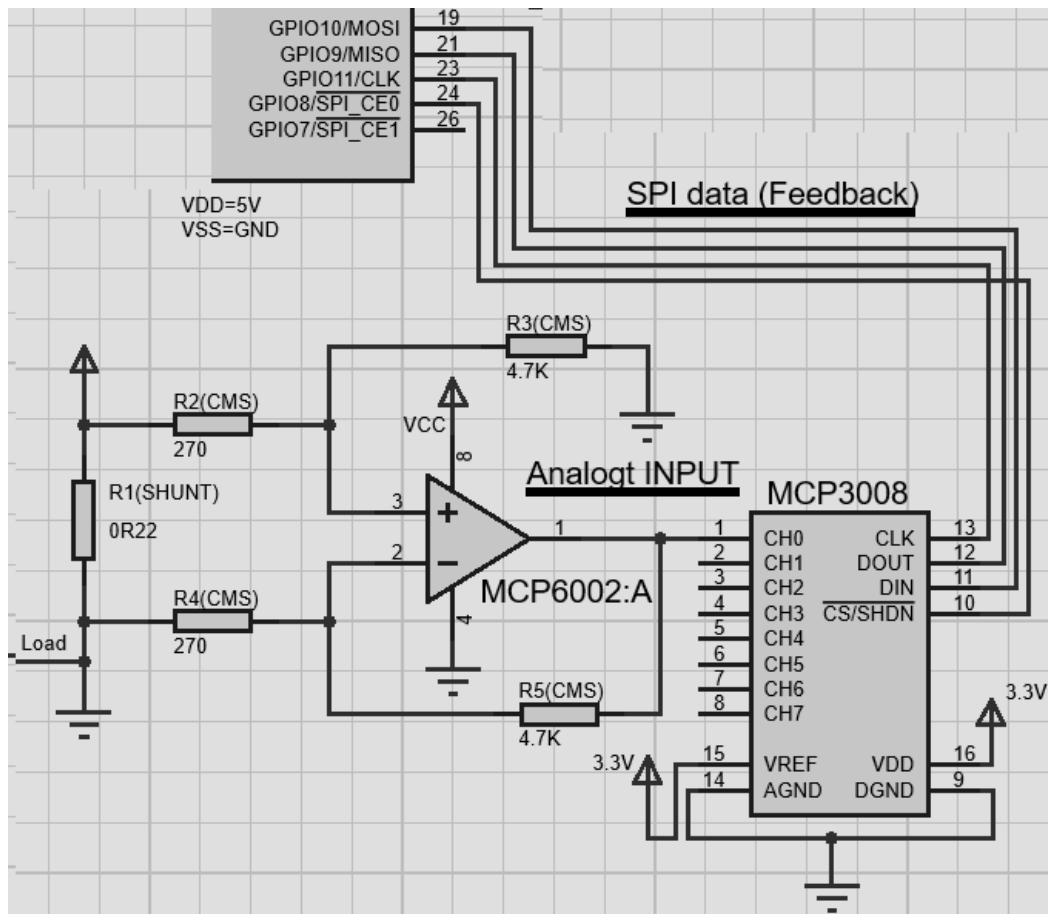
Figur 6: Elektrisk diagram af servo-motor 2 og forbindelsen til CMS.

I dette current measuring sensor (high-side CMS) kredsløb konverteres strømmen gennem belastningen til en spænding ved hjælp af en shuntmodstand, der er placeret mellem forsyningsspændingen og belastningen. Når strømmen løber gennem shuntmodstanden, skabes der et lille spændingsfald proportionalt med strømmen. Denne spænding er meget lav, og derfor anvendes en operationsforstærker i dette system, en MCP6002, til at forstærke spændingen, så signalet kan redefineres inden for et givent spektrum, der giver et analogt output.

## 5.5. Digitalt Data - SPI (Feedback)

Pi5 er ikke udstyret med indbyggede ADC-kanaler og kan derfor ikke direkte aflæse analoge spændinger, såsom udgangen fra CMS. For at muliggøre digital indsamling af analoge signaler blev der derfor integreret en ekstern ADC-chip i systemdesignet.

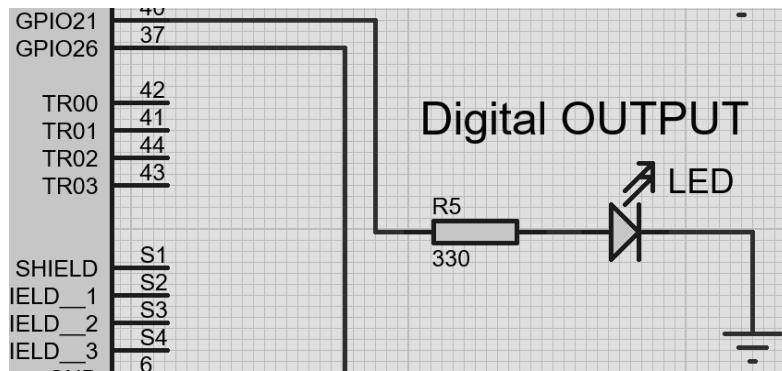
Valget faldt på MCP3008, da denne ADC-chip har otte analoge indgange, 10-bit oplosning og kommunikerer direkte via SPI-protokollen, som understøttes af Pi5 GPIO-header. Dette valg sikrer fremtidige udvidelsesmuligheder og en simpel integration i det eksisterende system.



Figur 7: Elektrisk diagram af CMS analogt input til MCP3008 og forbindelsen til Pi5 SPI

## 5.6. Digitalt Output

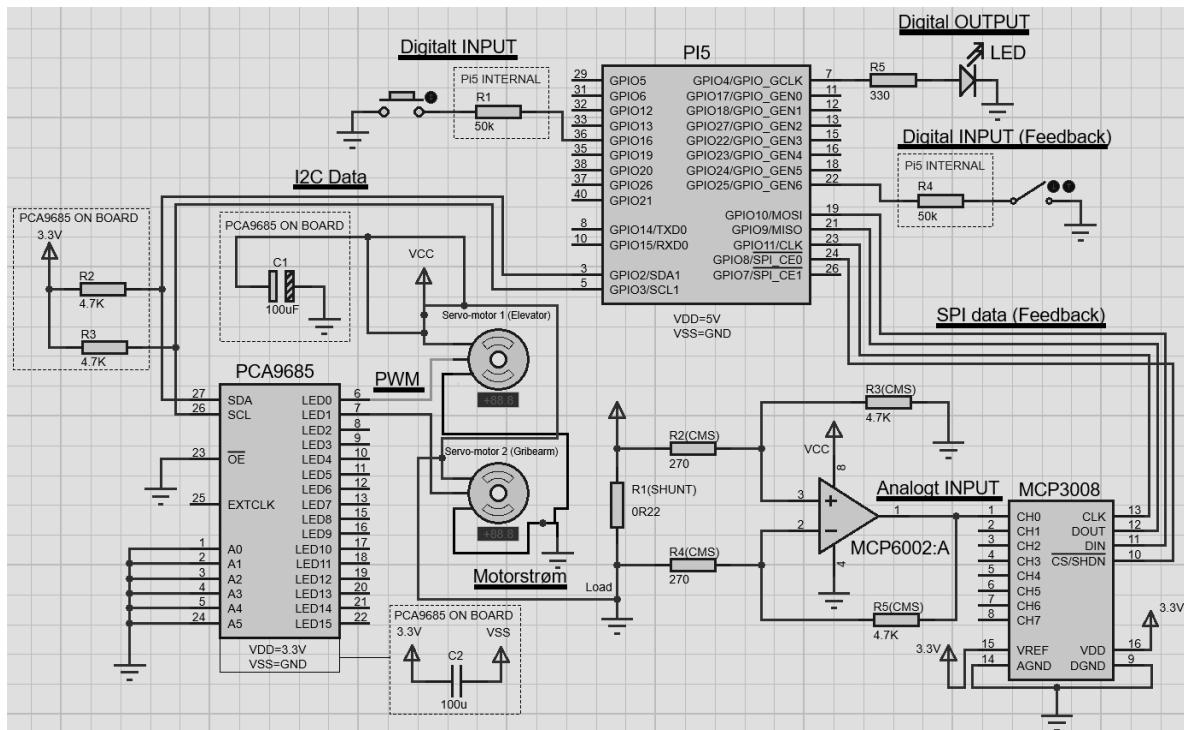
Det digitale output blev integreret med en simpel LED, for at signalere at systemets kørsel er fuldendt, dette kunne også være at sende data til en database for at registrere kørsel eller sende signalet videre til anden funktionalitet.



Figur 8 : Elektrisk diagram af det digitale output til LED

## 5.7. Færdigt design

Det samlede system integrerer styring og overvågning af to servo-motorer via en PCA9685 PWM-driver samt måling af motorstrøm gennem et high-side current sensing kredsløb. Alle signaler samles og behandles af en Pi5, som kommunikerer med de eksterne moduler via I<sup>2</sup>C og SPI. Systemet inkluderer både digitale input- og output signaler for brugerinteraktion og statusvisning.

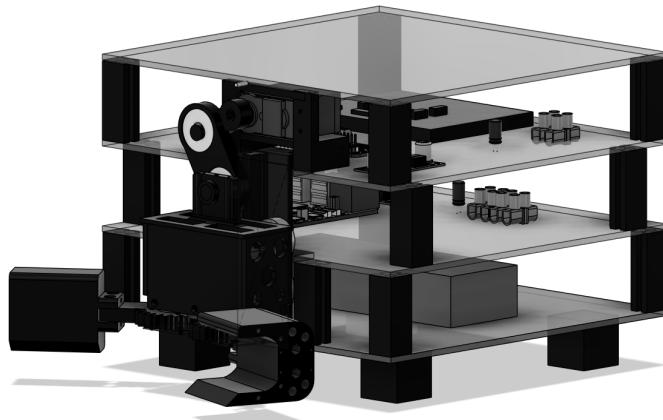


Figur 9 : Elektrisk diagram af det fulde system i proteus.

## 6. Implementering

### 6.1. 3D print

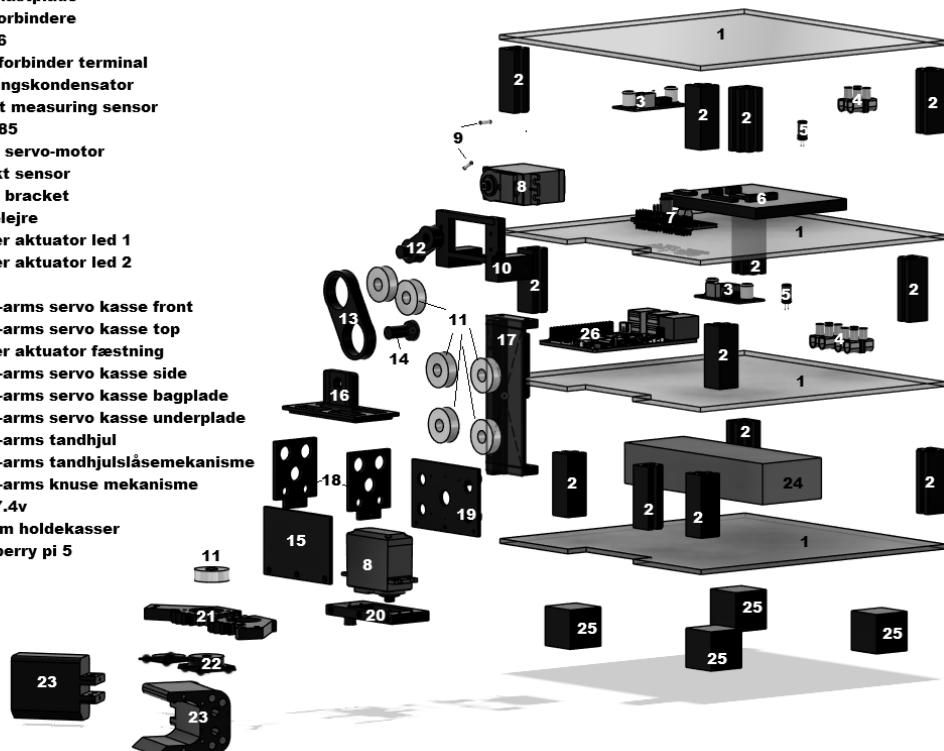
Det færdige 3D printede design i fusion 360, hvor den lineære aktuator, som er styret af en servo-motor, kører en gribearm op og ned, gribearmen lukker, når den lineære aktuator har sendt den ned.



Figur 10 : Systemet designet i proteus

I fusion 360 ses her et exploded view af systemet med betegnelse af de enkelte dele, så de er lettere at identificere i STL biblioteket. se: <https://github.com/Skradfy/STL-bibliotek>

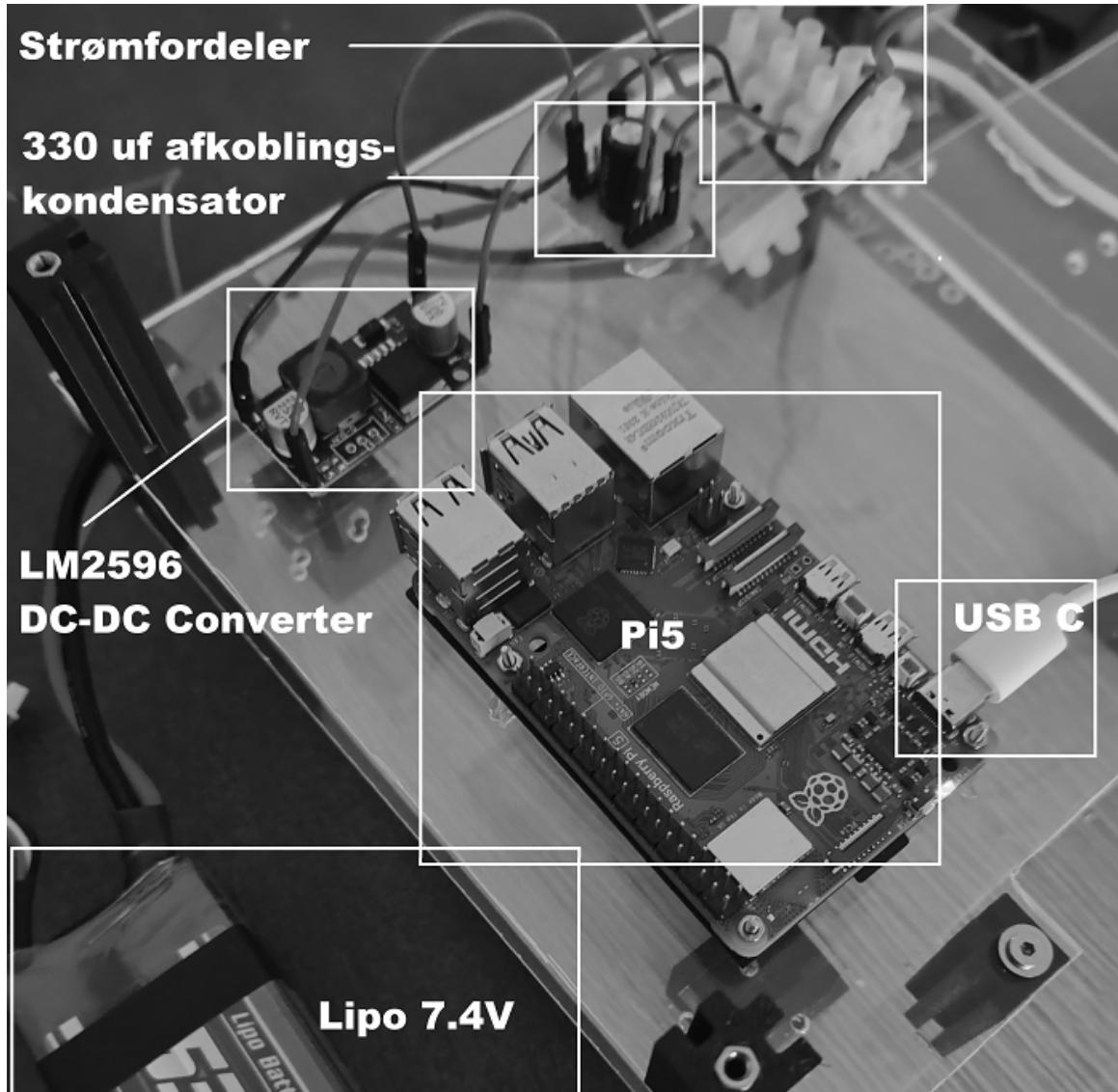
- 1. Akryl plastplade
- 2 Etage-forbindere
- 3. LM2596
- 4. Strøm-forbindende terminal
- 5. Afkoblingskondensator
- 6. Current measuring sensor
- 7. PCA9685
- 8. MG995 servo-motor
- 9. Kontakt sensor
- 10. Servo bracket
- 11. Kuglelejre
- 12. Lineær aktuator led 1
- 13: Lineær aktuator led 2
- 14: Lås
- 15: Gribearms servo kasse front
- 16: Gribearms servo kasse top
- 17: Lineær aktuator fæstning
- 18: Gribearms servo kasse side
- 19: Gribearms servo kasse bagplade
- 20: Gribearms servo kasse underplade
- 21: Gribearms tandhjul
- 22: Gribearms tandhjulslåsemekanisme
- 23: Gribearms knuse mekanisme
- 24. Lipo 7.4v
- 25. System holdekasser
- 26. Raspberry pi 5



Figur 11 : Exploded view i proteus

## 6.2. Forsyning

Forsyning bliver et af de centrale temaer i de kommende iterationer af systemet. Derfor blev der arbejdet med at tilslutte systemet til batteri, da det på sigt skal fungere som en mobil løsning. Til dette formål anvendes en separat LM2596 DC-DC Converter, som sikrer, at Pi5 får den korrekte spænding via den mere sikre USB-strømforsyning indgang.

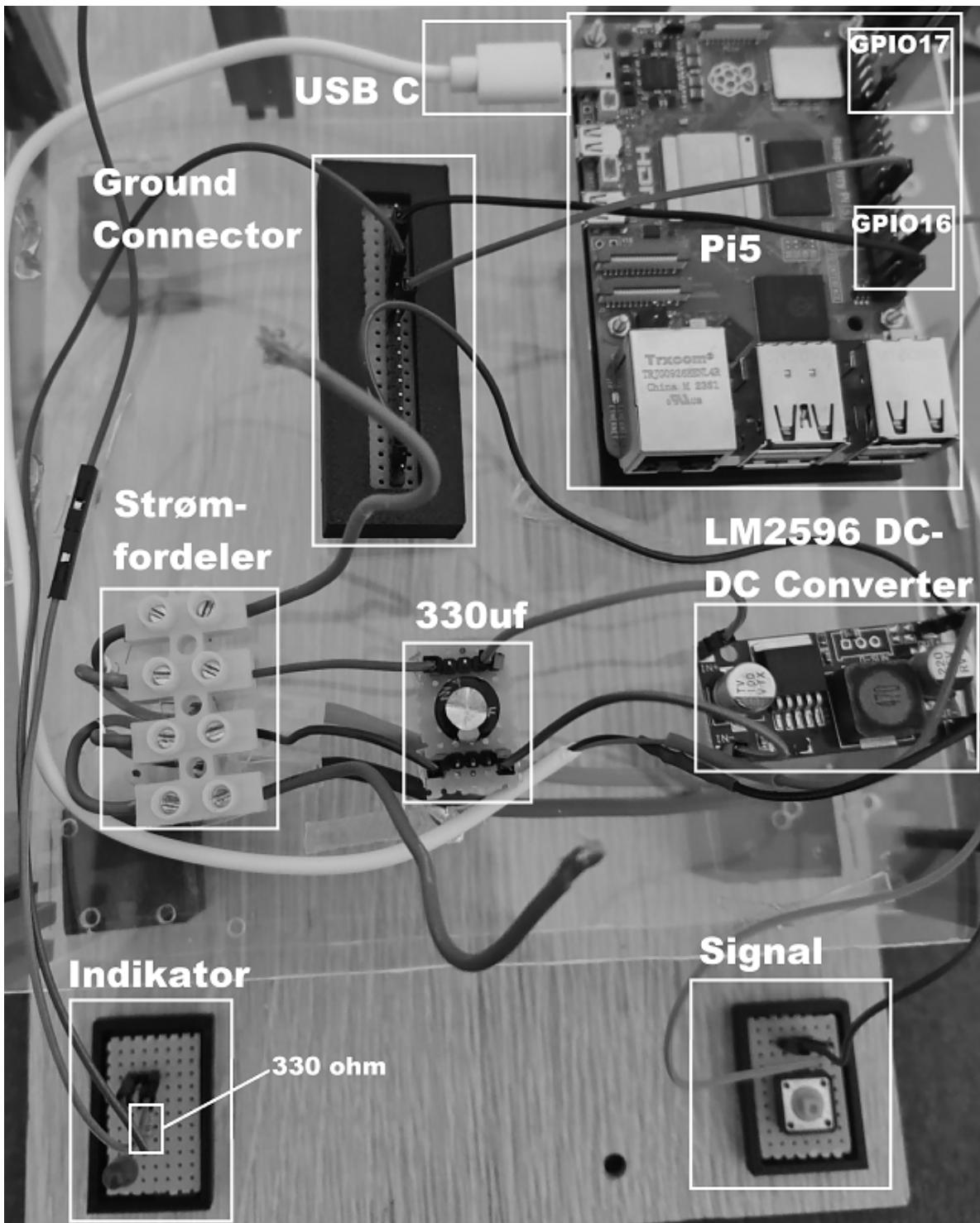


Figur 12 : Implementering af forsyning med DC-DC converter

On-board LED skal lyse grønt for at Pi5 booter op. Oplevelsen var, at hvis man kører alle 5V komponenter med Pi5, så ville Pi5 ikke boote og LED forblev rød. Det blev derfor besluttet at have to LM2596 i systemet, hvilket senere viste sig at være et problem, da LM2596 kun leverer 2A. Se bilag 13.11 Implementerings problematik.

### 6.3. Digitalt input og Digitalt output

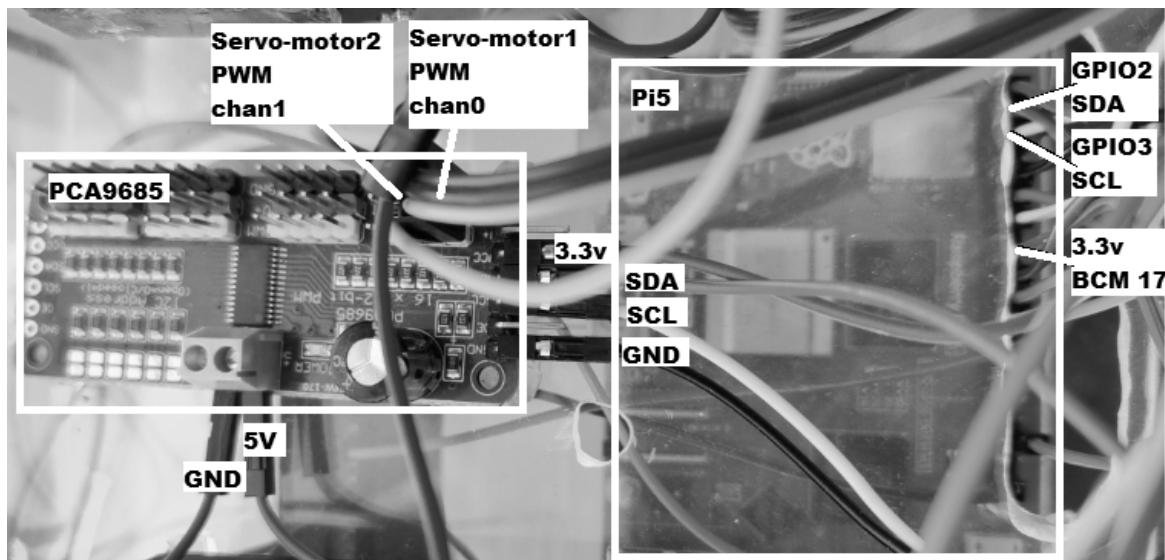
Pi5 implementeret med signal og indikator forbundet til GPIO17 og GPIO18. Indikator fik tilkoblet en 330 ohm modstand for at beskytte LED.



Figur 13 : Implementering af forsyning med DC-DC converter

## 6.4. I<sup>2</sup>C Data og PWM

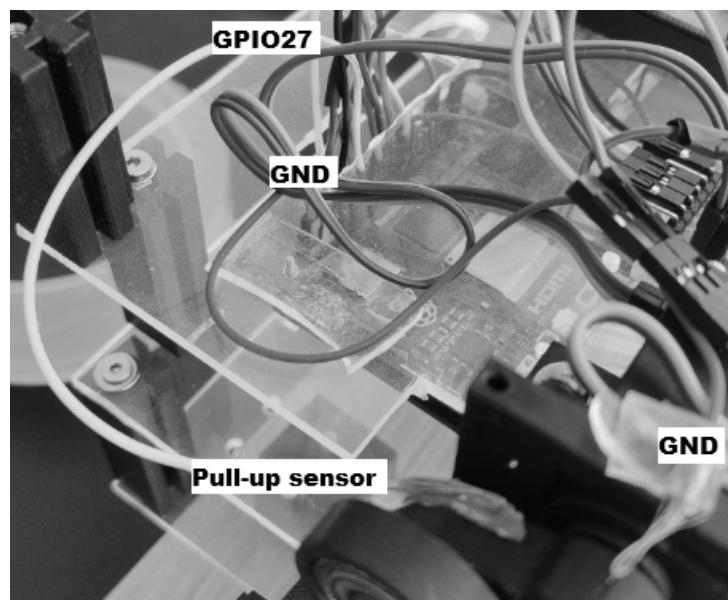
PCA forbundet til Pi5 via I<sup>2</sup>C, servo-motor 2 (gribearm) er forbundet til chan1 med ground, signal og 5v, men hvor 5v ledningen er delt i to, så servo-motor både får strøm, men også kan aflæses med CMS.



Figur 14 : Implementering af servo-motor til PCA9685 med I<sup>2</sup>C forbundet til Pi5

## 6.5. Digitalt Input (Feedback)

Simpel kontaktsensor implementeret for at servo-motor 1 har en dobbelt sikring i at give feedback til Pi5, top positionen kan allokeres.

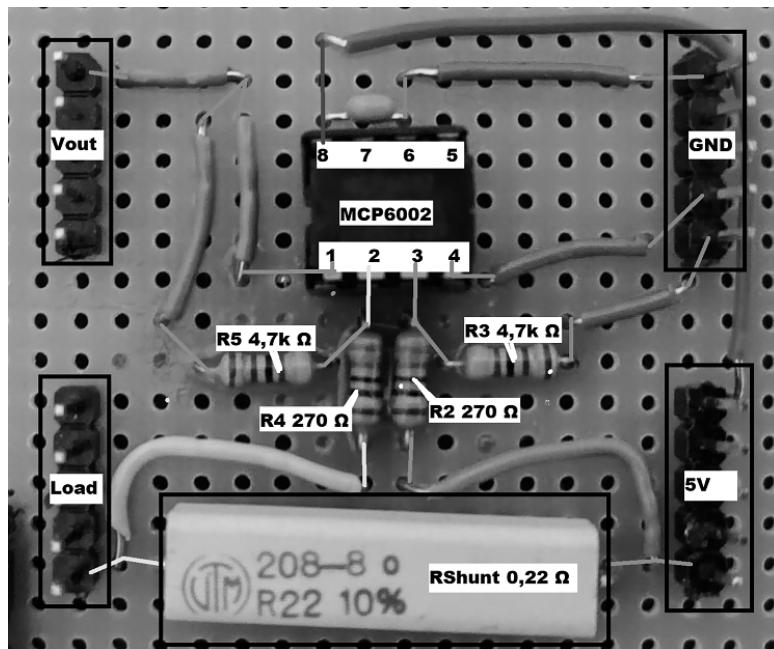


Figur 15 : Implementering af simpel kontaktsensor

## 6.6. Motorstrøm, Analogt Input og Digitalt Data - SPI (Feedback)

For at måle belastningen på servo-motor 2 (gribearm) blev der konstrueret en high-side current measuring sensor. En switch sensor kunne også have været anvendt, men da der kunne opsamles ukorrekte objekter på et tidligt stade i en mulig sneglegengenkelsens YOLO model, blev denne valgt og senere kan der ligeledes implementeres en switch sensor for at give feedback om at nu er gribearm lukket sammen.

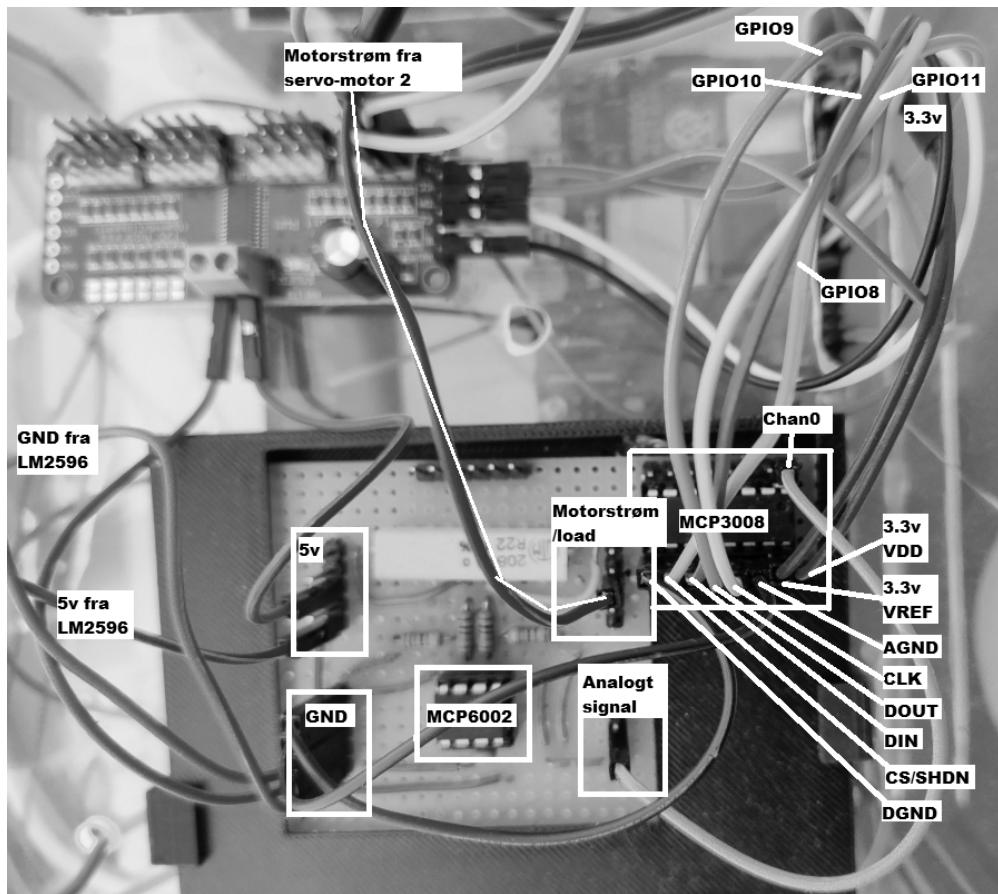
- 5V-forsyningen bruges både til at drive MCP6002 operationsforstærkeren og som referencespænding, som føres gennem en shuntnødstand på 0,22 ohm.
- GND fungerer som fælles nulpunkt. Her er der lavet fem GND-headers, så der er mulighed for flere jordforbindelser.
- Load tilsluttes servo-motor 2, så sensoren kan måle, hvor meget strøm motoren trækker under belastning.
- Vout fra sensoren føres videre til MCP3008 ADC'en, som omdanner det analoge signal til digitalt, så det kan læses af Raspberry Pi 5.



Figur 16 : Implementering af motorstrøm signal på perfboard

Motorstrøm fra servo-motor 2 føres til MCP6002, som måler spændingsforskellen mellem 5V og det strøm, der går gennem "Motorstrøm/load".

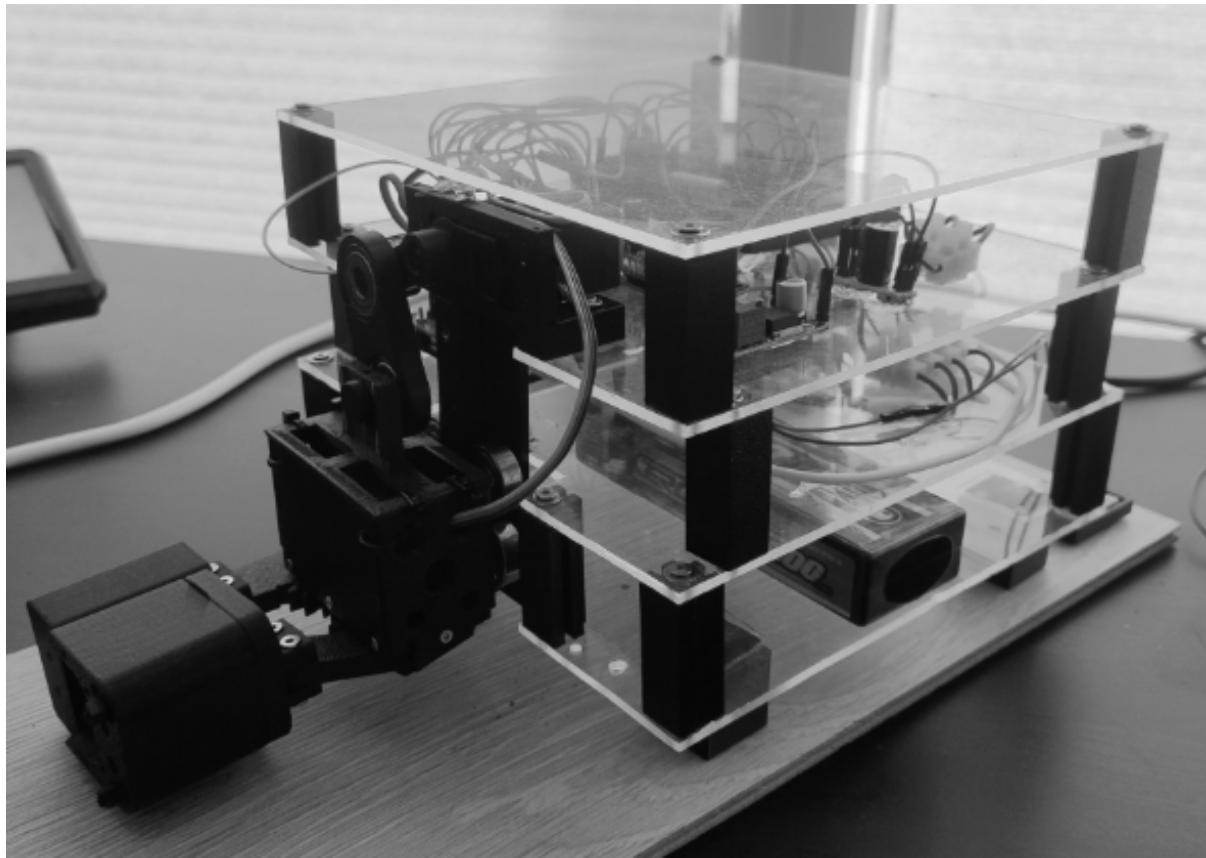
- MCP6002 omdanner denne forskel til et analogt signal, der afspejler hvor meget strøm motoren trækker.
- Det analoge signal sendes derefter til MCP3008, som konverterer signalet til digitalt.
- Via SPI-protokollen sendes det digitale feedback-signalet videre til Pi5.



Figur 17 : Implementering af motorstrøm til CMS som sender analogt signal til MCP3008 og videre til Pi5

## 6.7. Endelig Implementering

Den endelige implementering af systemet uden indsatte vægge sat på de midlertidige system-holde kasser.

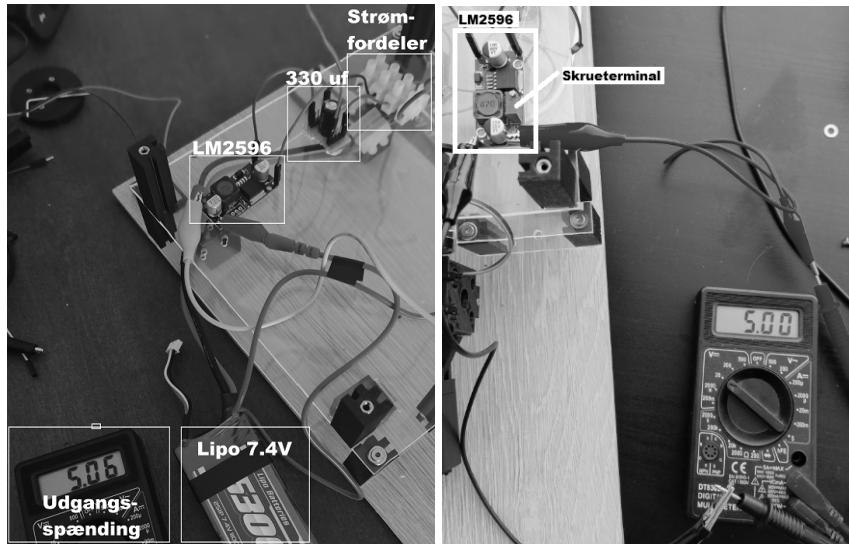


*Figur 18 : Endelige implementering af snegledræber systemet version alpha*

## 7. Test

### 7.1. Forsyning

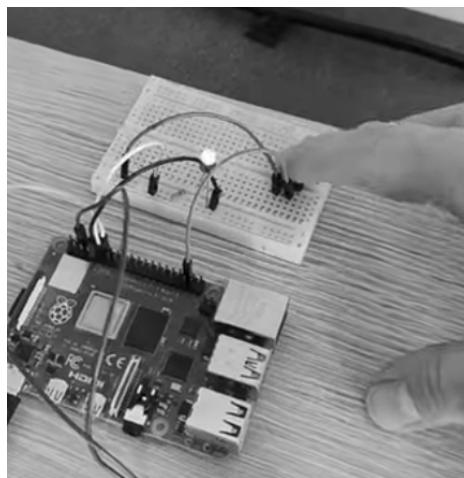
Test af forsyning er essentielt inden systemet tilsluttes, Pi5 skal have 5v stabilt med op til 3A strøm, så der ikke forekommer brown-out. Lipo 7,4v har en kapacitet på 5300 mAh.



Figur 19 : Test af begge LM2596 DC-DC converter fra Lipo 7.4v til ~5v stabilt

### 7.2. Digitalt input og Digitalt output

Test af signal og indikator blev foretaget med en exception handler, da GPIO pin ellers ville være optaget til næste test og det kræver en manuel kill process at frigøre GPIO igen.



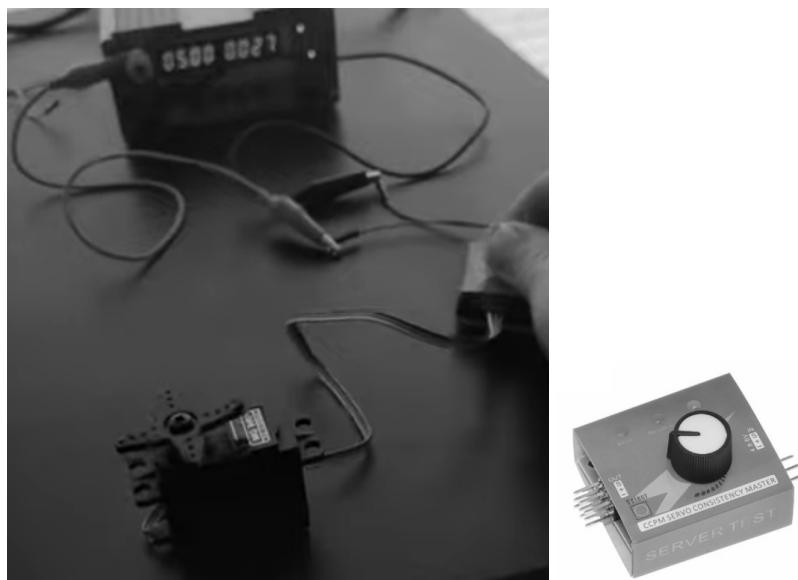
Figur 20 : Test af signal og indikator på et breadboard

Test video: <https://www.youtube.com/shorts/AXZIH-P8fgs>

Testkode: Se bilag 13.3.1

### 7.3. I<sup>2</sup>C data og PWM

For at teste servo-motorer og indstille dem korrekt blev denne servo-tester anvendt. Fordelen ved at anvende den er at servo-motorerne bliver lettere at indstille og kalibrere senere.



Figur 21: Servo-tester anvendt inden opsætning med PCA9685



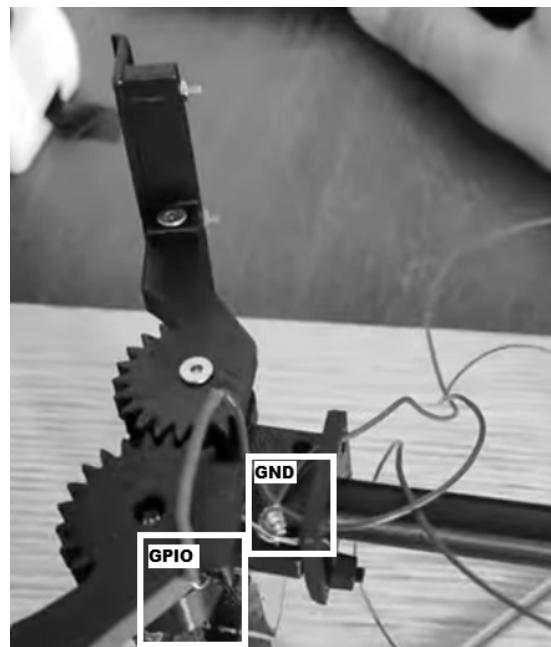
Figur 22: Servo-motor test med PCA9685

Test video: <https://www.youtube.com/shorts/Z49q3ElHybw>

Testkode: Se bilag 13.3.2

## 7.4. Digitalt Input (Feedback)

For at lave en test af kontaktsensor, blev der lavet en hurtig test med en SG90 servo og en Raspberry Pi 4. Testkode: Se bilag 12.2.



Figur 23 : Test af kontaktsensor

Testvideo: <https://www.youtube.com/shorts/nqLQ9tqrxFNQ>

Testkode: Se bilag 13.3.3

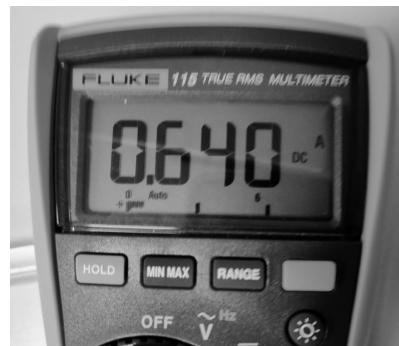
## 7.5. Motorstrøm og analogt input

Når motorer kører uden belastning vil strømtræk se nogenlunde således ud:



Figur 24 : Test strøm ved kørsel uden belastning af en SG90

Når en SG90 servomotor sættes i stall så vil strømmålingen se nogenlunde sådan ud:



Figur 25: Måling af strøm ved kørsel med belastning af en SG90

For at muliggøre feedback til Pi5, så den kan måle strømforbruget, blev der konstrueret en high-side current measuring sensor (CMS), som tidligere beskrevet. Ved hjælp af formlen fra designskemaet<sup>2</sup> (side 2) til beregning af udgangsspændingen ( $V_o$ ) er der udarbejdet en oversigt over det teoretisk forventede spændingsoutput.

$I_{in}$ i A	$R_1$ (shunt) i $\Omega$	$R_5$ i $\Omega$	$R_4$ i $\Omega$	$V_o$ i volt
0,1	0,22	4700	270	0,382962963
0,2	0,22	4700	270	0,7659259259
0,3	0,22	4700	270	1,148888889
0,4	0,22	4700	270	1,531851852
0,5	0,22	4700	270	1,914814815
0,6	0,22	4700	270	2,297777778
0,7	0,22	4700	270	2,680740741
0,8	0,22	4700	270	3,063703704

Figur 26: Tabel over  $V_o$  i relation til strømmens load -  $I_{in}$

Der blev simuleret med modstande i proteus, hvor load er modstanden ud fra denne formel:

$$I = V_{shunt}/R_{shunt}$$

For at ramme en omtrentlig strøm på 0,2A udregnes:

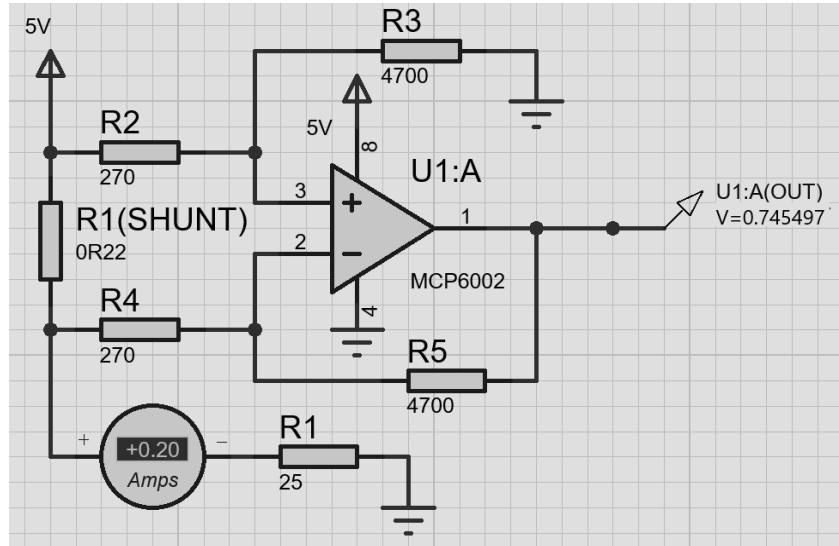
$$V_{shunt} = 5 \text{ Volt}$$

$$R_{shunt} = 25 \text{ ohm}$$

$$I = 5 / 25 = 0,2\text{A}$$

<sup>2</sup> <https://www.ti.com/lit/an/sboa310b/sboa310b.pdf?ts=1753097926881>

Dette svarer nogenlunde til drift uden ydre påvirkning, forventes kun en lille belastning, når servo-motorerne bevæger gribearmen og hermed ses et output på 0,74v



Figur 27: Simulering som viser den teoretiske "no load" strøm på 0,2A med et output på 0,74 v

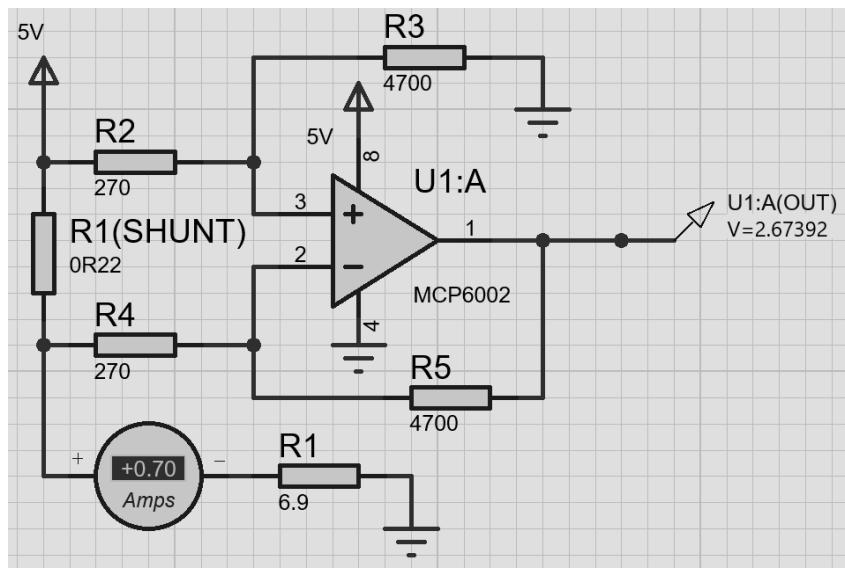
For at have en ide om en teoretisk stall blev der ligeledes testet i proteus og her med en modstand på 6,9 ohm for at få en 0,7A strøm:

$$V_{shunt} = 5 \text{ Volt}$$

$$R_{shunt} = 6,9 \text{ ohm}$$

$$I = 5 / 6,9 \approx 0,7A$$

Dette svarer til drift omkring stall punktet og der gives et output på 2,67v, så for at få et feedback om gribearmen har mødt modstand og man derfor kan konstatere at der er snegl



Figur 28: Simulering som viser den teoretiske "load" strøm på 0,7A med et output på 2,67 v

Forstærker Gain:

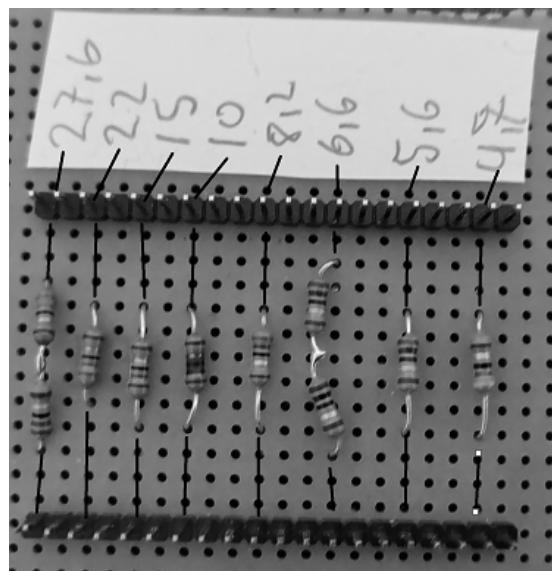
$$G=R_2/R_1$$

$$G=4700\Omega/270\Omega \approx 17,407$$

Overførsels Faktor volt/ampere:

$$\text{Volt pr. ampere} = G * R_{\text{shunt}} = 17,407 * 0,22 \approx 3,83 \text{ V/A}$$

Efterfølgende blev der lavet en lille test med forskellige modstande for at se hvad udfaldet ville blive i virkeligheden. Til dette blev der designet en hurtig modstandstabel for at få påvist de faktiske målinger med en nogenlunde tilsvarende modstand i forhold til skemaet fra figur 24:

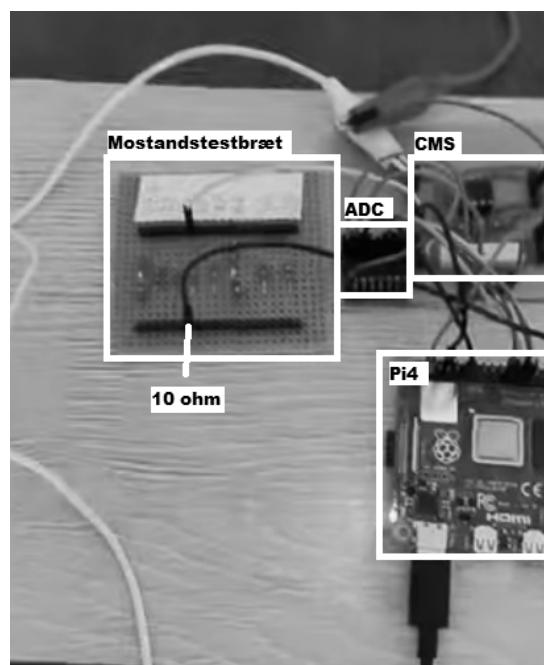


Figur 29: Test-prototype board for at teste kredsløbets præcision

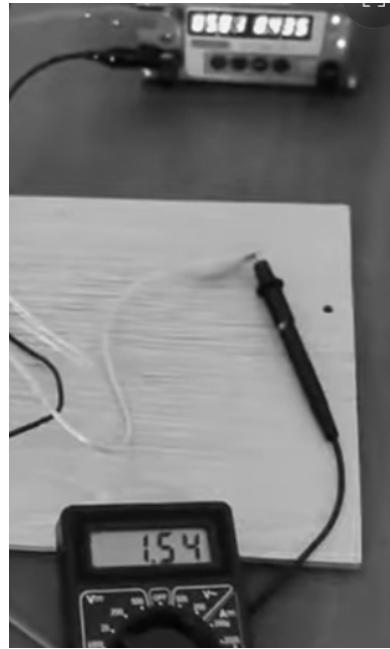
I det følgende skema vises test målingerne af den konstruerede CMS at 0,22A giver en  $V_o$  på 0,78V - Den teoretiske 0,2A giver 0,76V og 0,69A giver  $V_o$  på 2,7V - Den teoretiske 0,7A giver 2,68V, så kredsløbet ligger ikke langt væk fra det teoretiske og anses for anvendeligt til dette projekt.

Kredsløb		
R1 (ohm)	strøm (amps)	spænding
27,6	0,18	0,43
22	0,22	0,78
15	0,31	1,16
10	0,45	1,54
8,2	0,54	1,93
6,6	0,64	2,5
5,6	0,69	2,7
4,7	0,85	3,2

Figur 30: Målt output på kredsløbet ud fra test-prototype board -



Figur 31: Testopsætning for analogt signal ved måling af 10 ohms modstand



Figur 32: Den målte strøm og spænding fra cms med 10 ohms modstand

## 7.6. Digitalt Data - SPI (Feedback)

For at lave en test på cms, blev anvendt testkode - Denne kode læser MCP3008's kanal 0 og printer spændingen direkte til terminalen.

10	0,45	1,54
----	------	------

I testen kan man se at outputtet svarer til skemaet - 10 ohms modstand, 0,45 Ampere og 1,54 volt.

```
0 Spænding: 0.000 V
0 Spænding: 0.000 V
489 Spænding: 1.548 V
489 Spænding: 1.545 V
479 Spænding: 1.545 V
478 Spænding: 1.542 V
479 Spænding: 1.545 V
479 Spænding: 1.542 V
478 Spænding: 1.542 V
```

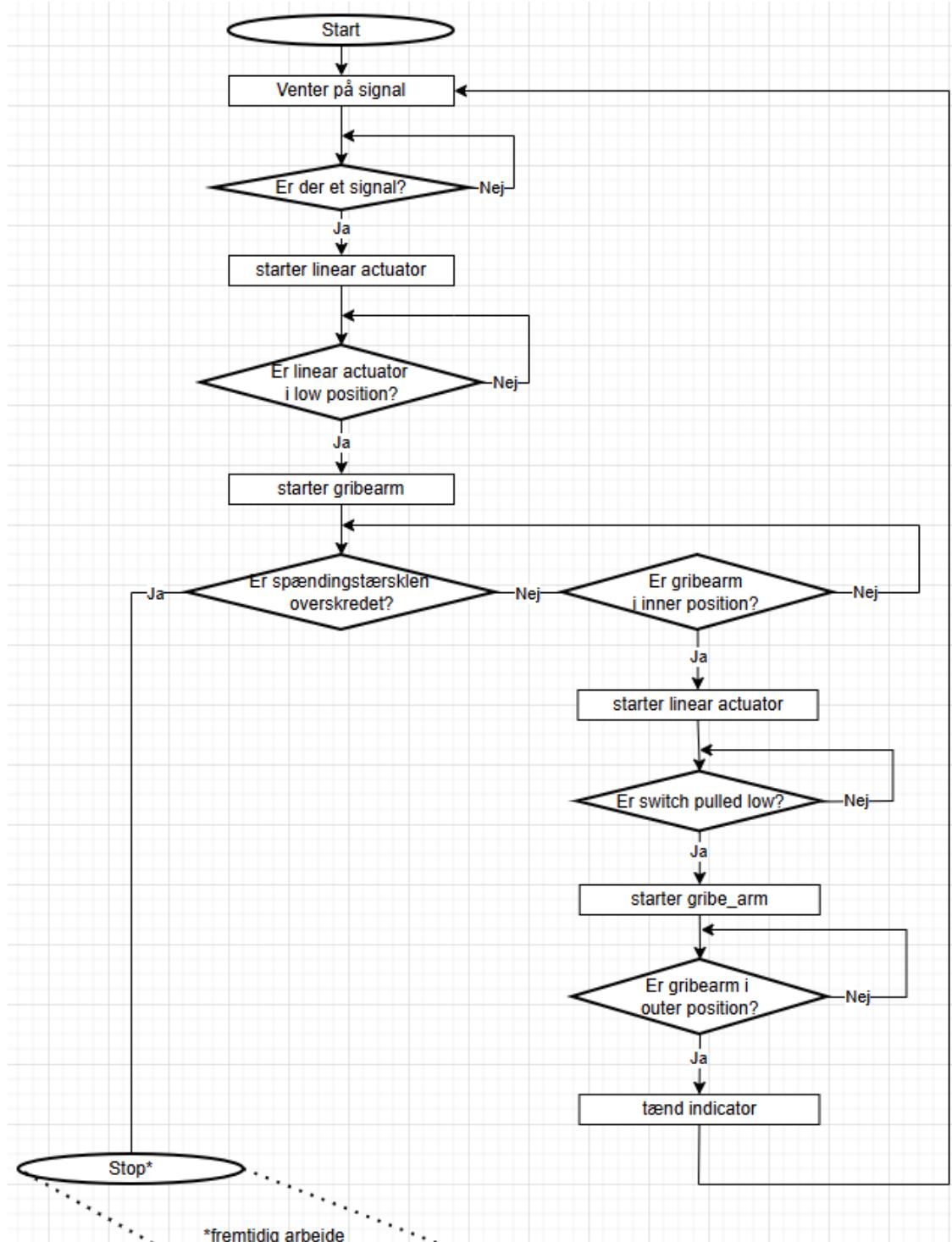
Figur 33: ADC output i terminalen

Testvideo: <https://www.youtube.com/shorts/vN-TK6jfSMQ>

Testkode: Se bilag 13.3.4

## 8. Kode og Algoritme Flow

Her ses det udviklede flowdiagram, som på dette stade ikke har en løsning på hvad der skal ske efter at spændings tærsklen er overskredet.



Figur 34: Flowdiagram for systemet

### Kort flow overblik:

Tryk → Servo1 ned → Servo2 luk → måling → Servo1 op → Servo2 åbn → LED tændt → reset

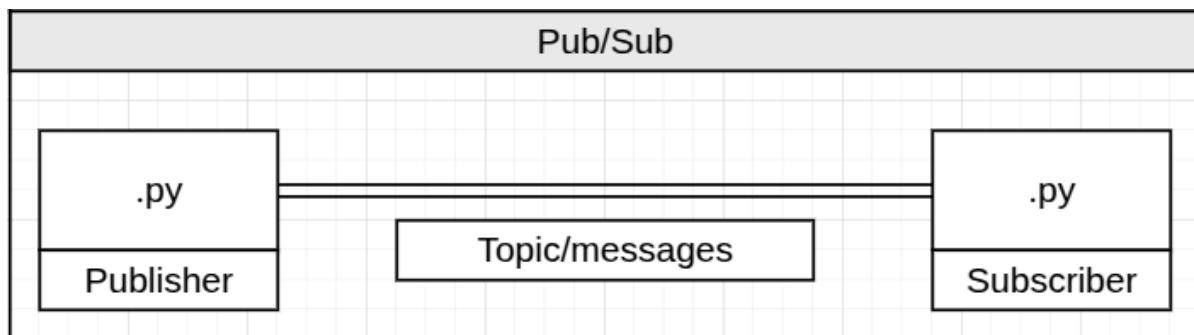
## 8.1. SSH

Der blev etableret SSH-adgang til Pi5. En statisk IP-adresse blev konfigureret, og udviklingsarbejdet blev udført via terminalen samt Visual Studio Code i Remote SSH-tilstand. Se bilag 13.4 for beskrivelse af den statiske IP-konfiguration og dokumentation af Remote SSH.

## 8.2 Robot Controller - Ros2 Jazzy

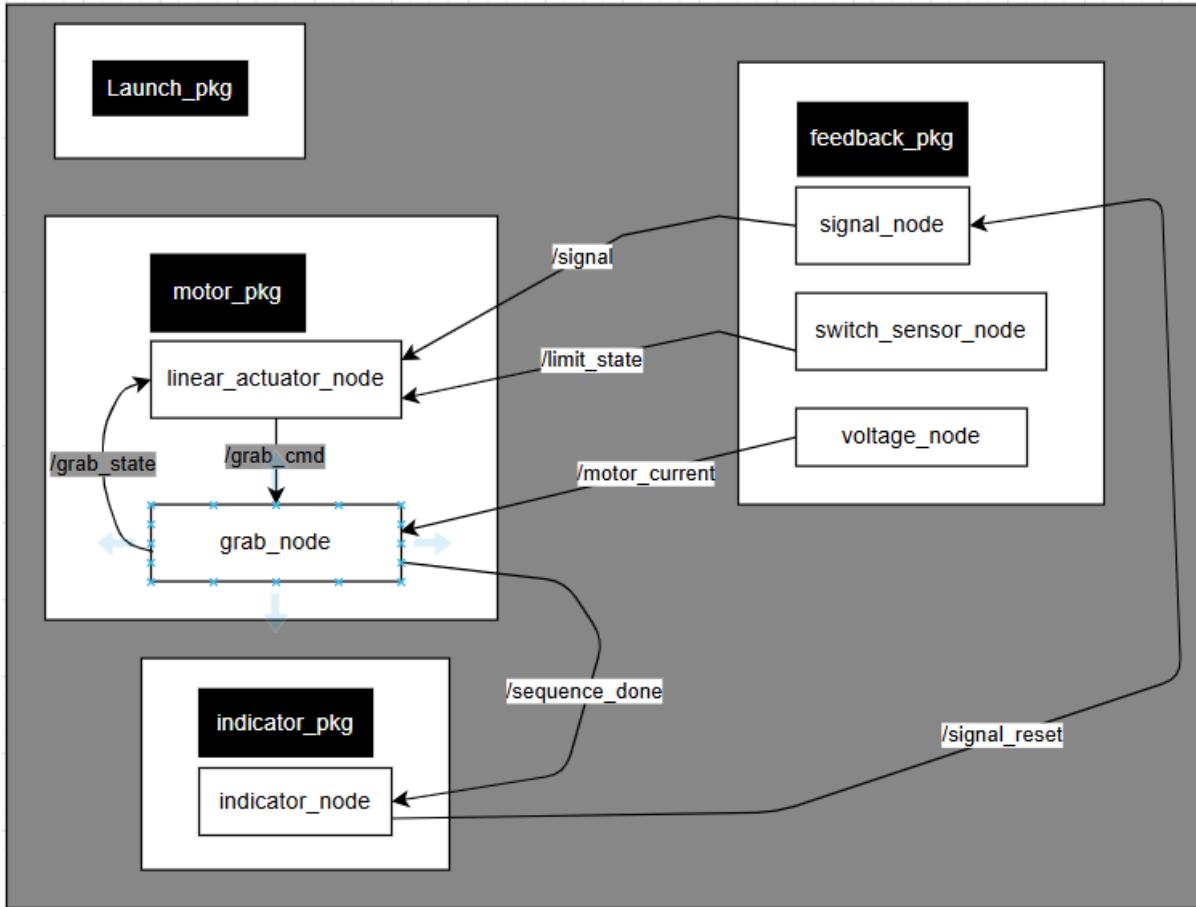
Valget af "Robot Controller" faldt på Ros2 Jazzy Jalisco, som overordnet set er middleware i laget mellem styresystemet og applikationslaget - altså en samling af værktøjer, biblioteker og koncepter, som ligger mellem Ubuntu og Python koden (applikationen). Årsagen til dette valg er, at der senere kan arbejdes simuleret med dette på platformen Gazebo (se afsnit 9) og det var ligeledes et personligt ønske at udvikle kompetencer indenfor dette felt, der afgjorde endeligt valg.

Ros2 kører med DDS som kommunikationsform, hvor der er tre metoder, koden kan kommunikere, men der blev valgt at holde projektet til pub/sub pipelinen, som indgangsvinkel til at arbejde med dette, da viden og erfaring er relativt beskeden på nuværende tidspunkt. De to andre pipelines hedder service og actions.



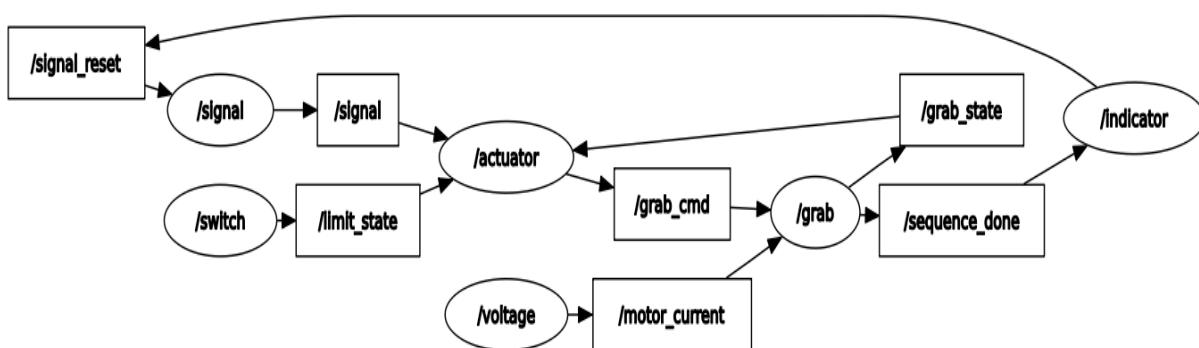
Figur 35: Den valgte kommunikationsmetode - .py kaldes noder, kommunikation mellem dem pipeline

For at danne sig et overblik over kommunikationen mellem de forskellige noder, er det vigtig at danne sig et overblik over de forskellige pipelines inden de udvikles og derpå kan man teste dem med graph, som følger med Ros2:



Figur 36: Overblik inden software var udviklet.

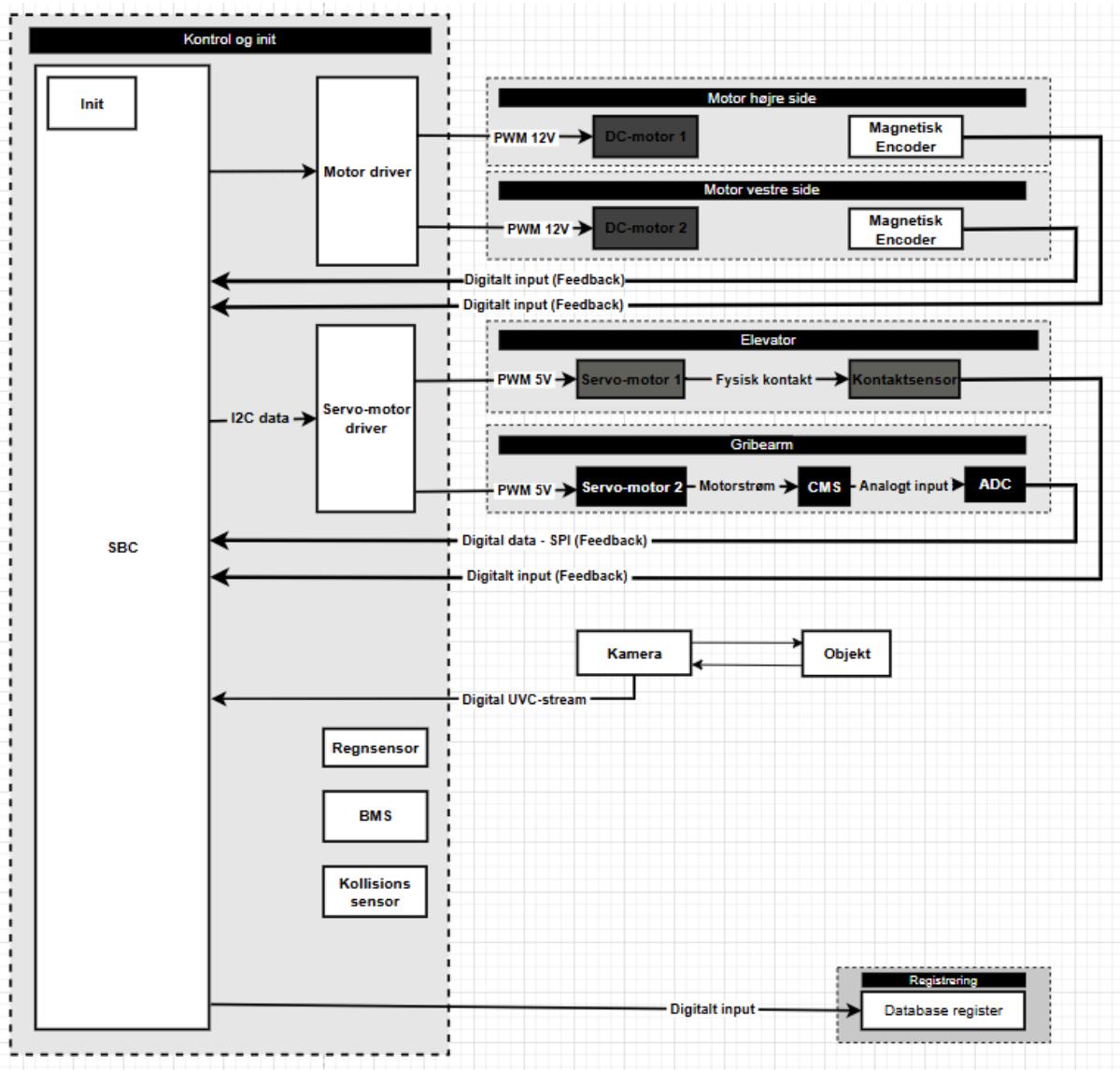
Efter implementering anvendes rqt\_graph for at sikre sig at noderne snakker sammen som det ønskes. install til jazzy base, se bilag 13.6.



Figur 37: Med rqt\_graph fås det endelige overblik over node kommunikationen

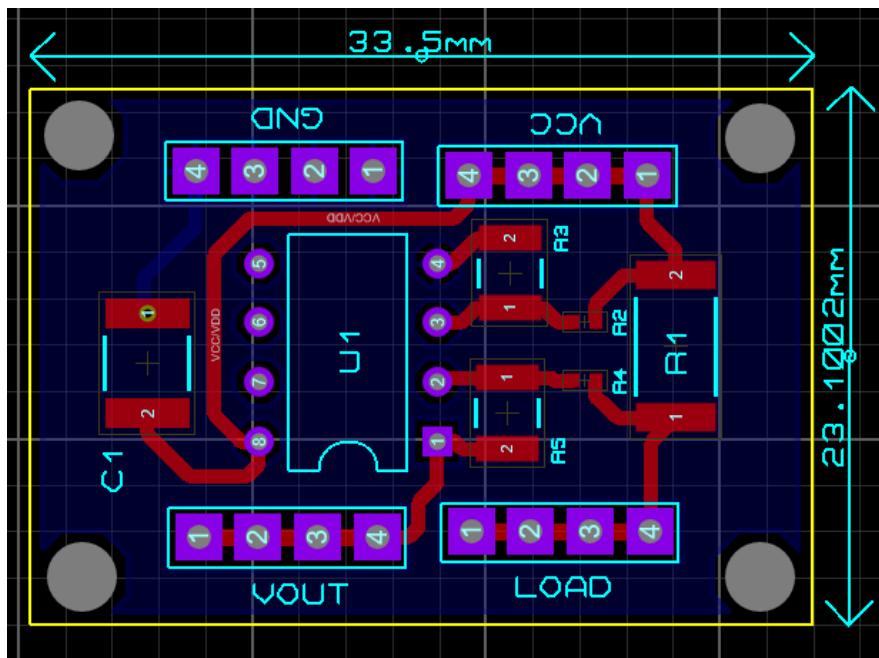
## 9. Fremtidige Udvidelser

Under arbejdet med dette system, blev der endvidere reflekteret over fremtidige udvidelser og der blev i processen lavet et midlertidigt blokdiagram med DC motor, Kamera, regnsensor, BMS, kollisionssensor og database register.



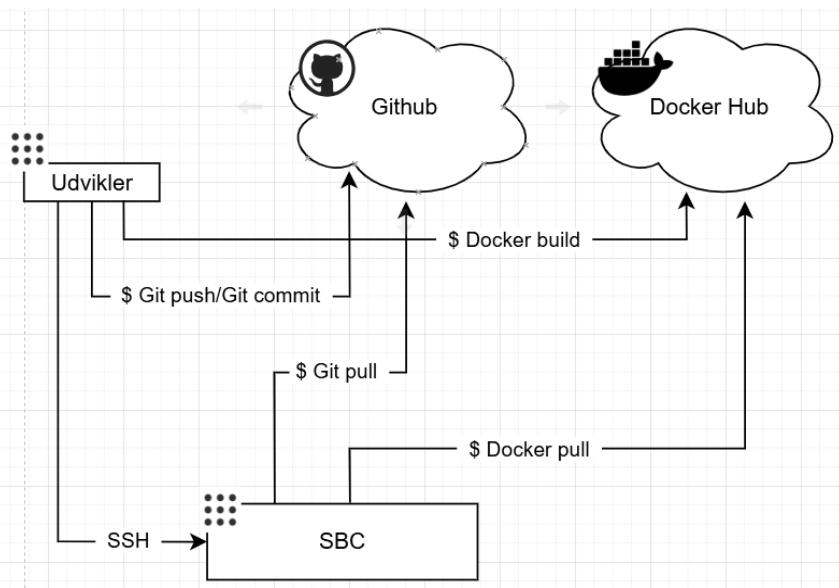
Figur 38: Overvejelser i forhold til at skabe et mere komplet og mobilt system - relativt ukomplet

Hvis en robot med denne funktionalitet skal produceres og sælges som et stand-alone produkt, er det vigtigt at gøre den så lille og kompakt som mulig, derfor er det essentielt at lave al elektronikken ved at udvikle PCB komponenter.



Figur 39: Design af fremtidigt PCB til CMS

For at undgå versions-konflikter og få et optimalt workflow kan platforme som Docker Hub og Github, med fordel, implementeres. Docker Hub og containerization logikken ville håndtere dependencies og OS mismatch og endvidere ville Github sikre et fælles workflow i tilfælde af at der er flere udviklere på opgaven. Se bilag 13.7. og 13.8.



Figur 40: Overvejelser af et potentieligt workflow med Github og Docker Hub implementering

C++ implementering i forhold til realtime komponenter som motorer, sensorer og kameraer vil være en fordel, da C++ er hurtigere og giver mere mening at anvende i en færdig implementering, da noderne i ros2 kan kommunikere i begge syntax.

I det fremtidige arbejde med Ros2 vil det være en stor fordel at mestre anvendelse af Gazebo, som er en 3D simulering af robotter, man kan udvikle egne robotter med .xacro filer fra udvirkede 3D model.



*Figur 41: Gazebo, som er kompatibel med Ros2 og som anvender Ros2 kode til simulering*

Live målinger med “eProximas” Fast DDS, skal med i senere iterationer, da det giver et godt livefeed over kommunikation mellem de forskellige noder i de forskellige pakker. Fast DDS blev installeret og testet se bilag 13.8 og 13.9.



*Figur 42: eProxima, som er kompatibel med Ros2 og som anvender Ros2 live kommunikation*

---

## 10. Konklusion

Projektet har vist, hvordan udvikling af en ROS2-baseret robot til bekæmpelse af dræbersnegle kræver en systematisk tilgang til både hardware og software. Arbejdet med integration af digitalt og analogt feedback, aktuatorstyring og closed-loop kontrol har tydeliggjort vigtigheden af modulær opbygning, hvor hver komponent og node spiller en klart defineret rolle.

Undervejs blev det tydeligt, at robusthed og driftssikkerhed afhænger af en velgennemtænkt forsyningsstrategi samt nøje afstemning mellem komponentvalg og systemkrav.

Implementeringen af en high-side current measuring sensor med analogt input via MCP3008 ADC har givet værdifuld erfaring med signalbehandling og grænseflader mellem forskellige hardwaretyper.

Projektet har samtidig synliggjort, at arbejdet med moderne open source-værktøjer som ROS2, Docker og GitHub ikke blot fremmer samarbejde og dokumentation, men også letter test, fejlfinding og videreudvikling. I praksis har der opstået uforudsete udfordringer, særligt omkring strømforsyning, som har krævet justeringer og genovervejelser af designvalg.

Samtidig har arbejdet med pub/sub-arkitekturen i ROS2 understreget betydningen af et fleksibelt og skalerbart softwaredesign, hvor fremtidige udvidelser – eksempelvis integration af nye sensor typer eller mobilitet – kan implementeres uden at kompromittere systemets stabilitet.

Samlet set peger projektet på, at en velintegreret, modulær og dokumenteret tilgang danner et solidt fundament for videreudvikling af autonome robotsystemer – både teknisk og metodisk. Bevis vises til eksamen, midlertidig drifts video:

<https://youtube.com/shorts/rz5jnoie0Ug>

## 11. Materialeliste

Antal	Komponent	Reference	Værdi
1	Raspberry Pi 5	SBC	
1	PCA9685	Servo-motor driver	
2	MG995	Servo-motor	
2	LM2596	DC-DC converter	Stepdown 5V
1	Lipo	Forsyning	7,4V
2	Strømfordeler	Forsyning	6 ud + indgange
4	Akryl plast plade	Chassis	20x20 cm
7	Kuglelejer	Chassis	22 Ø
1	Usb-c kabel	Forsyning	
1	MCP3008	ADC	8 kanaler
2	Afkoblingskondensatorer	Forsyning	330 uf
1	LED	Indikator	Rød
1	pushbutton	Signal	
1	MCP6002	CMS	
2	Modstand	CMS	270 ohm
2	Modstand	CMS	4700 ohm
1	Shuntnodstand	CMS	0,22 ohm
55	Headers	Fuldt system	
30	Ledning	Forsyning	
½ rulle	Filament	Chassis	

## 12. Referencer

High side current measuring design:

<https://www.ti.com/lit/an/sboa310b/sboa310b.pdf?ts=1753097926881>

MG995 datablad:

<https://www.alldatasheet.com/datasheet-pdf/view/1132435/ETC2/MG995.html>

PCA9685 datablad:

<https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf>

MCP3008 datablad:

<https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf>

MCP6002 datablad:

<https://ww1.microchip.com/downloads/en/DeviceDoc/MCP6001-1R-1U-2-4-1-MHz-Low-Power-Op-Amp-DS20001733L.pdf>

ROS2 dokumentation:

<https://docs.ros.org/en/jazzy/index.html>

Ubuntu 24.04.2 dokumentation:

[https://documentation.ubuntu.com/\\_/downloads/server/en/latest/pdf/](https://documentation.ubuntu.com/_/downloads/server/en/latest/pdf/)

SpiDev documentation:

[https://sigmdel.ca/michel/ha/rpi/dnld/draft\\_spidev\\_doc.pdf](https://sigmdel.ca/michel/ha/rpi/dnld/draft_spidev_doc.pdf)

Statisk IP addresse:

<https://linuxconfig.org/setting-a-static-ip-address-in-ubuntu-24-04-via-the-command-line>

gpiozero dokumentation:

<https://gpiozero.readthedocs.io/en/stable/>

LM2596 datablad:

<https://www.ti.com/lit/ds/symlink/lm2596.pdf>

Circuit python dokumentation

<https://docs.circuitpython.org/projects/bundle/en/latest/drivers.html>

Raspberry pi dokumentation:

<https://www.raspberrypi.com/documentation/>

Terminator dokumentation:

<https://gnome-terminator.org/>

## 13. Bilag

*“...you could just see where these technology trends were going and between injection molded plastic and a camera with a computer capable of running machine learning and visual object recognition I could build an incredibly affordable, incredibly capable robot and that's going to be the future.” - Colin Angle (2020) “How to build a successful robotics company” - <https://www.youtube.com/watch?v=j7gM19hphRk>*<sup>3</sup>

### 13.1. Prerequisites

#### Design og implementering:

Proteus 9

Fusion360

Creality studio

#### Software udvikling:

Ubuntu 24.04.2 Server

Ros2 - Jazzy Jalisco base: <https://docs.ros.org/en/jazzy/Installation/Ubuntu-Install-Debs.html>

Terminator 2.1 : <https://gnome-terminator.readthedocs.io/en/latest/index.html>

VScode 1.99.3: <https://code.visualstudio.com/>

- Plugins: ROS, CMake, C/C++, C/C++ extension pack, Python, Remote-SSH

Gedit <https://gedit-text-editor.org/>

gpiozero - \$ sudo apt install python3-gpiozero

spidev

adafruit\_pca9685

#### Fremtidige overvejelser af prerequisites:

Github

Docker 28.1.1

VScode

- Plugins: CMake, C/C++, C/C++ extension pack

---

<sup>3</sup> Colin Angle (2020) “How to build a successful robotics company” - <https://www.youtube.com/watch?v=j7gM19hphRk>

## 13.2. Wiring RPI

RPI	
3.3v x 2	MCP3008
GPIO17	LED
GPIO10	DIN MCP3008
GPIO9	DOUT - MCP3008
GPIO11	CLK - MCP3008
GPIO18	GRAB SERVO
3.3v	PCA9685
GPIO8	MCP3008 CS/SHDN
GPIO16	KNAP
GPIO13	ELEVATOR SERVO
GPIO27	POSITIONS SENSOR
GPIO2	SDA PCA9685
GPIO3	SCL PCA9685

### 13.3. Testkode

#### RPI5 pins og problemer:

For at undgå proces drab efter program kørsel blev der arbejdet med context managers og exception handling. Hvis der kører kode uden skal man dræbe processerne med:

#### Proces drab:

Find proces:

```
$ ps aux | grep 'servo_test.py'
```

Dræb proces:

```
$ sudo kill -9 <PID>
```

#### 13.3.1. Testkode - Digital Input og Digital Output

```
### lille test, der demonstrerer digitalt input (knap) og digitalt output (LED) med
### gpiozero. Jeg har delt den op i logiske blokke, så du nemt kan se, hvad der sker

from gpiozero import Button, LED
import time

# _____
# Pin definitioner
#
led_pin = 17
button_pin = 16

# _____
# Context-manager og exception handling for frigørelse af pin
#
with LED(led_pin) as led, Button(button_pin) as button:
    led.off() # LED sikres slukket fra start
    print("System klar til at tage imod input. LED er slukket.")

try:
    while True:
        button.wait_for_press() # Programmet stopper til knappen trykkes
        led.on() # LED tændes for at indikere en handling
        print("Knappen blev trykket! LED er nu tændt i 5 sekunder.")
        time.sleep(5) # Vent i 5 sekunder mens LED er tændt
        led.off() # LED slukkes igen
        print("LED er slukket igen. Tryk på knappen for at tænde den igen.")

# _____
# Programmet afsluttes med ctrl+C
#
except KeyboardInterrupt:
    print("\nAfslutter program. LED slukkes og ressourcer frigives.")
```

### 13.3.2. Testkode - I<sup>2</sup>C Data og PWM

```
#!/usr/bin/env python3                                # eksekverbar direkte fra terminalen

import time
import board
import busio
from adafruit_pca9685 import PCA9685
from adafruit_motor.servo import Servo

# -----
# Hardware opsætning
#
i2c = busio.I2C(board.SCL, board.SDA) # Opret I2C-bus
pca = PCA9685(i2c) # PWM-driver på default-adresse 0x40
pca.frequency = 50 # 50 Hz er standard for MG995
servo = Servo(pca.channels[0]) # Kanal 0 → ét servo-objekt

# -----
# Vinkel indtastning
#
def ask_angle() -> float | None:
    try:
        raw = input("Vinkel (0-180, q=quit): ").strip()
        if raw.lower() in {"q", ""}:
            return None
        angle = float(raw)
        if not 0 <= angle <= 180:
            print("Ugyldig vinkel - skriv mellem 0 og 180.")
            return ask_angle()
        return angle
    except ValueError:
        print("Kun tal - prøv igen.")
        return ask_angle()

# -----
# Hovedløkken
#
def main():
    print("PCA9685 servo-test klar.")
    try:
        while True:
            angle = ask_angle()
            if angle is None:
                break
            print(f"→ Sætter servo til {angle}°")
            servo.angle = angle
            time.sleep(0.5)
    finally:
        servo.angle = None
        pca.deinit()
        print("Lukker ned - farvel!")

if __name__ == "__main__":
    main()
```

### 13.3.3. Testkode - Digitalt Input (Feedback)

```
from gpiozero import AngularServo, Button
from time import sleep

# _____
# Hardware opsætning
# _____
servo = AngularServo(18, min_angle=0, max_angle=90)
home_button = Button(27, pull_up=True)

# _____
# Servo kørsel
# _____
try:
    print("Starter homing - søger hjem (knap på GPIO 27)")
    servo.angle = 0
    sleep(0.5)

    found_home = False
    current_angle = 0
    step = 1 # Grad pr. step

    # _____
    # Hovedløkken
    # _____
    while not found_home and current_angle <= 90:
        servo.angle = current_angle
        sleep(0.05)
        if home_button.is_pressed:
            found_home = True
            print(f"Hjemmeposition fundet ved vinkel {current_angle} grader!")
        else:
            current_angle += step

    if not found_home:
        print("Knap blev ikke trykket - servo nåede max vinkel!")

    # _____
    # Exception handling for at undgå at pin sætter sig fast
    # _____
except KeyboardInterrupt:
    print("\nAfslutter sikkert (Ctrl+C)...")


# _____
# Afkobling af servo
# _____
finally:
    print("Frigør servo og GPIO pins.")
    servo.detach()
```

### 13.3.4. Testkode - Motorstrøm og Analogt Input og Digitalt Data - SPI (Feedback)

```
### Simpel SPI test med en MCP3008 til Raspberry Pi 5
### Læser spændingen på channel 0
### Printer værdien hvert 100 ms.

import spidev # Officiel SPI library til Pi
import time    # Til forsinkel mellem værdi output

# -----
# SPI INITIALISERING
# -----
spi = spidev.SpiDev()          # Opret et SPI-objekt
spi.open(0, 0)                 # Bus 0, Device 0 (CE0) - se wiring-diagram
spi.max_speed_hz = 1_000_000 # 1 MHz ☺ stabil hastighed for MCP3008
def read_voltage(channel, vref=3.3):

    # SPI-kommando: 1 = start-bit, (8+channel)<<4 = single-ended + kanal, 0 = dummy-byte
    # xfer2 sender hele listen ud i én transaktion og modtager en liste med samme længde,
    # hvor hver byte er, hvad ADC'en sendte tilbage samtidig (full-duplex)
    adc = spi.xfer2([1, (8+channel)<<4, 0])

    # Kombiner de 10 bit fra de to sidste bytes til én værdi (0-1023)
    data = ((adc[1] & 3) << 8) + adc[2]

    # Omregn rådata til volt
    volt = (data / 1023.0) * vref
    return volt

# -----
# Hovedløkke
# Løkken kører uendeligt, indtil der trykkes Ctrl + C
# -----
while True:
    volt = read_voltage(0)           # Læs kanal 0
    print(f"Spænding: {volt:.3f} V") # Udskriv med 3 decimaler
    time.sleep(0.1)                 # Vent 100 ms før næste mæling
```

## 13.4. SSH

Den letteste måde at arbejde med en singleboard computer som er sat op headless, er at anvende SSH protokollen som fjernstyrings metode. Først skal man finde IP addressen enten med AngryIPScanner eller Nmap, når der er adgang til SBC, så er det bedst at lave en statisk IP adresse, så man ikke skal lede efter adressen hvergang DHCP sætter en ny adresse. Det gøres ved at:

Konfiguration af netadgang filen:

```
$ sudo nano /etc/netplan/50-cloud-init.yaml
```

Opsætning af sikkerhed:

```
$ sudo chmod 600 /etc/netplan/01-netcfg.yaml
```

Opsætning af statisk IP:

```
$ sudo netplan apply
```

Koden til opsætning:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: true
      optional: true
  wifis:
    wlan0:
      dhcp4: false
      addresses:
        - 192.168.1.XX/24
      routes:
        - to: default
          via: 192.168.1.1
  nameservers:
    addresses: [1.1.1.1, 8.8.8.8]
  access-points:
    "SSID":
      password: "PASSWORD"
```

## 13.5. Ros2 Jazzy Jalisco

Install af Ros2 til ubuntu 24.04.2:

<https://docs.ros.org/en/jazzy/Installation/Ubuntu-Install-Debs.html>

Lav work space:

```
$ mkdir ros2_ws
$ ~/ros2_ws $ mkdir src
```

```
$ ~/ros2_ws $ colcon build
```

For at få build, install og log:

```
build install log src  
grabbot@grabbot:~/ros2_ws$
```

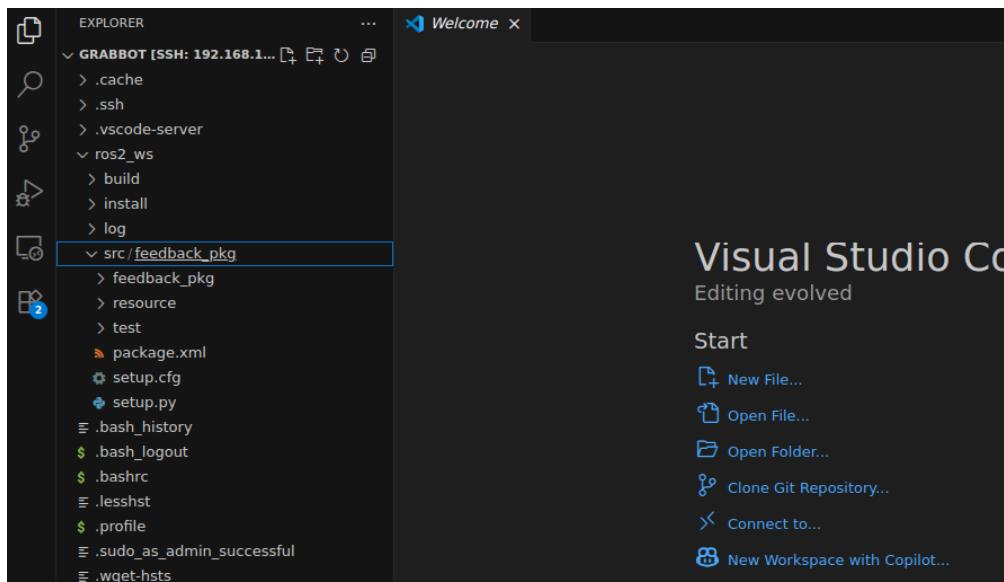
Tilføj source kommandoer til .bashrc

```
$ sudo nano .bashrc source /opt/ros/jazzy/setup.bash  
source ~/ros2_ws/install/setup.bash  
source /opt/ros/jazzy/setup.bash  
source ~/ros2_ws/install/setup.bash
```

Create package:

```
$ ~/ros2_ws/src $ ros2 pkg create feedback_pkg --build-type ament_python --dependencies  
rcpy
```

Remote ssh i VS-code:



Create node:

```
~/ros2_ws/src/feedback_pkg/feedback_pkg$ touch signal_node.py  
~/ros2_ws/src/feedback_pkg/feedback_pkg$ chmod +x signal_node.py
```

Opsætning af Igpio:

```
$ sudo apt update  
$ sudo apt install -y python3-lgpio python3-pip python3-libgpiod python3-rpi.gpio  
$ sudo apt install -y python3-gpiozero
```

Giv sudo rettigheder til user, adgang til dialout og i2c og tilføj common rules:

```
$ sudo chown -R $USER:$USER ~/ros2_ws  
$ sudo usermod -aG dialout,i2c $USER  
$ sudo apt install -y rpi.gpio-common
```

**Installer circuitpython:**

```
$ sudo pip3 install --break-system-packages adafruit-blinka
```

**Lytte med på de forskellige topics:**

```
$ ros2 topic echo /limit_state
```

## 13.5. Implementeret Ros2 kode

Opbygning af denne kode gennemgang består af dannelse af pakker med linux kommandoerne efterfulgt af package.xml og setup.py, hvor man kan se dependencies og Entry points som er den brugerstyrede del af disse standardkoder. Efterfølgende kan koden til de noder pakken indeholder, læses.

### 13.5.1. Feedback\_pkg

~ros2\_ws/src

```
$ ros2 pkg create feedback_pkg --build-type ament_python --dependencies rclpy gpiozero  
spidev
```

**package.xml:**

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>feedback_pkg</name>
  <version>1.0.0</version>
  <description>Denne pakke håndterer al feedback og sensorisk input i robotten.  

    Den indeholder noder til digital input fra tryknap, kontaktfeedback fra  

    lineær aktuatorens endeposition og spændingsmåling fra current measuring  

    sensor via MCP3008 ADC.</description>
  <maintainer email="grabbottodo.todo">grabbottodo</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>
  <depend>gpiozero</depend>
  <depend>spidev</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

**setup.py:**

```
from setuptools import find_packages, setup

package_name = 'feedback_pkg'

setup(
    name=package_name,
    version='1.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
         ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=[
        'setuptools',
        'rclpy',
        'gpiod',
        'spidev',
    ],
    zip_safe=True,
    maintainer='grabbott',
    maintainer_email='grabbott@todo.todo',
    description='Denne pakke håndterer al feedback og sensorisk input i robotten. Den indeholder noder til digital input fra trykknap, kontaktfeedback fra lineær aktuatorens endeposition og spændingsmåling fra current measuring sensor via MCP3008 ADC.',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'signal_node = feedback_pkg.signal_node:main',
            'switch_sensor_node = feedback_pkg.switch_sensor_node:main',
            'voltage_node = feedback_pkg.voltage_node:main',
        ],
    },
)
```

### 13.5.1.1. Signal\_node

```
#!/usr/bin/env python3

# signal_node.py
```

```
#  
_____  
# - Venter på trykknop (GPIO16)  
# - publicerer std_msgs/Empty på topic '/signal', når knappen trykkes ind  
#  
_____  
  
# —— Imports  
  
import rclpy  
from rclpy.node import Node  
from std_msgs.msg import Empty  
from gpiozero import Button  
  
#  
_____  
  
class SignalNode(Node):  
    def __init__(self):  
        super().__init__("signal_node")  
  
        # —— hardwareopsætning  
  
        self.button = Button(GPIO=16,  
                            pull_up=True,  
                            debounce_time=0.05)  
  
        # —— publisher  
  
        self.pub = self.create_publisher(msg=Empty, topic="/signal",  
                                         qos_profile=10)  
  
        # —— subscriber til reset  
  
        self.create_subscription(msg=Empty, topic="/signal_reset",  
                               callback=self.cb_reset,  
                               qos_profile=10)  
  
        # —— registrér callback når knappen trykkes ——————  
        self.button.when_pressed = self.handle_press  
        self.get_logger().info(" signal_node klar, tryk på knappen!")  
  
        # —— Udsend ét Empty-msg ved knaptryk  
  
    def handle_press(self) -> None:  
        self.pub.publish(Empty())  
        self.get_logger().info("Signal sendt.")  
  
    def cb_reset(self, msg=Empty) -> None:  
        self.get_logger().info("Reset modtaget - klar til nyt knaptryk.")  
  
# —— Main-indgang  
  
def main():
```

```

rclpy.init()
node = SignalNode()
try:
    rclpy.spin(node) # kør, indtil Ctrl-C
finally:
    node.destroy_node()
    rclpy.shutdown()

if __name__ == "__main__":
    main()

```

### 13.5.1.2. Switch\_sensor\_node

```

#!/usr/bin/env python3

# switch_sensor_node.py
#
_____
# - Lytter på en end-switch (GPIO27)
# - publicerer "up" eller "released" på /limit_state.
#
_____
# — Imports
_____
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
from gpiozero import Button

#
_____
class SwitchSensorNode(Node):
    def __init__(self):
        super().__init__("switch_sensor_node")
        self.pub = self.create_publisher(String, topic="/limit_state", qos=10)

        # — end-switch: aktiv (pressed) =
"up"_____
        self.switch = Button(27, pull_up=True, bounce_time=0.02)
        self.switch.when_pressed = lambda: self.publish("up")
        self.switch.when_released = lambda: self.publish("released")
        self.get_logger().info("Switch-sensor klar på GPIO 27.")

        # ——publish
state_____
    def publish(self, state=str):
        self.pub.publish(String(data=state))
        self.get_logger().info(f"/limit_state: {state}")

```

```
# ----- Main-indgang

def main():
    rclpy.init()
    node = SwitchSensorNode()
    try:
        rclpy.spin(node)
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == "__main__":
    main()
```

#### 13.5.1.3. Voltage\_node

```

# —— Timer kalder self.measure()
self.create_timer(1.0 / SAMPLE_RATE, self.measure)
self.get_logger().info(f"VoltageNode kører {SAMPLE_RATE} Hz")

# ——————
# 1. Læs rå-ADC-tal (0-1023) fra MCP3008
# ——————
def _read_adc_raw(self) -> int:
    # SPI-ramme: [start-bit, kanal-vælg, dummy]
    frame = [1, (8 + ADC_CHANNEL) << 4, 0]
    resp = self.spi.xfer2(frame)
    raw = ((resp[1] & 0x03) << 8) | resp[2] # 10-bit tal
    return raw

# ——————
# 2. Mål → konverter → publicer
# ——————
def measure(self):
    raw = self._read_adc_raw() # 0-1023
    volts = raw * V_REF / 1023.0 # ADC → Volt
    amps = volts / GAIN_V_PER_A # Volt → Ampere
    self.publisher.publish(Float32(data=amps))

# —— Main-indgang ——
def main():
    rclpy.init()
    node = VoltageNode()
    try:
        rclpy.spin(node) # kør, indtil Ctrl-C
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == "__main__":
    main()

```

### 13.5.2. Motor\_pkg

~ros2\_ws/src

```
$ ros2 pkg create motor_pkg --build-type ament_python --dependencies rclpy gpiozero
spidev
```

#### package.xml:

```

<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
    <name>motor_pkg</name>
    <version>0.0.0</version>
    <description>Denne pakke håndterer aktuering og kontrol af de to
        servo-motorer via en PCA9685 PWM-driver. Den inkluderer en
        sekventiel styret lineær aktuator-node og en intelligent

```

```

        gribearm-node med overstrømsbeskyttelse via analog
    feedback.</description>
<maintainer email="grabbottodo.todo">grabbottodo</maintainer>
<license>TODO: License declaration</license>

<depend>rclpy</depend>
<depend>adafruit-circuitpython-pca9685</depend>
<depend>adafruit-circuitpython-servo</depend>

<test_depend>ament_copyright</test_depend>
<test_depend>ament_flake8</test_depend>
<test_depend>ament_pep257</test_depend>
<test_depend>python3-pytest</test_depend>

<export>
    <build_type>ament_python</build_type>
</export>
</package>

```

### **setup.py:**

```

from setuptools import find_packages, setup

package_name = 'motor_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=[
        'setuptools',
        'rclpy',
        'adafruit-blinka',
        'adafruit-circuitpython-pca9685',
        'adafruit-circuitpython-motor',
    ],
    zip_safe=True,
    maintainer='grabbottodo',
    maintainer_email='grabbottodo.todo',
    description='Denne pakke håndterer aktuering og kontrol af de to servo-motorer
via en PCA9685 PWM-driver. Den inkluderer en sekventiel styret lineær
aktuator-node og en intelligent gribearm-node med overstrømsbeskyttelse via
analog feedback.',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={

}

```

```

        'console_scripts': [
            'linear_actuator_node = motor_pkg.linear_actuator_node:main',
            'grab_node = motor_pkg.grab_node:main',
        ],
    },
)

```

### 13.5.1.2. linear\_actuator\_node

```

#!/usr/bin/env python3
# grab_node.py
#
# -----
# - Subscriber: /grab_cmd      ("close" | "open")
# - Publisher : /grab_state    ("closed" | "opened" | "fault")
# - Publisher : /sequence_done (Empty) - sendes når griberen er åbnet igen
# - Overstrøms-watchdog på /motor_current ( **Volt** fra CMS )
# ----

# —— Imports

import time, rclpy, board, busio
from rclpy.node import Node
from std_msgs.msg import String, Float32, Empty
from adafruit_pca9685 import PCA9685
from adafruit_motor import servo

# —— Konstanter

PCA_CHANNEL    = 1
SERVO_FREQ     = 50
ANGLE_OPEN     = 20
ANGLE_CLOSE    = 100
MOVE_TIME_S    = 1.0           # sek. før status-publish

#   CMS konverterer 0,22 Ω-shunt × gain 17,4 → ≈ 3,83 V/A.
#   Stall (≈ 0,7A) måler ca. 2,7V. Vi tripper lidt over - 3,0V.
MAX_VOLT       = 3.0          # V - watchdog-grænse
TRIP_SAMPLES   = 8            # antal målinger over grænsen før stop
CHECK_PERIOD   = 0.05         # hovedløkke-periode (sek.)

STATE_IDLE     = "idle"
STATE_CLOSING  = "closing"
STATE_OPENING   = "opening"
STATE_FAULT    = "fault"

#
# ——
class GrabNode(Node):
    def __init__(self):
        super().__init__("grab_node")

```

```

# hardware-setup
i2c = busio.I2C(board.SCL, board.SDA)
self.pca = PCA9685(i2c); self.pca.frequency = SERVO_FREQ
self.servo = servo.Servo(self.pca.channels[PCA_CHANNEL],
                         min_pulse=500, max_pulse=2700)

# pub/sub
self.state_pub = self.create_publisher(String, "/grab_state", 10)
self.done_pub = self.create_publisher(Empty, "/sequence_done", 10)

self.create_subscription(String, "/grab_cmd", self.cb_cmd, 10)
self.create_subscription(Float32, "/motor_current", self.cb_cur, 10)

# interne variabler
self.state = STATE_IDLE
self.over_cnt = 0
self.move_deadline = 0.0
self.final_state = None # "opened"/"closed"

# hovedløkke
self.create_timer(CHECK_PERIOD, self.loop)

self.get_logger().info("Grab-node klar - venter på /grab_cmd.")

# —— Kommando-callback ——
def cb_cmd(self, msg: String):
    cmd = msg.data.lower().strip()
    if cmd == "close" and self.state == STATE_IDLE:
        self._start_move(ANGLE_CLOSE, "closed", STATE_CLOSING)
    elif cmd == "open" and self.state == STATE_IDLE:
        self._start_move(ANGLE_OPEN, "opened", STATE_OPENING)
    else:
        self.get_logger().warn(f"Ignorerer /grab_cmd: {cmd} (state={self.state})")

def _start_move(self, angle: float, final_state: str, next_state: str):
    self.get_logger().info(f"Servo → {angle}° ({final_state.upper()})")
    self.servo.angle = angle
    self.state = next_state
    self.over_cnt = 0
    self.final_state = final_state
    self.move_deadline = time.time() + MOVE_TIME_S

# —— Strøm-watchdog (volt) ——
def cb_cur(self, msg: Float32):
    if self.state != STATE_CLOSING:
        return
    volts = msg.data # CMS sender VOLT
    if volts > MAX_VOLT:
        self.over_cnt += 1
        if self.over_cnt >= TRIP_SAMPLES:
            self.get_logger().error(f"!! Overstrøm {volts:.2f} V - stopper!")
            self.servo.angle = None
            self._publish_state(STATE_FAULT)
    else:
        self.over_cnt = 0

```

```

# —— Hovedløkke
_____
def loop(self):
    if self.move_deadline and time.time() >= self.move_deadline:
        self.move_deadline = 0.0
    if self.final_state:
        self._publish_state(self.final_state)
        self.final_state = None

# —— Status-publish helper ——
def _publish_state(self, word: str):
    self.state_pub.publish(String(data=word))
    if word in ("opened", "closed"):
        if word == "opened":
            self.done_pub.publish(Empty())
        self.state = STATE_IDLE
    elif word == STATE_FAULT:
        self.state = STATE_FAULT

# —— Oprydning
_____
def destroy_node(self):
    self.servo.angle = None
    self.pca.deinit()
    super().destroy_node()

# —— Main-indgang
_____
def main():
    rclpy.init()
    node = GrabNode()
    try:
        rclpy.spin(node)
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == "__main__":
    main()

```

### 13.5.1.2. grab\_node

```

#!/usr/bin/env python3
# grab_node.py
#
# -----
# - Subscriber: /grab_cmd      ("close" | "open")
# - Publisher : /grab_state    ("closed" | "opened" | "fault")
# - Publisher : /sequence_done (Empty) - sendes når griberen er åbnet igen
# - Overstrøms-watchdog på /motor_current (volt fra CMS)
# -----
#
# —— Imports

```

---

```

import time, rclpy, board, busio
from rclpy.node import Node
from std_msgs.msg import String, Float32, Empty
from adafruit_pca9685 import PCA9685
from adafruit_motor import servo

# — Konstanter

PCA_CHANNEL = 1
SERVO_FREQ = 50
ANGLE_OPEN = 20
ANGLE_CLOSE = 158
MOVE_TIME_S = 1.0           # tid før status-publish

VOLT_PER_AMP = 3.83          # 0,22 Ω × 17,41
MAX_CURRENT = 0.80           # A
MAX_VOLT = MAX_CURRENT * VOLT_PER_AMP   # ≈ 3,06 V
TRIP_SAMPLES = 5
CHECK_PERIOD = 0.05          # hovedløkkens periode (sek.)

STATE_IDLE = "idle"
STATE_CLOSING = "closing"
STATE_OPENING = "opening"
STATE_FAULT = "fault"

#



class GrabNode(Node):
    def __init__(self):
        super().__init__("grab_node")

        # hardware-setup
        i2c = busio.I2C(board.SCL, board.SDA)
        self.pca = PCA9685(i2c)
        self.pca.frequency = SERVO_FREQ
        self.servo = servo.Servo(self.pca.channels[PCA_CHANNEL],
                               min_pulse=500, max_pulse=2700)

        # pub/sub
        self.state_pub = self.create_publisher(String, "/grab_state", 10)
        self.done_pub = self.create_publisher(Empty, "/sequence_done", 10)

        self.create_subscription(String, "/grab_cmd", self.cb_cmd, 10)
        self.create_subscription(Float32, "/motor_current", self.cb_cur, 10)

        # interne variabler
        self.state = STATE_IDLE
        self.over_cnt = 0
        self.move_deadline = 0.0
        self.final_state = None   # "opened"/"closed"

        # hovedløkke
        self.create_timer(CHECK_PERIOD, self.loop)

```

---

```

    self.get_logger().info("Grab-node klar - venter på /grab_cmd.")



---


# —— Kommando-callback ——
def cb_cmd(self, msg: String):
    cmd = msg.data.lower().strip()
    if cmd == "close" and self.state == STATE_IDLE:
        self._start_move(ANGLE_CLOSE, "closed", STATE_CLOSING)
    elif cmd == "open" and self.state == STATE_IDLE:
        self._start_move(ANGLE_OPEN, "opened", STATE_OPENING)
    else:
        self.get_logger().warn(f"Ignorerer /grab_cmd: {cmd} (state={self.state})")

def _start_move(self, angle: float, final_state: str, next_state: str):
    self.get_logger().info(f"Servo → {angle}° ({final_state.upper()})")
    self.servo.angle = angle
    self.state = next_state
    self.over_cnt = 0
    self.final_state = final_state
    self.move_deadline = time.time() + MOVE_TIME_S



---


# —— Strøm-watchdog ——
def cb_cur(self, msg: Float32):
    if self.state != STATE_CLOSING:
        return
    v = msg.data
    if v > MAX_VOLT:
        self.over_cnt += 1
        if self.over_cnt >= TRIP_SAMPLES:
            self.get_logger().error(f"!! Overstrøm {v:.2f} V - stopper!")
            self.servo.angle = None
            self._publish_state(STATE_FAULT)
    else:
        self.over_cnt = 0



---


# —— Hovedløkke ——
def loop(self):
    # afslut bevægelse efter MOVE_TIME_S
    if self.move_deadline and time.time() >= self.move_deadline:
        self.move_deadline = 0.0
        if self.final_state:
            self._publish_state(self.final_state)
            self.final_state = None



---


# —— Status-publish helper ——
def _publish_state(self, word: str):
    self.state_pub.publish(String(data=word))
    if word == "opened":
        self.done_pub.publish(Empty())
        self.state = STATE_IDLE
    elif word == "closed":
        self.state = STATE_IDLE
    elif word == STATE_FAULT:
        self.state = STATE_FAULT



---


# —— Oprydning ——

```

---

```

def destroy_node(self):
    self.servo.angle = None
    self.pca.deinit()
    super().destroy_node()

# —— Main-indgang

```

---

```

def main():
    rclpy.init()
    node = GrabNode()
    try:
        rclpy.spin(node)
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == "__main__":
    main()

```

### 13.5.3. Database\_pkg

#### 13.5.1.2. indicator\_node

```

#!/usr/bin/env python3

# signal_node.py
#
_____
# - Venter på trykknap (GPIO16)
# - publicerer std_msgs/Empty på topic '/signal', når knappen trykkes ind
#
_____
# —— Imports
_____
import rclpy
from rclpy.node import Node
from std_msgs.msg import Empty
from gpiozero import Button
#
_____
class SignalNode(Node):
    def __init__(self):
        super().__init__("signal_node")

```

```

# —— hardwareopsætning

self.button = Button(GPIO=16,
                     pull_up=True,
                     debounce_time=0.05)

# —— publisher

self.pub = self.create_publisher(msg=Empty, topic="/signal",
                                 qos_profile=10)

# —— subscriber til reset

self.create_subscription(msg=Empty, topic="/signal_reset",
                        callback=self.cb_reset,
                        qos_profile=10)

# —— registrér callback når knappen trykkes —————
self.button.when_pressed = self.handle_press
self.get_logger().info(" signal_node klar, tryk på knappen!")

# —— Udsend ét Empty-msg ved knaptryk

def handle_press(self) -> None:
    self.pub.publish(Empty())
    self.get_logger().info("Signal sendt.")

def cb_reset(self, msg=Empty) -> None:
    self.get_logger().info("Reset modtaget - klar til nyt knaptryk.")

# —— Main-indgang

def main():
    rclpy.init()
    node = SignalNode()
    try:
        rclpy.spin(node) # kør, indtil Ctrl-C
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == "__main__":
    main()

```

#### 13.5.4. Launch file

```

~/ros2_ws$ mkdir launch
~/ros2_ws/launch$ touch launch.py
~/ros2_ws/launch$ chmod +x launch.py

```

```

from launch import LaunchDescription

```

```

from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        # ----- feedback_pkg -----
        Node(package="feedback_pkg", executable="signal_node", name="signal"),
        Node(package="feedback_pkg", executable="switch_sensor_node", name="switch"),
        Node(package="feedback_pkg", executable="voltage_node", name="voltage"),

        # ----- motor_pkg -----
        Node(package="motor_pkg", executable="linear_actuator_node", name="actuator"),
        Node(package="motor_pkg", executable="grab_node", name="grab"),

        # ----- database_pkg -----
        Node(package="database_pkg", executable="indicator_node", name="indicator"),
    ])

```

Alle koder og forklaringer findes på:

[https://github.com/Skradfy/ros2\\_eksamens\\_kode/tree/main](https://github.com/Skradfy/ros2_eksamens_kode/tree/main)

## 13.6. Rqt\_graph install

```

$ sudo apt update
$ sudo apt install ros-jazzy-rqt-graph
$ sudo apt install ros-jazzy-rqt-gui
$ sudo apt install ros-jazzy-rqt-gui-py
$ source /opt/ros/jazzy/setup.bash
~/ros2_ws $ source/install/setup.bash

```

**SSH X-forwarding for at kunne display grafik i jazzy-base:**

```

$ ssh -X grabbot@192.168.1.xx
$ sudo apt install libxcb-xinerama0 (Qt kræver denne)

```

```
$ source /opt/ros/jazzy/setup.bash
```

kør launch.py filen

```
$ rqt_graph
```

## 13.7. Github

Github bash script til RPI så således ud:

```

#!/bin/bash

sudo apt update && sudo apt upgrade -y
sudo apt install gh
git config --global init.defaultBranch main
gh auth login
gh repo clone Skradfy/3.semesters_eksamen

```

## 13.8. Docker

For enkel installationen blev følgende bash-script anvendt:

```
#!/bin/bash
Docker install
sudo apt-get install docker.io docker-compose
sudo service docker start
```

Dockerfile i src med alle dependencies:

```
FROM ros:jazzy-ros-base

# Set non-interactive mode for APT
ENV DEBIAN_FRONTEND=noninteractive

# Install gpiozero, pigpio, and required Python packages
RUN apt-get update && apt-get install -y \
    python3-gpiozero \
    python3-rpi.gpio \
    pigpio \
    python3-pigpio \
    && apt-get clean

# (Optional) Install via pip if you want the latest python library
RUN pip3 install --no-cache-dir pigpio

# Set up ROS 2 environment for every shell
RUN echo "source /opt/ros/jazzy/setup.bash" >> /root/.bashrc

# Default work directory
WORKDIR /root/rosl2_ws

CMD [ "bash" ]
```

Byg docker image:

```
sudo docker build -t grabbot_env .
```

## 13.9. Ros2 DDS test

I dokumentationen omtales DDS og som default fast DDS og det vil jeg starte med at teste.

Test af overførsels rate på 1 Hz pr sekund

$$I(s) = \frac{1}{rate(Hz)} = \frac{1}{1} = 1 \text{ sek}$$

talker testkode:

```
self.timer = self.create_timer(1, self.publish_cmd)
```

listener:

```
airobot@airobot:~$ ros2 topic hz /right_motor_cmd
average rate: 1.000
    min: 0.998s max: 1.002s std dev: 0.00201s window: 2
average rate: 1.000
    min: 0.998s max: 1.002s std dev: 0.00174s window: 3
average rate: 1.000
    min: 0.998s max: 1.002s std dev: 0.00161s window: 5
average rate: 1.000
    min: 0.998s max: 1.002s std dev: 0.00150s window: 7
average rate: 1.000
    min: 0.998s max: 1.002s std dev: 0.00126s window: 9
```

lav side:  $1,000\text{ s} - 1,998\text{ s} = 0,002\text{ s}$  (2ms)

høj side:  $1,002\text{ s} - 1,000\text{ s} = 0,002\text{ s}$  (2ms)

procentafvigelse:  $\frac{0,002s}{1} \cdot 100\% = +/- 0.2\%$

Test af overførsels rate på 0.200 hz pr sekund

$$I(s) = \frac{1}{rate(Hz)} = \frac{1}{0.200} = 5\text{ sek}$$

Talker testkode:

```
self.timer = self.create_timer(5, self.publish_cmd)
```

Listener:

```
^Cairobot@airobot:~$ ros2 topic hz /right_motor_cmd
average rate: 0.200
    min: 4.997s max: 5.003s std dev: 0.00295s window: 2
average rate: 0.200
    min: 4.997s max: 5.003s std dev: 0.00244s window: 3
average rate: 0.200
    min: 4.995s max: 5.003s std dev: 0.00301s window: 4
average rate: 0.200
    min: 4.995s max: 5.004s std dev: 0.00330s window: 5
average rate: 0.200
```

lav side:  $5,000\text{ s} - 4,995\text{ s} = 0,005\text{ (5ms)}$

høj side:  $5,004\text{ s} - 5,000\text{s} = 0,004\text{ (4ms)}$

$$\text{procentvis afvigelse: } \frac{0,005\text{s}}{5\text{s}} \cdot 100\% = 0,1\%$$

## 13.10. FastDDS Monitor

Install:

```
sudo apt update
sudo apt install openjdk-17-jre
sudo apt install libfreetype6 libxi6 libxtst6
```

```
xhost +local:root
```

```
sudo docker load -i "ubuntu-fastdds-monitor v3.1.0.tar"
```

```
sudo docker run -it --rm --net=host \
-e DISPLAY=$DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
ubuntu-fastdds-monitor:v3.1.0
```

## 13.11. Implementerings problematik

Efter implementering og opstart af Pi5 i systemet meldte terminalen undervoltage ved brug af kommandoen `$ sudo dmesg -T | grep -i voltage`, derfor blev Pi5 implementeret med 5V strømforsyning i stedet, indtil et bedre alternativ på sigt bliver udarbejdet. Efter nærmere undersøgelse giver LM2596 max 2-2,5 A og en Pi5 skal have adgang til 5A. Et alternativ kunne være: Pololu 5V, 5A Step-Down Voltage Regulator D24V50F5.

```
[sudo] password for gradosse:
[Tue Jul 29 11:28:17 2025] mmc0: cannot verify signal voltage switch
[Tue Jul 29 11:28:17 2025] mmc1: cannot verify signal voltage switch
[Tue Jul 29 11:28:23 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:29:29 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:29:31 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:29:33 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:29:35 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:29:37 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:29:39 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:29:49 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:30:32 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:30:34 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:31:05 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:31:07 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:32:00 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:32:04 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:32:27 2025] hwmon hwmon2: Undervoltage detected!
[Tue Jul 29 11:32:29 2025] hwmon hwmon2: Voltage normalised
[Tue Jul 29 11:32:39 2025] hwmon hwmon2: Undervoltage detected!
```