Sensur av hovedoppgaver

Universitetet i Sørøst-Norge
Fakultet for teknologi og maritime fag

Prosjektnummer: **2020-09**
For studieåret: **2019/2020**
Emnekode: **SFHO3201-1 19H Bacheloroppgave**

Prosjektnavn
Forbedring av vindtunnel for Universitetet i Sørøst-Norge

Utført i samarbeid med: Universitetet i Sørøst-Norge

Ekstern veileder: Harald Hovland

Sammendrag:
Oppgaven går ut på å forbedre en vindtunnel som tidligere sto på Vitenskapssenteret på Krona. Vår oppdragsgiver ønsker at vi forbedrer vindtunnelen på en slik måte at den blir egnet til akademisk bruk, slik at studenter i faget Fluidmekanikk kan benytte seg av vindtunnelen til å utføre lab forsøk. Vi har da i oppgave å komme med et nytt design av vindtunnelen som dekker kravene som blir satt med oppdragsgiver.

Stikkord:
- Ombygge vindtunnel
- Akademisk bruk i lab
- Designe nytt konsept

Tilgjengelig: JA

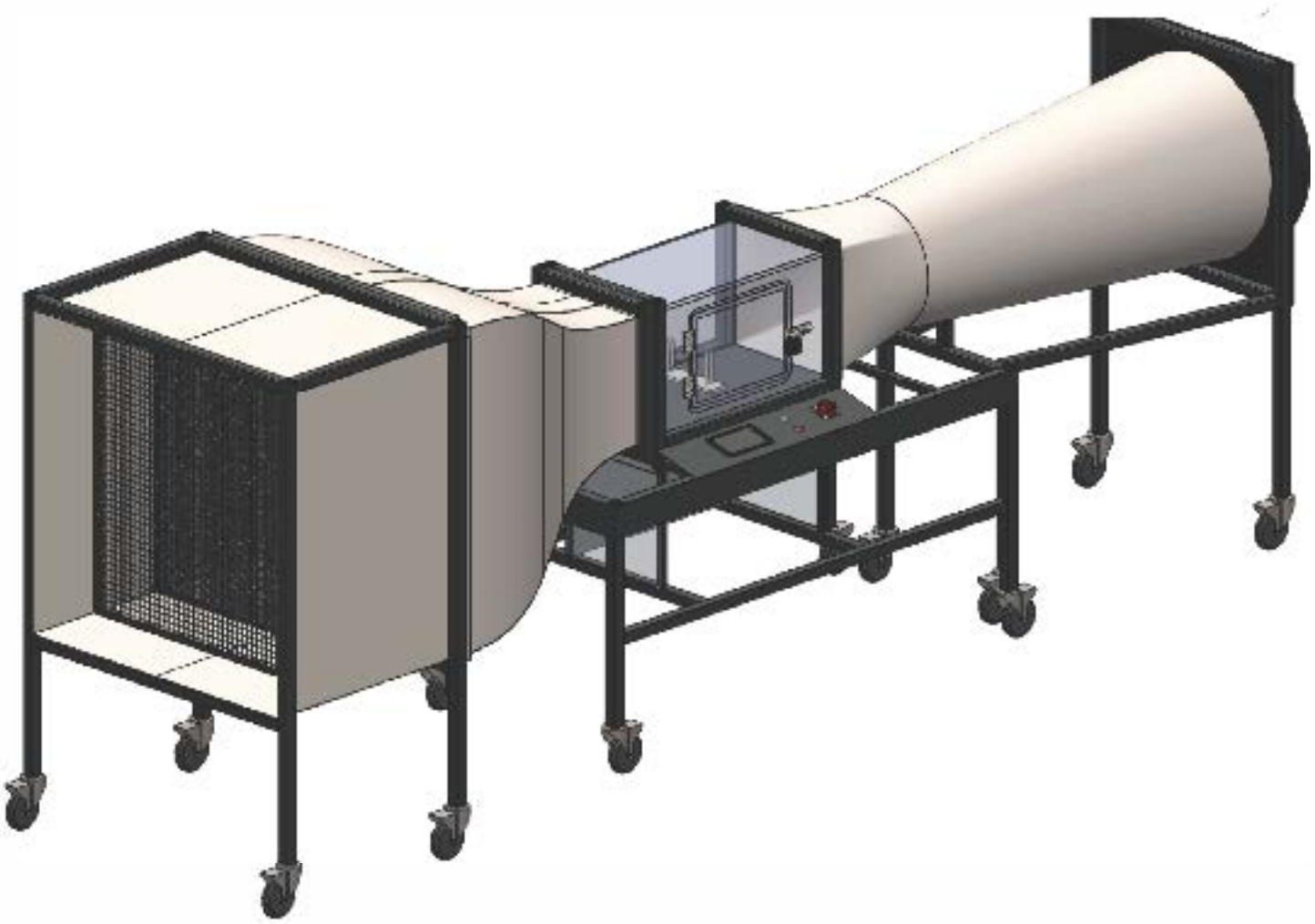
Prosjekt deltagere og karakter:

Navn	Karakter
Steffen Barskrind	
Kristian Auestad	
Joachim Haug	
Håvard Gaska	
Kristoffer Andersen	
Marius Balsvik	

Dato: 15. juni 2020

_____	_____	_____
Jamal Safi	Karoline Moholth	Harald Hovland
Intern Veileder	Intern Sensor	Ekstern Sensor

Forbedring av vindtunnel for Universitetet i Sørøst-Norge



Gruppe 2020-09

Kristoffer Andersen Kristian Auestad Marius Balsvik

Steffen Barskrind Håvard Gaska Joachim Haug

2020-05-24

Universitetet i Sørøst-Norge

2020

Sammendrag

Dette bachelorprosjektet har gått ut på å gjøre mekaniske, elektrotekniske, og informasjonstekniske forbedringer på en vindtunnel som eies av Universitetet i Sørøst-Norge. USN ønsket at vindtunnelen, som ble anskaffet fra Kongsberg Vitensenter, skulle bygges om fra et enkel demonstrasjonsobjekt til en maskin som kan brukes til analyse i fluidmekanikkfaget.

På grunn av COVID-19-pandemien som påvirket og stengte samfunnet gjennom våren 2020, måtte den opprinnelige planen om å gjøre fysiske endringer på vindtunnelen settes til side. Til tross for dette fortsatte gjennomføringen av oppgaven i det teoretiske, med større fokus på design og simulering av en forbedret vindtunnel.

Resultatet av prosjektet har blitt et design for en større og bedre vindtunnel med blant annet bedre luftkvalitet, automatisk styring av vindhastighet, og et interaktivt kontrollpanel som lar brukeren styre og observere funksjonene til vindtunnelen fra ett sted.

Innhold

1. Prosjektgruppa

1.1. Gruppemedlemmer

Steffen Barskrind

Data

Prosjektleder/Risiko

Kristian Auestad

Data

SCRUM master

Joachim Smørdal

Elektro

Dokumentansvarlig

Håvard Gaska

Maskin

Maskinsjef

Kristoffer Andersen

Maskin

Testansvarlig

Marius Balsvik

Maskin

Product owner

1.2. Gruppekontrakt

1.2.1. Arbeidsmiljø og ordninger

Hensikten med gruppekontrakten er å sørge for at vi har et klart og tydelig regelverk å forholde oss til under hele prosjektløpet. Dette regelverket skal sørge for at vi som både gruppe og enkelt individ får et godt arbeidsmiljø, faste arbeidstider, konsekvenser ved forsinket oppmøte uten å si ifra og prosedyrer for konflikthåndtering, se kontrakten på Vedlegg M.

1.2.2. Arbeidstider

Kjernetid for prosjektgruppen er fra klokken 9:15 til 15:15 på mandag, onsdag og fredag hvis ikke annet er fastsatt, men tidene og dagene vil bli litt endret når vi er ferdige med det ene faget vi har ved siden av bacheloroppgaven. Om det nærmer seg tidsfrister for viktige innleveringer, presentasjoner og milepæler så kan det bli satt opp overtid på kjernedagene eller helgejobbing.

Del I.

Oppgaven og metodikk

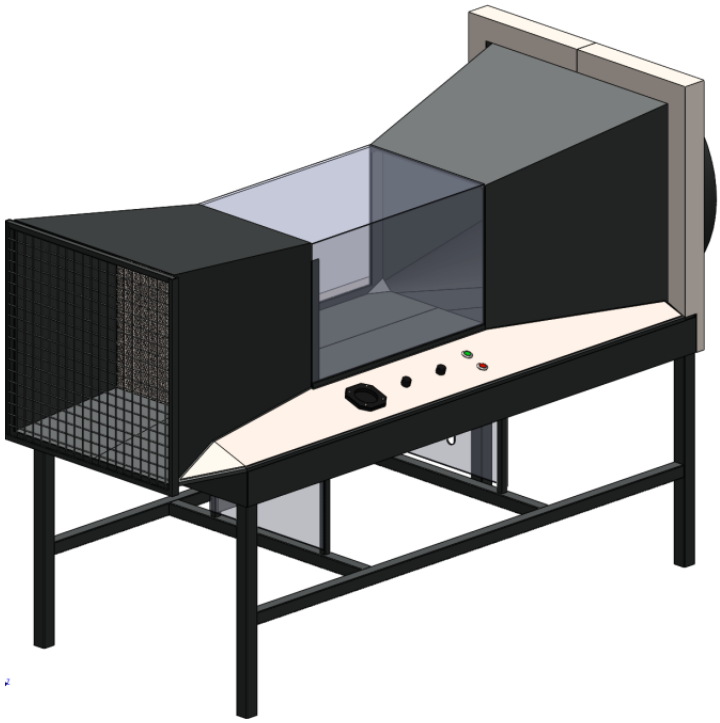
2. Oppdraget

2.1. Oppdragsgiver

Universitetet i Sørøst-Norge (USN) er et norsk universitet som ble etablert 4. Mai 2018. USN er det fjerde største universitetet i Norge med over 18.000 studenter og ca. 1600 ansatte. Universitetet har åtte campuser som befinner seg i Bø, Porsgrunn, Notodden, Rauland, Drammen, Hønefoss, Kongsberg og Horten. Vår oppdragsgiver fra USN er Kjell Enger som er foreleser for maskiningeniørene innen fluidmekanikk, konstruksjonsteknikk, statikk med fasthetslære, og faget subsea ved campus Kongsberg.

2.2. Problemstilling

Universitetet i Sørøst-Norge campus Kongsberg har mottatt en vindtunnel som tidligere har blitt brukt til demonstrasjoner på Kongsberg Vitensenter (se Figur 2.1). Den eksisterende vindtunnelen er ikke egnet for laboratorieforsøk. Oppdragsgiver ønsker en oppgradering av vindtunnelen slik at den kan bli brukt av studenter i laboratoriearbeid. Det fulgte også med en vindmølle som oppdragsgiver ønsket en oppgradering på, men etter avtale med oppdragsgiver lenger ut i prosjektet ble vi enige om å se bort ifra den hvis vindtunnel prosjektet viser seg å være tidkrevende.



Figur 2.1.: Vindtunnelen fra Kongsberg Vitensenter(Bilde tatt i lab Vedlegg E)

2.3. Oppgavebeskrivelse

Oppgaven går ut på å forbedre en vindtunnel som tidligere sto på Kongsberg vitensenter. Vår oppdragsgiver ønsker at vi forbedrer vindtunnelen på en slik måte at den blir egnet til akademisk bruk, slik at studenter i faget fluidmekanikk kan benytte seg av vindtunnelen til å utføre lab forsøk. For å få til dette må det eksisterende systemet dokumenteres slik at vi kan kartlegge systemets nytteverdi og forbedringsmuligheter. Oppdragsgiver ønsker en vindtunnel for praktisk bruk, som gir korrekte måleresultater, visualisert luftstrøm og et brukervennlig brukergrensesnitt.

Noen av hovedpunktene ved oppgaven er:

- Vindtunnelen skal være i stand til å gjøre de målinger som er nødvendig for laboratoriearbeid.

- Vindtunnelen skal ha en innfestning som passer for ulike testmodeller.
- Luftstrømmen skal kunne visualiseres.
- Måledata skal kunne avleses, lagres, og presenteres digitalt på en datamaskin.

2.4. Målsetting for prosjektet

Vår gruppe har som primærmål å videreutvikle vindtunnelen i henhold til de kravene med høyest prioritet som vi har formulert utifra oppgavebeskrivelsen, oppdragsgivers ønsker og forslag fra prosjektgruppen. Vi har som mål å ferdigstille systemet med disse kravene innenfor tidsfristen. Hvis tiden strekker til, har vår gruppe som sekundærmål å fullføre de kravene som har lavere prioritet når det gjelder vindtunnelen.

2.4.1. Covid-19

Dette prosjektet var originalt ett praktisk prosjekt der målet var å utvikle ett fysisk sluttprodukt. Grunnet Covid-19 utbruddet, så har skolen vært nødt til å stenge bygget, og dermed har vi ikke fått tilgang til å jobbe på vindtunnelen. Derfor har oppgaven våres blitt omgjort til å være en teoretisk oppgave, som vil si at vi skal tilrettelegge for den neste prosjektgruppen som skal jobbe med denne oppgaven.

3. Prosjektmodell

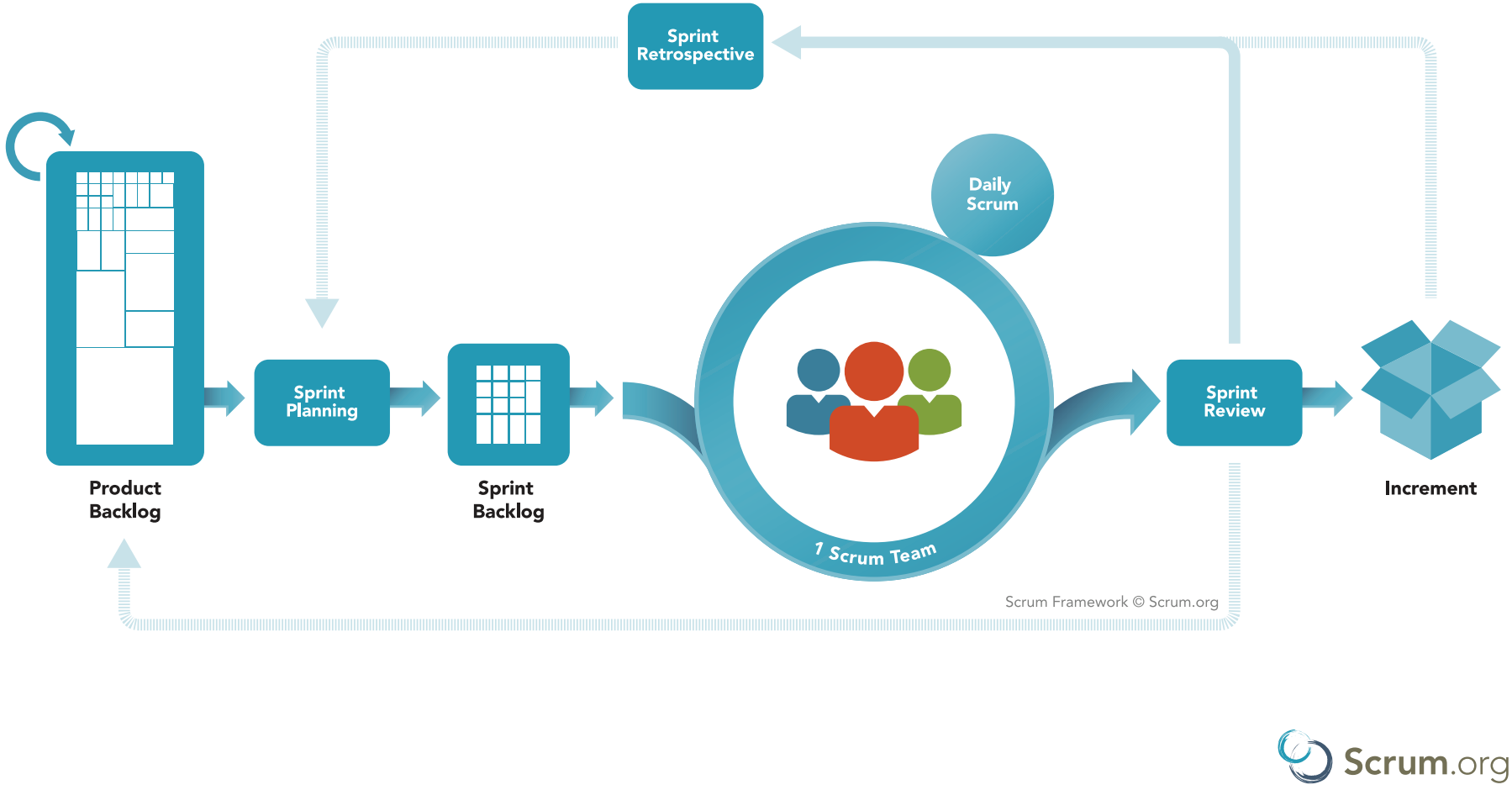
Vi valgte å bruke Scrum som prosjektmodell. Grunnlaget for dette valget er todelt: For det første er det den modellen gruppa har størst erfaring med, totalt sett, slik at vi trenger minst mulig tid på å tilpasse oss arbeidsprosessen. For det andre er Scrum en iterativ prosess som lar oss gjennomføre oppgaven i små sprang. Med tanke på hvor vagt definert det endelige målet for oppgaven er på dette stadiet, gir det mening for oss å kunne jobbe på denne måten.

3.1. Scrum

Vår forståelse av Scrum kommer hovedsakelig fra forelesninger på skolen og *The Scrum Guide*[27].

Scrum er et rammeverk for å utvikle, levere, og opprettholde komplekse produkter som opprinnelig ble brukt til softwareutvikling, men som etterhvert har fått større anerkjennelse i andre industrier. I Scrum blir en kompleks oppgave lett delt inn i små, fordøyelige deloppgaver som blir utført i løpet av korte arbeidsperioder kalt **sprints**. På slutten av hver sprint skal det, i prinsippet, foreligge et fungerende produkt som oppfyller i hvertfall endel av kravene som ble satt for oppgaven. Scrum-prosessen vises i Figur 3.1.

En Scrum-gruppe består av et **development team**, som er utviklerne på prosjektet; en **product owner**, som er bindeleddet mellom oppdragsgiver og development teamet; og en **Scrum mas-**



Figur 3.1.: Scrum-prosessen [28]

ter, som har ansvaret for å styre sprintene og sørge for at development teamet følger Scrum-prosessen.

3.1.1. Sprint

En sprint varer som regel fra 1-4 uker, avhengig av hvor stort prosjektet og prosjektgruppa er, og hvor stort kravet til hurtig iterering er. Vi har valgt å bruke 2 uker på hver sprint for å få en balanse hvor vi har nok tid til å gjennomføre en iterasjon, og nok iterasjoner til å kunne gjennomføre prosjektet. Grovplanleggingen av sprintene er satt opp i et Gantt-diagram (vist i Avsnitt 3.3).

Hver sprint starter med et planleggingsmøte, **sprint planning** hvor vi velger hvilke oppgaver fra **product backlog** som skal tas med i sprinten og legges i **sprint backlog**. Product backlog er lista over alle oppgavene som må gjøres for å gjennomføre prosjektet, mens sprint backlog er de oppgavene vi skal ta for oss i denne sprinten.

Hver dag i sprinten har vi et kort møte kalt **daily Scrum**. Dette møtet skal være maks 15 minutter hvor alle deltakerene står, der hver person beskriver kort hva de har jobbet med siden siste møte, hva de skal jobbe med videre, og om de trenger hjelp til å fullføre. Hensikten med daily Scrum er å holde gruppemedlemmene à jour med fremdriften til gruppa.

For å spore og holde styr på sprinter, oppgaver, og arbeidstid benytter vi et system som heter **Axosoft**, som er beskrevet i Avsnitt 3.5.4. For å holde bedre styr på tidsbruk og tidslogging for Scrum-oppgavene bruker vi **Axosoft Stopwatch**, et tilleggsprogram som automatisk sporer tid brukt på en oppgave i Axosoft (Avsnitt 3.5.10).

For å anslå hvor godt vi ligger ann i sprinten benytter vi også en graf kalt et **burndown chart**, som viser hvor mange av arbeidstimene vi har fullført. Denne grafen produserer en trendlinje

som gir en indikasjon på om vi kommer til å møte tidsfristen for sprinten. Eksempel i figur 3.5 på side 21.

3.2. Prosessen

3.2.1. Stakeholderkrav

Det første overordnede i prosessen er kravene vi får fra stakeholdere og via oppdragsgiver. Disse blir skrevet inn i dokument for kravspesifikasjoner og gitt en ID for sporing: `SK 1.xx` for krav til vindtunnel, og `SK 2.xx` for krav til vindmølla. Disse kravene noteres så ordrett som mulig fra kommunikasjon med stakeholder slik at vi vet hva som opprinnelig ble sagt. Et utdrag fra lista over stakeholderkrav vises i Tabell 3.1.

Stakeholder Krav-ID	Krav	Opphav
SK 1.1	Det skal være lett å bytte testmodeller i vindtunnelen.	Kjell
SK 1.2	Innfestningen til testmodeller skal være solid, enkel i bruk, og repeterbar.	Kjell
SK 1.3	Data som måles skal kunne avleses og presenteres grafisk.	Kjell
SK 1.4	Avleste data skal kunne leses som et Excelark.	Kjell
SK 1.5	Vindhastigheten ved testmodellen skal leses; avlesning skal skje foran modellen.	Kjell
SK 1.6	Luftmotstandskraften på testmodell skal leses.	Kjell
SK 1.7	Orientering til testmodellen skal leses.	Kjell
SK 1.8	Lufttemperatur i testkammeret skal leses.	Kjell
SK 1.9	Luftfuktighet i testkammeret skal leses.	Kjell
SK 1.10	Lufttrykk i testkammeret skal leses.	Kjell
SK 1.11	Avlesninger skal ikke påvirkes av vibrasjoner fra vifte/motor/e.l.	Kjell
SK 1.12	Signaler fra sensorer må filtreres eller forsterkes om nødvendig.	Kjell
SK 1.13	Dersom vibrasjoner ikke kan fjernes skal innfestningen utstyres med akselerometer for å kunne måle for vibrasjoner i målingene.	Kjell

Tabell 3.1.: Utdrag fra lista over stakeholderkrav.

3.2.2. Kravanalyse

Oppfatningen vår av de overordnede kravene blir satt opp på diagramform som vi kan bruke til å trekke frem mer detaljerte krav i et hierarki.

Hovedfokuset for kravanalysen har vært:

- Vindtunnelen skal være sikker for bruker og andre i nærheten.
- Det skal kunne eksperimenteres med en testmodell i vindtunnelen.
- Det skal kunne utføres målinger på vindtunnelen.
- Vindtunnelen skal være enkel å flytte.
- Måledata fra sensorer på vindtunnelen skal vises og overføres til PC.

3.2.3. Krav

Ut fra kravanalysen identifiserte vi kravene som skal styre utviklingen og brukes som basis for verifiseringen av produktet. Disse samles i kravspesifikasjonen og grupperes i følgende kategorier:

HMS – Krav til Helse, miljø, og sikkerhet.

VT – Krav til vindtunnelen.

TM – Krav til testmodellen.

M – Krav til målinger i vindtunnelen og på testmodellen.

DB – Krav til behandling, visning, og overføring av måledata.

VM – Krav til vindmølla.

Krav gis en prioritet fra A til C som representerer hvor viktig vi mener det er å tilfredstille det:

A: Viktig; vi legger planer på grunnlag av å implementere disse kravene.

B: Ønskelig; vi ønsker å oppfylle disse kravene, men de må vike for A-kravene.

C: "Kjekt å ha"; hvis vi får tid.

I tillegg bruker vi "prioriteten" **X** for krav som har blitt slettet.

ID til krav er på formen `K.GG xx` der `GG` representerer kategorien og `xx` er et løpenummer.

Et utdrag fra kravlisten vises i Tabell 3.2, og hvorvidt kravene har blitt oppfylt diskuteres i del IV på side 279.

Krav ID	Krav	Prioritet	Stakeholder krav ID	Testmetode	Test ID
K.VT 02	Det skal være jevn (+/- 6%) vindhastighet over målesnittet (snittet i testkammeret der lufta møter testmodellen) i test-kammeret.	A		M	T.VT 02
K.VT 04	Lufthastigheten i testkammeret skal kunne settes av bruker.	A	SK 1.14	F	T.VT 04
K.VT 05	Dimensjoner på testkammeret skal være minst 40cm x 40cm x 40cm (+/- 20%).	B	SK 1.49	M	T.VT 05
K.VT 06	Brukermanual og vedlikeholdsmanual skal lages for vind-tunnelen.	A	SK 1.27	R	T.VT 06
K.VT 08	Vindtunnelen skal kunne flyttes av én person uten hjelpe-midler.	A	SK 1.22	F/R	T.VT 08
K.VT 09	Vindtunnelen må ikke ha eksterne mål større enn 210 x 85 x 190 cm (LxBxH), og vekt større enn 800 kg, for å kunne transporteres til måleteknisk lab og i heisene på Krona.	A	SK 1.55, SK 1.57	R/M	K.VT 09
K.VT 09.03	Dersom maksverdiene for eksterne mål og vekt ikke kan overholdes, skal vindtunnelen kunne deles i moduler som overholder kravene individuelt.	A	SK 1.22, SK 1.55	M/F	T.VT 09.03
K.VT 10	Luftstrømmen i vindtunnelen skal kunne visualiseres ved hjelp av røyk(skjer i den laminære delen).	A	SK 1.15	R/F	T.VT 10

Tabell 3.2.: Utdrag fra kravlisten.

3.2.4. User stories og tasks

En **user story** er en kort beskrivelse av en funksjon eller oppgave som systemet skal oppfylle, skrevet på en naturlig måte fra perspektivet til brukeren av produktet eller en av stakeholderne til prosjektet. Vi har valgt å bruke formen "Som <rolle> skal jeg kunne utføre <funksjon>". Se Tabell 3.3 for eksempler.

User stories lages utifra kravene til prosjektet, og brukes som det øverste laget i Scrum-prosessen. Under hver user story oppretter vi én eller flere **tasks**, som er spesifikke oppgaver som vi utfører for å oppfylle ønsket til brukeren som gitt i user storyen.

Vi gir user stories ID på formen `US 1.xx`, hvor `xx` er løpenummeret.

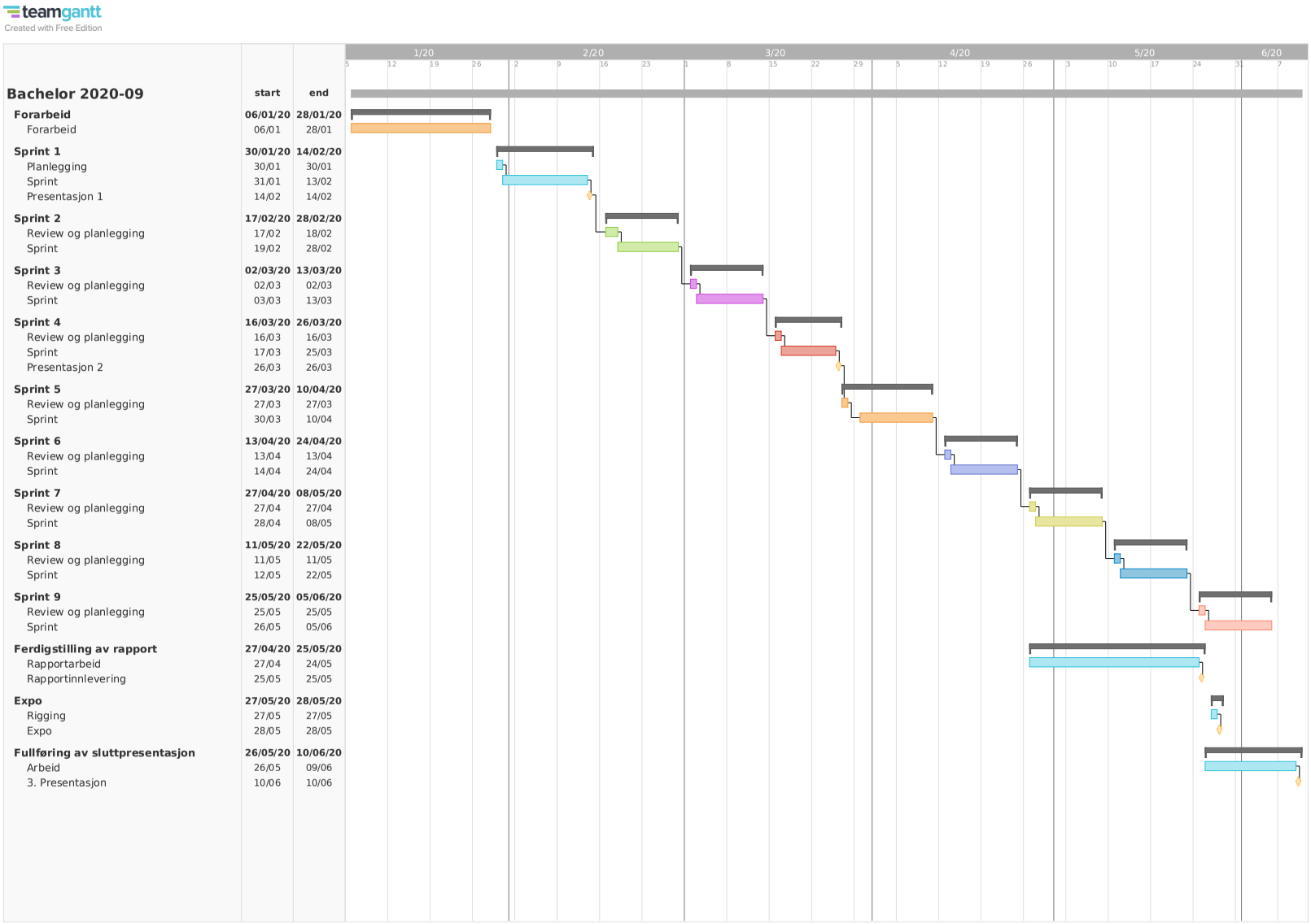
3.3. Gantt-diagram

Det opprinnelige Gantt-diagrammet, som vist i Figur 3.2, ble opprettet i begynnelsen av prosjektet.

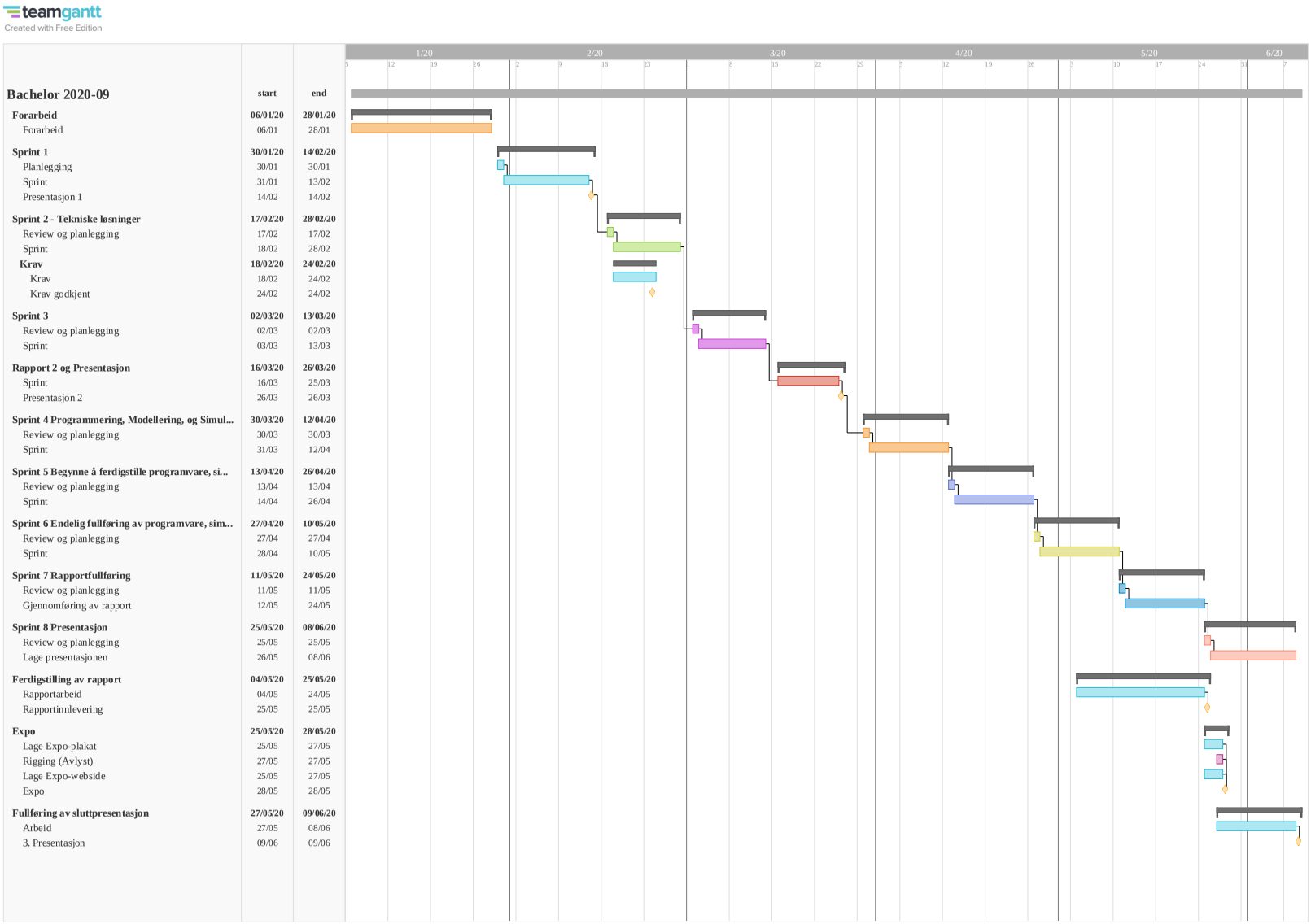
Etter presentasjon 2 ble prosjektplanen revidert til 8 sprinter istedenfor 9, da tempoet vi hadde forestilt oss fra begynnelsen ikke viste seg å være oppnåelig. Det reviderte Gantt-diagrammet vises i Figur 3.3, og en større versjon er vedlagt i Vedlegg C.

3.4. Utfordringer med prosessen

Arbeid med Scrum har vært både nyttig og utfordrende.



Figur 3.2.: Opprinnelig Gantt-diagram



Figur 3.3.: Endelig Gantt-diagram

User story ID	**Ref. Krav ID**	**User story**
US 1.01	K.M 01, K.DB 02	Som bruker skal jeg kunne lese lufthastigheten i testkammeret.
US 1.02	K.VT 04	Som bruker skal jeg kunne justere lufthastigheten i testkammeret.
US 1.03	K.M 02, K.DB 02	Som bruker skal jeg kunne lese lufttrykket i testkammeret.
US 1.04	K.M 03, K.DB 02	Som bruker skal jeg kunne lese lufttemperaturen i testkammeret.
US 1.05	K.M 04, K.DB 02	Som bruker skal jeg kunne lese luftfuktigheten i testkammeret.
US 1.06	K.M 07, K.DB 02	Som bruker skal jeg kunne lese pitchvinkelen til testmodellen.

Tabell 3.3.: Utdrag fra User stories.

På den ene siden har vi hatt god nytte av sprintene. Det å ha en arbeidsperiode hvor man kan konsentrere seg om arbeidet, og så ha en periode hvor vi kan gå gjennom hva vi har gjort, og hva som skal gjøres videre. Det har vært en god måte å jobbe på.

På den andre siden har vi ikke vært så gode som vi burde på å dele oppgavene i fullt så små deler som vi burde. Dette har ført til at en del oppgaver har blitt hengende gjennom flere sprinter. Selv om de har blitt jobbet på, så har omfanget av oppgaven ikke gjort oss i stand til å fullføre den før senere.

Vi ble, etterhvert, bedre på å splitte opp oppgavene i mindre deler og så fordele dem, men med etterpåklokskap om hvordan gruppa fungerte hadde det kanskje vært bedre å bruke en arbeidsprosess med en mer sentral oppgavefordeling.

3.5. Verktøy

For å styre og gjennomføre prosjektet bruker vi forskjellige verktøy:

3.5.1. Discord

For vanlig kommunikasjon mellom gruppemedlemmer har valgte vi å bruke chatprogrammet Discord. Valget kommer av at alle gruppemedlemmene bruker det fra før og er fornøyde med det. Discord gir oss enkle muligheter for å sende og motta skriftlige beskjeder, og lar oss dele filer, bilder, og linker.

Med tanke på den store mengden hjemmearbeid vi har måttet gjøre, har vi brukt Discord mer enn vi trodde. Så og si alt av samtaler vi normalt hadde hatt ansikt til ansikt har blitt gjort via Discord.

3.5.2. Skype

Vi brukte Skype i små mengder på begynnelsen av prosjektet for å snakke med gruppemedlemmer som ikke kunne komme til skolen, siden tale via Discord ikke fungerer på skolens nettverk.

Etter at skolen ble stengt har vi ikke brukt Skype, delvis fordi det ikke lenger var nødvendig, men mest fordi skolen gikk over til å bruke Zoom isteden.

3.5.3. Zoom

Zoom er et online møte- og konferanseverktøy som tok fullstendig av etter at COVID-19-pandemien begynte å holde alle hjemme. Skolen anskaffet en masselisens som alle studenter og ansatte kan benytte seg av.

Vi gikk over til å bruke Zoom for alle videokonferanser og møter med veileder, oppdragsgiver, og sensor, samt at vi brukte det til presentasjon 2.

3.5.4. Axosoft

Til å organisere arbeidsoppgavene og sprintene i Scrum har vi brukt online-verktøyet Axosoft.

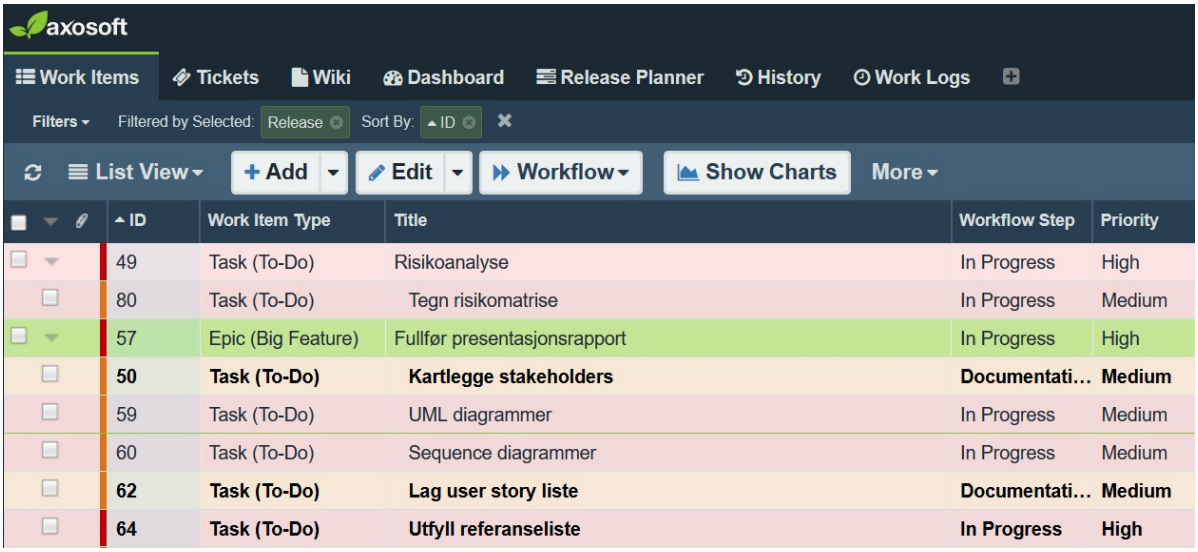
Axosoft gir oss lister over user stories og tasks, Scrum board, sprintplanlegging, assistent for daily Scrums, burndown charts, mm.

Etter flere måneders bruk føler vi at Axosoft har vært litt mindre nyttig enn vi hadde håpet. Evnen til å koble sammen oppgaver er dårligere og mindre intuitivt enn, for eksempel, Azure Devops.

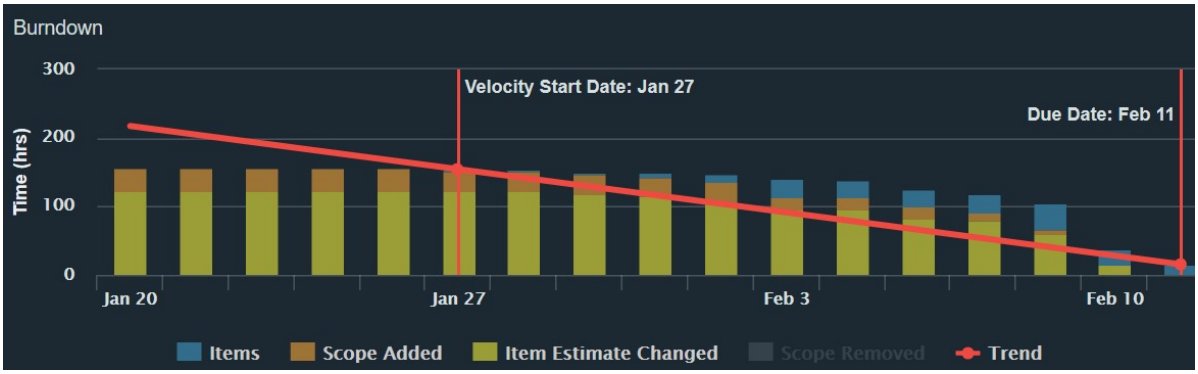
3.5.5. Axosoft Stopwatch

For å måle hvor mye tid som brukes på hver arbeidsoppgave bruker vi Axosoft Stopwatch, som er et eget program hvor man kan logge tiden man bruker på hver task i Axosoft ved hjelp av en innebygd stoppeklokke.

Med dette programmet har vi kunnet dokumentere arbeid man har gjort og få mer nøyaktig dokumentasjon på arbeidstimene til gruppa.



Figur 3.4.: Skjermbilde fra Axosoft



Figur 3.5.: Burndown chart i Axosoft

3.5.6. Clockify

Vi har brukt Clockify-tjenesten som et mer generelt stemplingsur for å holde ta tiden på hele arbeidsdagen, i motsetning til arbeidet på hver task.

3.5.7. Google Drive

Vi har brukt en felles mappe på Google Drive til å lagre og dele alle filer, diagrammer, notater, og lister vi har produsert.

3.5.8. Overleaf

Overleaf er et online-verktøy for å samarbeide om skriving av LaTeX-dokumenter. Vi bruker det til å skrive rapporter, møtereferater, ukerapporter, og andre tekstdokumenter.

3.5.9. Lucidchart

Lucidchart er en tjeneste for å tegne diagrammer og figurer, som vi har brukt til å tegne størsteparten av diagrammene og figurene til prosjektet.

3.5.10. SolidWorks

Til å lage 3D-modeller og sammenstillinger har vi valgt å bruke SolidWorks, siden det er dette DAK-programmet som maskiningeniørene er vant til å bruke.

Inne i SolidWorks har vi brukt SolidWorks Simulation til fluidsimulering av modeller og stressanalyse.

3.5.11. Matlab og Simulink

Matlab og Simulink ble brukt til å gjøre beregninger og simulasjoner på styringssystemet.

3.5.12. KiCad

Planen var å bruke OrCad som er det kretskortdesign programmet som vi bruker på skolen. Men siden den gamle PC-en vi hadde med OrCad lisens sluttet å virke, så gikk vi til innkjøp av ny PC. Grunnet tidsbegrensninger fikk vi ikke mulighet å ordne ny OrCad lisens, derfor var vi nødt å bruke gratisprogramvaren KiCad istedenfor for design av kretskortene til dette prosjektet.

3.5.13. LTspice

For simulering og testing av elektriske kretser har vi brukt kretssimuleringsverktøyet LTspice.

3.5.14. Git og GitHub

Git er et versjonskontrollsystem for kildekode som lar flere utviklere jobbe sammen på et prosjekt. Vi har brukt det under utviklingen av programvaren til den innebygde datamaskinen.

GitHub er et websted hvor brukere kan opprette en sentral lagringsplass, eller repositorium, for kildekoden til prosjekter. Dette har latt oss jobbe på den samme koden fra flere datamaskiner samtidig.

3.5.15. 3d Studio Max

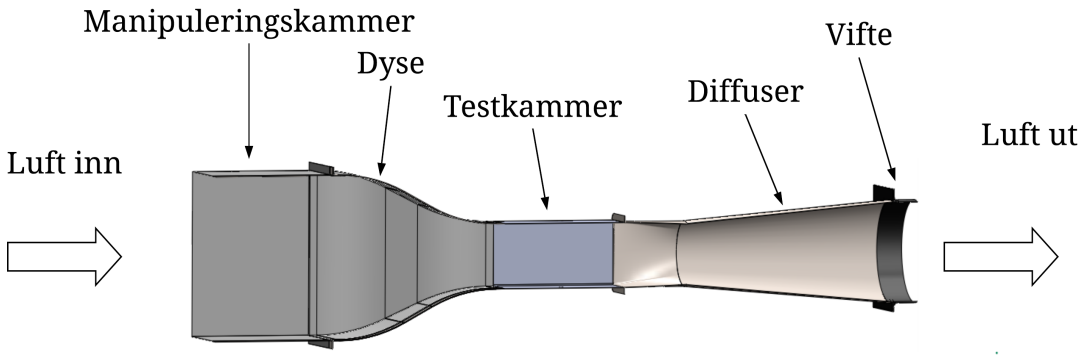
Programvare for 3D-modellering. Har i vårt tilfelle blitt brukt til å importere modeller fra SolidWorks og rendre mer realistiske bilder.

4. Vindtunneler

4.1. Om forskjellige vindtunneler

4.1.1. Intro

En vindtunnel er et redskap som brukes til å teste, forske på, og visualisere hvordan luftstrømmer påvirker objekter under kontrollerte forhold. Som regel er det en form for vifte som trekker eller blåser luft gjennom en strømretter (honeycomb og/eller skjermer), slik at man får en kontrollert luftstrøm som så trekkes gjennom en dyse (nozzle). Luftstrømmen fortsetter inn i testseksjonen, også kalt testkammeret, man vanligvis har en laminær (stødig/jevn) luftstrøm som møter testobjektet og strømmer forbi den. Til slutt kommer lufta ut av testseksjonen til en spreder (diffuser), og ut bak viften som har trekket lufta gjennom. Dette er de helt grunnleggende komponentene i vindtunneler hvor hensikten er undervisnings- og forskningsrelatert. Se Figur 4.1 for en illustrasjon av grunnkomponentene.



Figur 4.1.: Illustrasjon av grunnelementene i en vindtunnel, tverrsnitt.

4.1.2. Typer

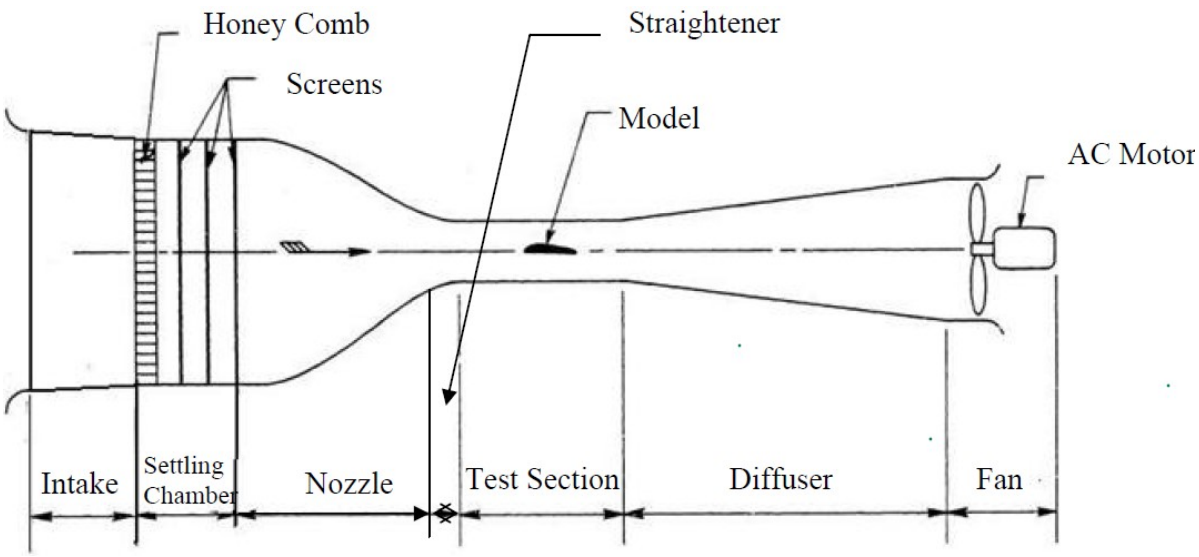
Vindtunneler for forskning, i motsetning til de for underholdning (f. eks. vertikale vindtunneller hvor man kan sveve innendørs), kan deles inn i forskjellige typer som funksjon av vindhastigheten de utsetter testmodeller for. Den mest relevante typen med hensikt til vindhastighet er subsoniske vindtunneler siden vi skal designe for hastigheter under lydens hastighet. Det vil si vindhastighet under Mach 1, som er ca 344 $\mathrm{m/s}$ ved standard forhold som 21 grader celsius og 0 meter høyde. En vindtunnel med vindhastighet fra 0 til 0.4 Mach klassifiseres som lavhastighets-subsonisk, og mellom 0.4 og 0.75 Mach som høyhastighets-subsonisk [1].

Åpen vindtunnel

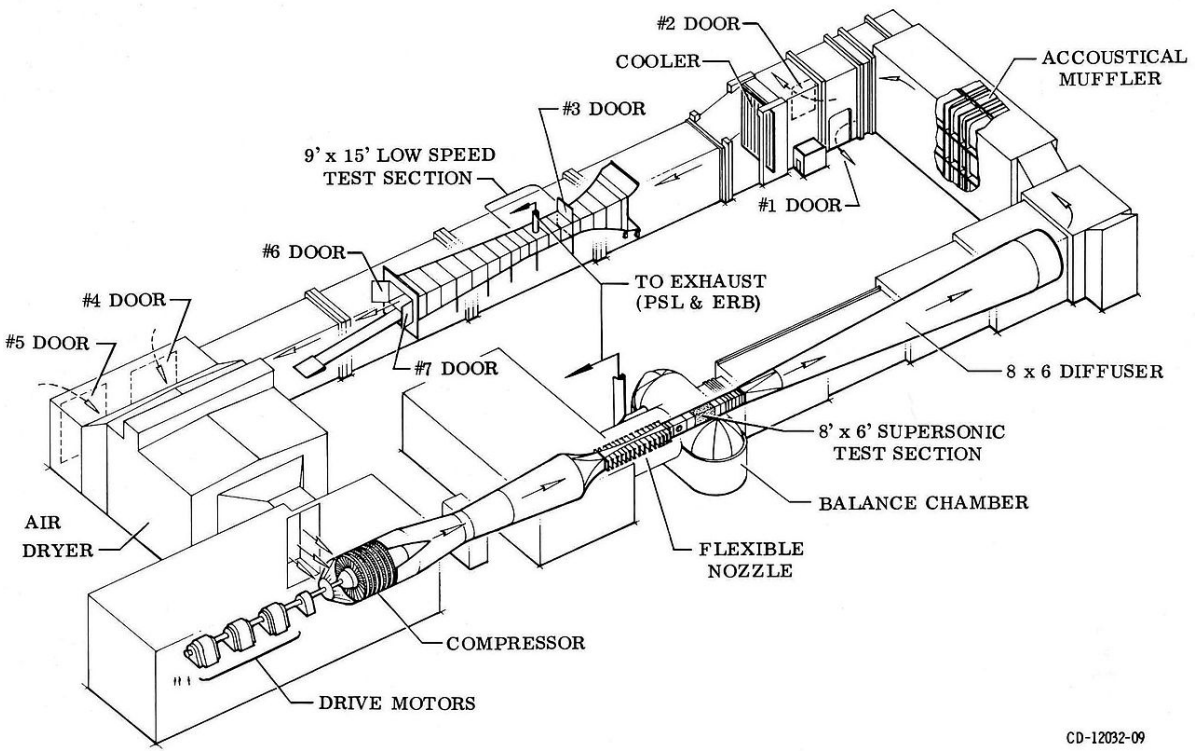
I en åpen vindtunnel tas luften fra systemets omgivelser og slippes ut i andre enden igjen. Denne typen er relativt billig i konstruksjon, og er mer påvirket av vind og temperatur i omgivelsene enn et lukket system. Den er billigere i drift siden det er mindre deler som trenger strøm og den trenger mindre plass.

Lukket vindtunnel

I et lukket system blir luften resirkulert innvendig flere ganger, og tunnelen er ofte helt isolert fra omgivelsene. Her er det lettere å kontrollere størrelser som temperatur, vindhastighet, og det kan oppnås mer laminær luftstrøm. Denne typen vindtunnel er plasskrevende, og mer krevende i konstruksjon. Det vil også trenges et dedikert kjølesystem siden luften i utgangspunktet går i konstant sirkulasjon og tar i mot gradvis mer varme fra komponentene. Ved bruk av røyk til visualisering av luftstrømmen vil det også bli vanskelig å fjerne den.



Figur 4.2.: En åpen vindtunnel som tar inn luft fra omgivelsene [13].



Figur 4.3.: En lukket vindtunnel som gjensirkulerer luft og er mindre påvirket av omgivelsene [32].

4.1.3. Modeller / Målinger

Nytteverdien av vindtunneler er først og fremst å overvåke og undersøke luftstrømmens påvirkning på forskjellige geometriske modeller. For eksempel brukes ofte små skalamodeller av luftfartøy og vinger, eller biler som krever lite luftmotstand (Formula 1). Man kan også observere vibrasjoner som virker på brodekke, men for å simulere tilsvarende krefter på en skalamodell i forhold til det den er modellert etter må man bruke en høyere lufthastighet. For eksempel, hvis du har en skalamodell som er 5 ganger mindre enn det du skal undersøke så må du bruke 5 ganger høyere hastighet på modellen enn det du ville ha brukt i full skala for å få tilsvarende utslag. Dette er fordi skalamodellen vil ha 5 ganger mindre areal som blir utsatt for krefter, og man må dermed kompensere med 5 ganger høyere vindhastighet for å få tilsvarende Reynoldstall [10].

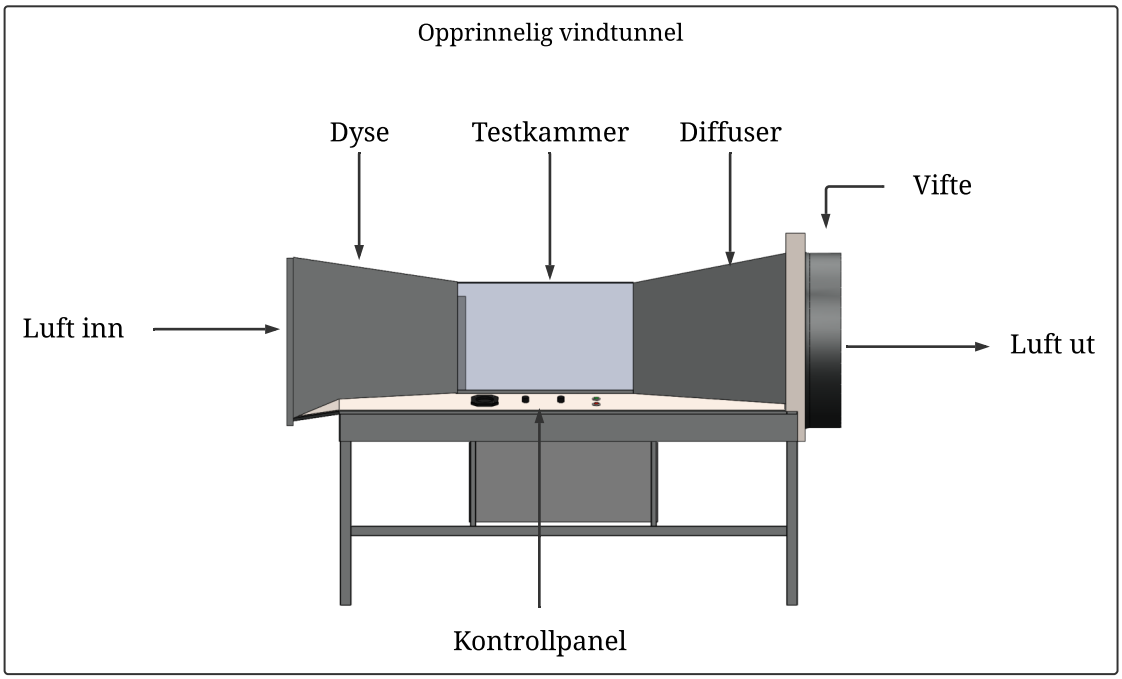
Ved disse testene er man interessert i analytiske data, dette inkluderer men er ikke begrenset til: Trykk, lufthastighet, turbulens, og kreftene modellen er utsatt for. For å måle størrelser som krefter, trykk og vindhastighet trenger man en rekke sensorer og forskjellige komponenter.

Det er også vanlig med diverse visualiseringsverktøy. Røyk er det vanligste hjelpemiddelet, siden testkammeret vanligvis er dekket av glass eller gjennomsiktig plast kan man lett se hvordan strømmen treffer modellen. Hjelpemiddler som høyhastighetskamera som tar mange bilder i sekundet blir også brukt for å dokumentere ting og hendelser det blotte øye ikke kan se.

4.2. Opprinnelig vindtunnel

Vindtunnelen som vi fikk av USN er en enkel lavsubsonisk type som betyr at oppnådde lufthastigheter er under lydens hastighet. Slik den står i dag er de ytre dimensjonene 211 cm x 137 cm

x 81 cm L/H/B. Vist i Figur 4.4, fra venstre består den av en dyse, et testkammer, etterfulgt av en spreder hvor innfestningen til en vifte befinner seg.

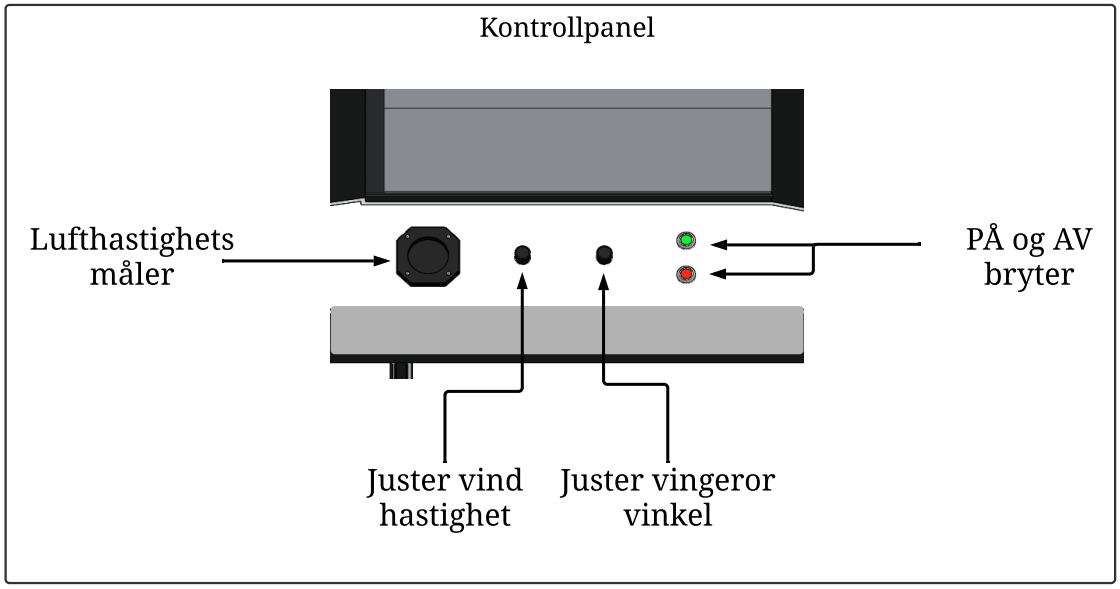


Figur 4.4.: Modell av den opprinnelige vindtunnel.

Tunnelen fungerer ved at vifta trekker luften på innsiden av systemet ut til høyre. Dette skaper et undertrykk inne i tunnelen som gjør at luften ellers i rommet beveger seg hit. Den eneste måten luften kan bevege seg inn er gjennom åpningen i dysen til venstre. Denne luften vil da igjen bli trukket ut gjennom viften i systemet og vi har et kontinuerlig kretsløp.

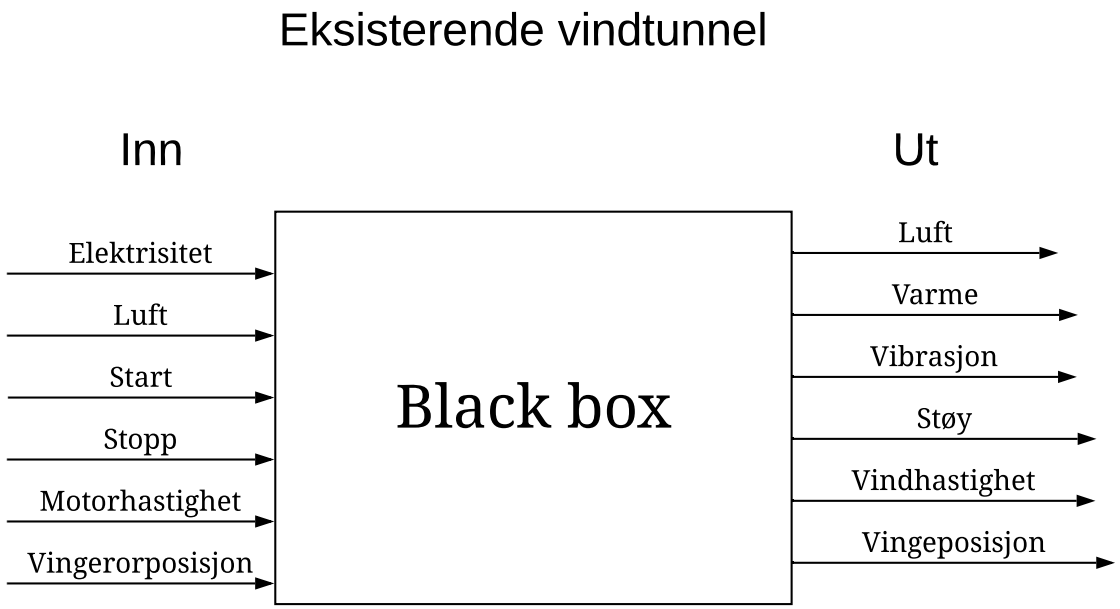
I testkammeret i midten befinner det seg en vingeprofil og hensikten med systemet så langt har vært at brukeren har kunnet justere både vindhastighet i kammeret og vinkelen på vingeroret (se Figur 4.5) for å få vingen til å heve seg vertikalt. I testkammeret befinner det seg også et anemometer som er en analog vindmåler som sender en analog respons til en instrumentmåler på kontrollpanelet. Testkammeret består av gjennomsiktig plexiglass som dekker både sidene og toppen av kammeret.

Plexiglasset kan løftes av vindtunnelen etter at man har løsnet skruene som holder et par plater og plexiglasset fast på plass oppå vindtunnelen.



Figur 4.5.: Kontrollpanel på den opprinnelige vindtunnelen.

Figur 4.6 viser det eksisterende systemets inngang/utgang parametere. Den tomme boksen representerer selve systemet som ses bort ifra i dette diagrammet.



Figur 4.6.: I/O oversikt for den eksisterende vindtunnelen.

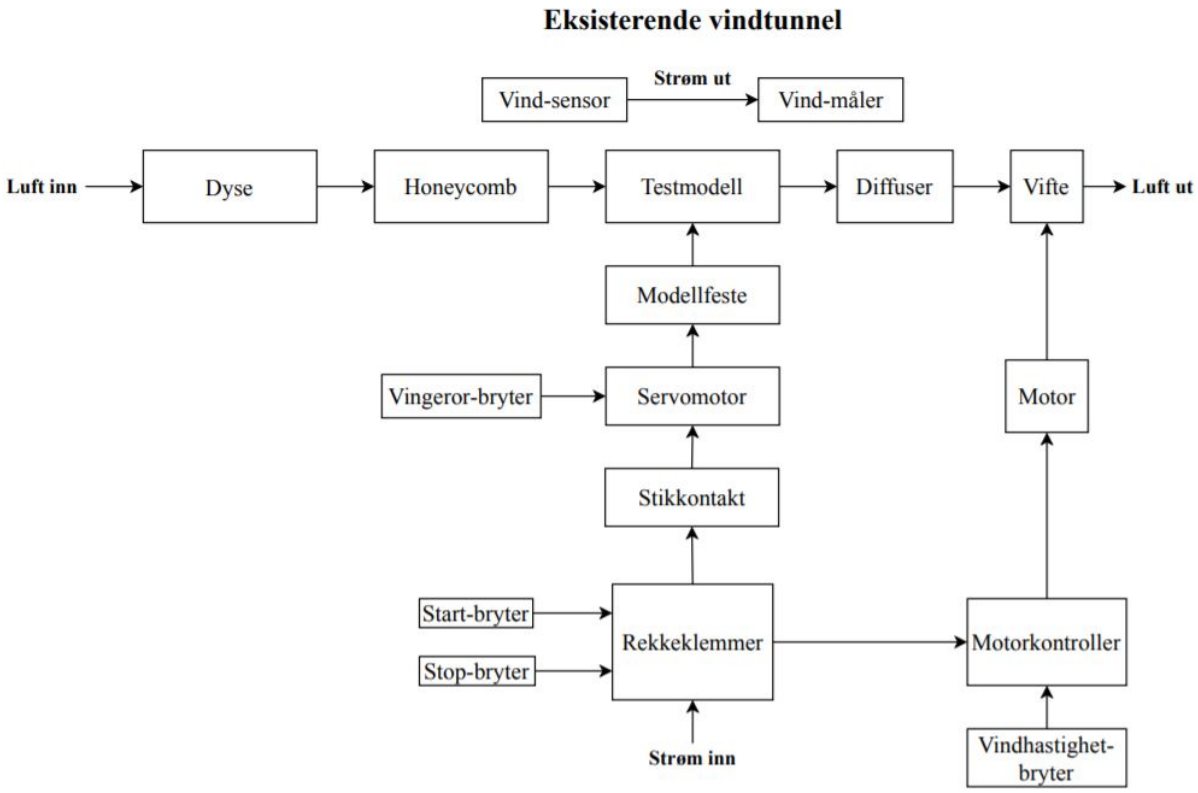
Tabell 4.1 viser nøkkelparameterne som vi er mest interessert i, for å få en oversikt over hvordan systemet yter i nåværende tilstand.

Parameter	**Verdi**
Max vindhastighet	11.7 m/s
Motorhastighet	890 rpm
Volumstrøm	8000 kubikkmeter/time
Reynolds tall	65 000

Tabell 4.1.: Ytelsesparametere for eksisterende vindtunnel.

4.2.1. Systemstruktur

Det overordnede systemet defineres som et elektromekanisk system. Figur 4.7 viser fullstendig overordnet systemstruktur for den eksisterende vindtunnelen og hvordan alle de mekaniske og elektriske komponentene henger sammen.



Figur 4.7.: Overordnet systemstruktur for den eksisterende vindtunnelen.

Produkt	**Spesifikasjoner**
Vifte	ZIEHL-ABEGG-FC063-6DQ.4I.A7 aksial vifte med diecast blader av aluminium
Motor	400 V/50 Hz, 3-fas AC motor D/Y, 1,3 A, 590 W, 60 C, 0.66 cos pi, 85 Pa, 12000 kubikkmeter/time og 890 rpm/min.
Motorkontroller	ZIEHL-ABEGG Dcontrol PKDT5 for spenningsregulert 3-fas motor.
Servokontroller	Graupner Quartz + Servo 764.
Servomotor	TowerPro Micro Servo 9G SG90.
Tidsrelé	Crouzet MCR1 8 A, 0.1s-100h, OFF-delay tidsrelé.
Strømforsyning	6 Watt/1.2A switching mode power supply enhet. Universal input: 100-240v AC 50/60Hz for servokontroller.
Stikkontakt	230 V/50 Hz, dobbel stikkontakt.
Startbryter	Elan EF 03.01.
Stoppbryter	Elan EF 10.1.
Bryter for vindjustering	Analog dimmer.
Bryter for vingeror	Analog dimmer.
Vindsensor	Anemometer.
Vindmåler	Raa Varv Sweden analog måler.

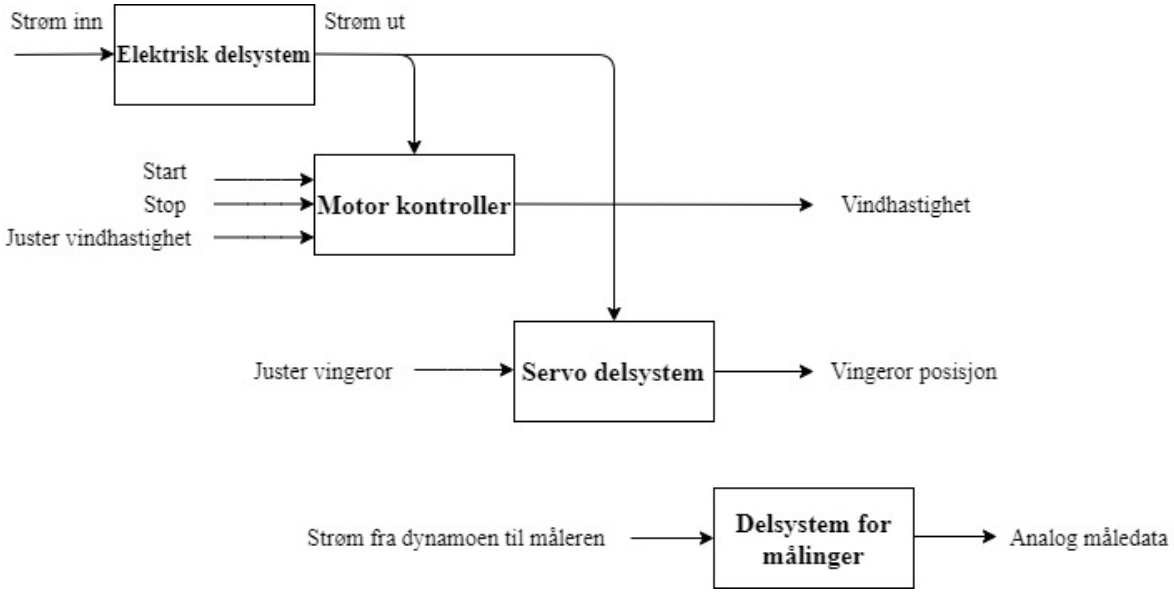
Tabell 4.2.: Produktspesifikasjoner.

4.2.2. Systemfunksjonalitet

Hovedfunksjoner

Funksjonaliteten til eksisterende system deles opp i fem hovedfunksjoner:

- Systemstart
- Systemstopp
- Juster vindhastighet
- Juster vingeror
- Mål vindhastighet



Figur 4.8.: Funksjonaliteten til den eksisterende vindtunnelen.

4.2.3. Elektrisk delsystem

Beskrivelse av det elektriske delsystemet

Det elektriske delsystemet til vindtunnelen sørger for å tilføre rikelig med elektrisk energi som konverteres til mekanisk energi for at det mekaniske delsystemet skal kunne operere.

Systemet får strømforsyning fra 400 Volt vekselspenning på 50 Hz fra vegguttaket fordelt på 3 faser. Tilført strøm distribueres til start og stopp bryterne, analog bryter for vindhastighet, analog bryter for vingeror og den elektriske spenningsregulerte 3 fase AC motoren.

AC motoren har som oppgave å konvertere den elektriske energien til mekanisk energi som får vifta til å rotere. Når delsystemet er tilkoblet til strømforsyningen, så står systemet i ro til man trykker på startknappen. Hvor fort vifta roterer, avhenger av hvor mye spenning som blir tilført til motorkontroller kortet ved hjelp av den analoge bryteren for vindhastighet.

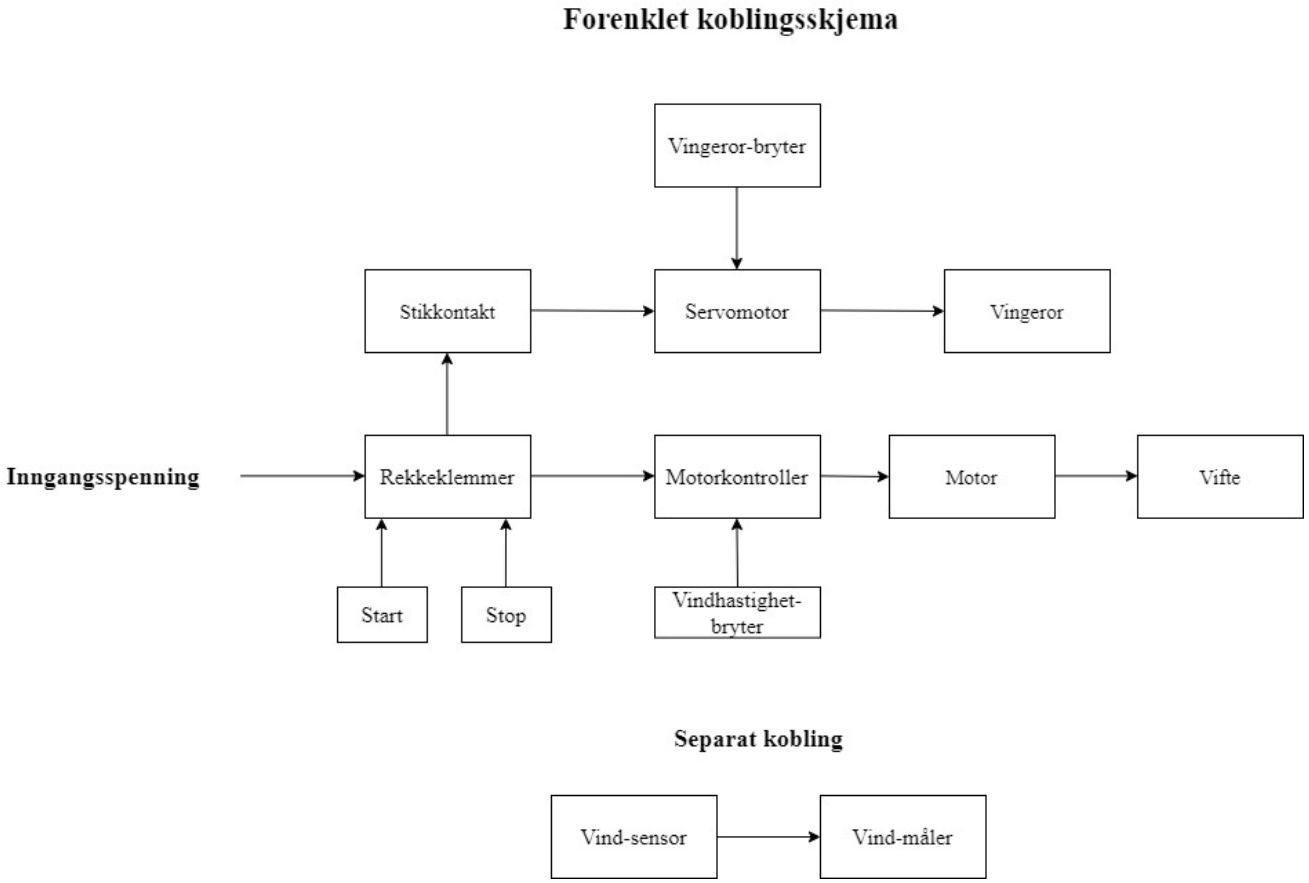
Justering av vingeroret fungerer på samme måte bare at den analoge bryteren for justering av vingeror styrer servokontrolleren som får strømforsyning fra stikkontakten under vindtunnelen. Systemet stopper ikke før man trykker på stoppknappen som bryter kretsen.

Alle de elektriske komponentene er jordet.

Vind-sensoren og vind-måleren er ikke tilkoblet det elektriske delsystemet men står for seg selv som en separat kobling. Vind-sensoreren er et anemometer. Anemometeret vil rotere ved påført vindstrøm. Rotasjonen vil generere en strøm som blir sendt til den analoge vind-måleren. Vindhastigheten som vises på måleren avhenger av mengden strøm den får tilført som følge av vindhastigheten, jo mer tilført strøm jo høyere verdi vil måleren vise.

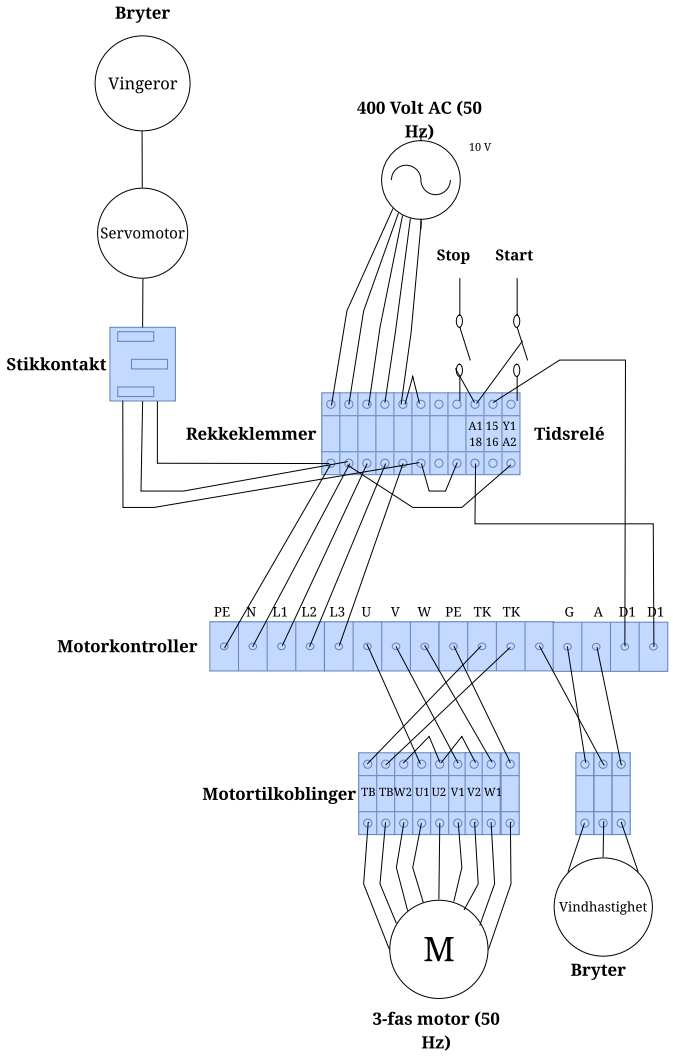
Koblingsskjemaer

Figur 4.9 viser en forenklet versjon av hvordan det elektriske delsystemet er koblet sammen.



Figur 4.9.: Forenklet kretsskjema for det elektriske delsystemet.

Figur 4.10 viser i detalj hvordan det elektriske delsystemet er koblet sammen.



Figur 4.10.: Detaljert kretsskjema for elektrisk delsystem.

Rekkeklemmer

Koblingsboksen med rekkeklemmer er det sentrale koblingspunktet der alle inngangskoblinger fra linjen kobles til på oversiden av rekkeklemmene. Alle lastkoblinger som servokontroller og motorkontroller kobles til på undersiden av rekkeklemmene. Det står også et tidsur inni koblingsboksen som brukes for å gi en forsinkelse (OFF-delay) når man trykker på stoppbryteren. Tabell 4.3 viser en koblingsoversikt for rekkeklemmene og Vedlegg F viser et bilde av de virkelige rekkeklemmene under vindtunnelen.

Linje (overside)	**Klemme nr.**	**Last (underside)**
Jord fra strømforsyning.	1	Jord til stikkontakt + PE port på motorkontroller.
Blå N-leder fra strømforsyning.	2	Blå N-leder til støpsel + motorkontroller og Blå leder til A2 utgangen på tidsreléet.
Grå L1-leder fra strømforsyning.	3	Grå L1-leder til L1 port på motorkontroller kortet.
Sort L2-leder fra strømforsyning.	4	Sort L2-leder til L2 port på motorkontroller kortet.
Brun L3-leder fra strømforsyning + sort jumper ledning til Klemme nr. 6.	5	Brun L3-leder til L3 port på motorkontroller kortet.
Sort jumper leder fra klemme nr. 5.	6	Sort jumper leder til klemme nr. 8 + Brun leder til stikkontakt.
Ingen tilkobling.	7	Ingen tilkobling.
Brun leder til Stopp-knappen.	8	Sort jumper leder fra klemme nr. 6.
2 blå ledere til Start + Stopp knapp fra A1.	A1/18 (tidsrelé)	Brun leder til D1 på motorkontroller kortet fra 18.
Blå leder til D1 på motorkontroller kortet fra 15.	15/16 (tidsrelé)	Ingen kobling på 16.
Brun leder til Start-knappen.	Y1/A2 (tidsrelé)	Blå leder til undersiden av klemme nr. 2.

Tabell 4.3.: Rekkeklemmetabell.

Motorkontroller

Vedlegg G viser motorkontroller kortet. Dette kortet styrer rotasjonshastigheten til den spenningsregulerte 3-fas AC motoren til vifta. På kortet har vi tilkoblinger for linjene fra strømforsyningen (L1, L2, L3, N og PE), tilkoblinger til motor (U, V og W), tilkoblinger til motorbeskyttelse (TK) og tilkoblinger for regulering av vindhastighet (E, GND og A). E porten får tilført variabel spenning fra bryteren for vindhastighet som fungerer som et potensiometer. A porten tilfører potensiometeret 10 Volt inngangsspenning, GND er jord. Et potensiometer gir variabel motstand som øker eller minker spenningen som blir tilført til motoren når brukeren vrir på bryteren. Tabell 4.4 viser en koblingsoversikt for motorkontrollerkortet.

Inngang på kortet	**Koblet til**
PE (jord)	Undersiden av rekkeklemme nr. 1.
N (nøytral)	Undersiden av rekkeklemme nr. 2.
L1	Undersiden av rekkeklemme nr. 3.
L2	Undersiden av rekkeklemme nr. 4.
L3	Undersiden av rekkeklemme nr. 5.
U	U1 på motor (fase L2).
V	V1 på motor (fase L1).
W	W1 på motor (fase L3).
PE (jord)	PE på motor (jord).
TK	TB på motor.
TK	TB på motor.
E	Inngang 2 på bryter for vindhastighet.
GND	Inngang 1 på bryter for vindhastighet.
A	Inngang 3 på bryter for vindhastighet.
D1	Port 15 på tidsrelé-et.
D1	Port 18 på tidsrelé-et.

Tabell 4.4.: Koblinger for motorkontroller kortet.

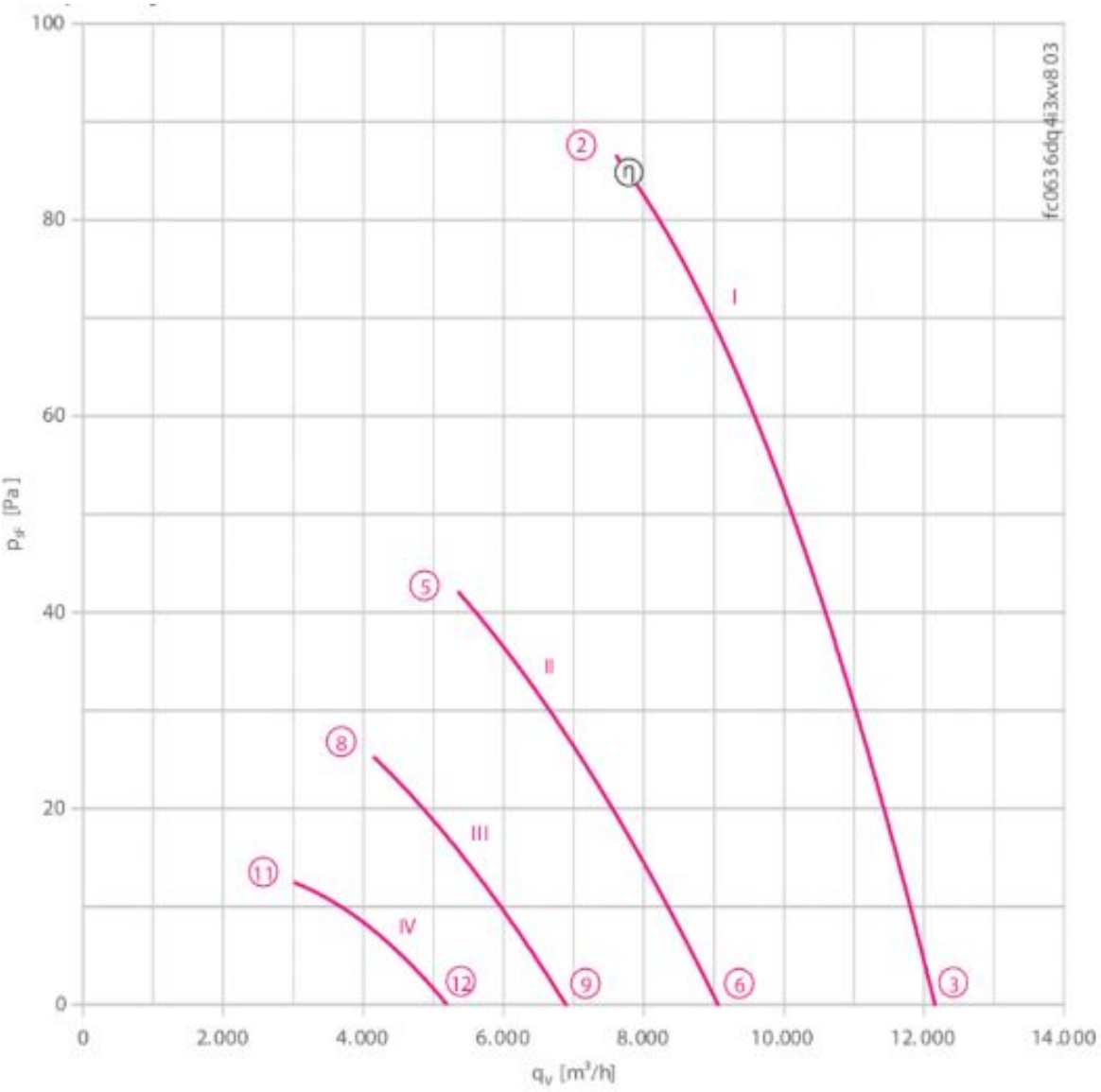
4.2.4. Mekanisk delsystem

Til tross for flere henvendelser til hovedprodusenten av tunnelen har vi ikke klart å få tak i eksisterende dokumentasjon. Vi har derfor dokumentert komponenter på egenhånd så langt det lar seg gjøre. En av hovedkomponentene i systemet er vifta som kan kategoriseres som en aksialvifte dvs. at luftgjennomstrømning er parallell med vifteakselen. Etter forespørsel til produsenten Ziehl-Abegg har det blitt bekreftet at modellen er av typen FC063-6DQ.4I.A7 og vi har mottatt datablad på denne. Inkludert i databladet er det som kalles viftekarakteristikk, som er gjengitt i Figur 4.11. Denne gir konkret informasjon om hvilket volum som kan flyttes per tidsenhet mot endret trykk basert på tilført strøm til motoren.

Utformingen på tunnelen er slik at når man ser på konstante strømningsforhold vil 2 tverrsnitt i tunnelen vinkelrett på strømningen gi lik total volumstrøm igjennom hvert av disse. Det vil si at produktet av hastighet i m/s og areal i m^2 gir m^3/s.

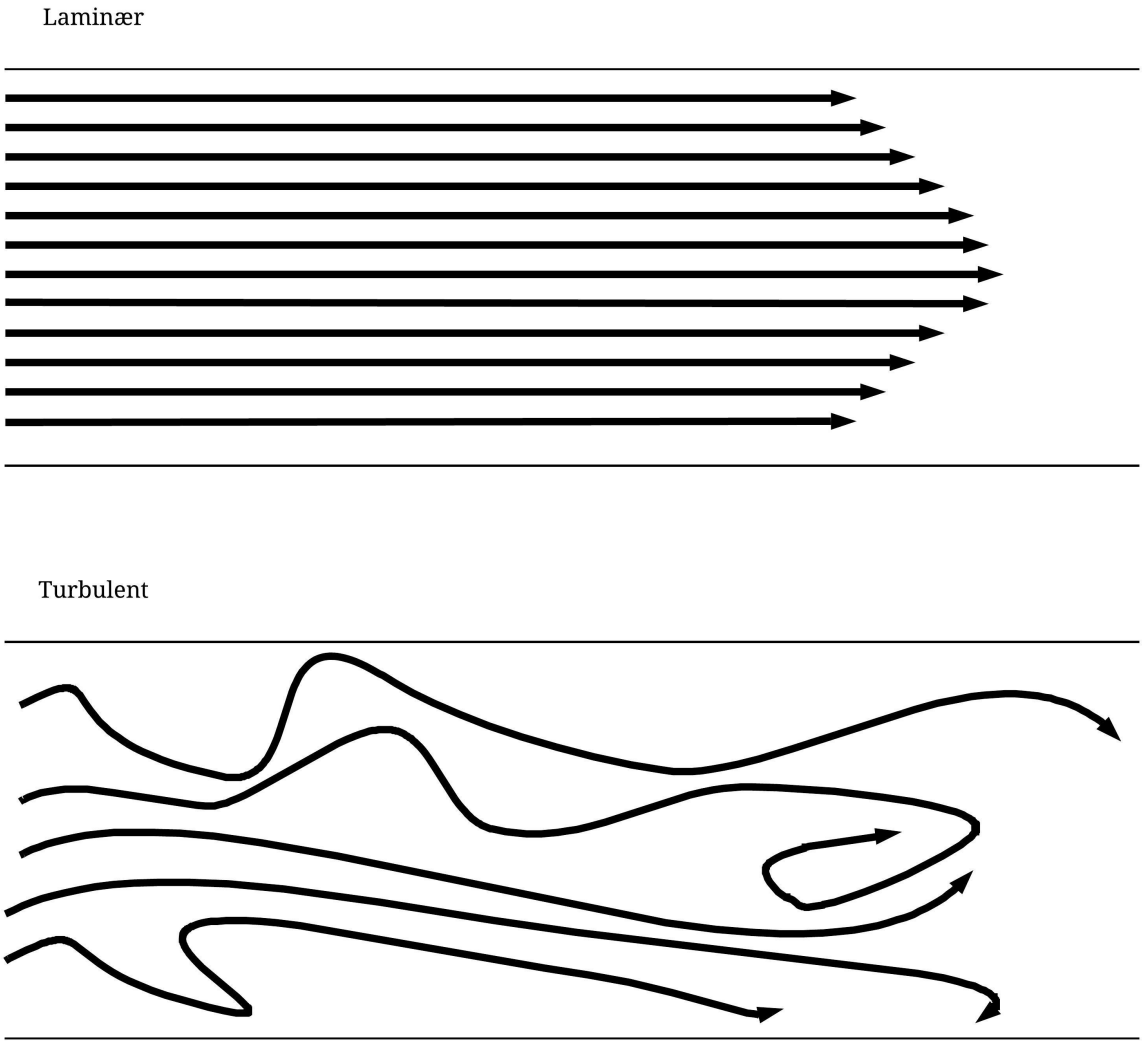
Inngangspartiet er firkantet med en rist foran for å hindre små objekter fra å komme inn og fører luften gjennom et minskende tverrsnitt mot testkammeret. Her trer Bernoullis lov i effekt og den korresponderende vindhastigheten øker etterhvert som tverrsnittet minker. Ca midt i inngangspartiet (dyse/nozzle) er det en strømretter som har en honeycomb-struktur som forsøker å gjøre luftstrømmen som føres gjennom noenlunde laminær, se Figur 4.12.

Idag er strømretteren bare klemt inntil veggen av dysa og med hull på sidene som slipper luft gjennom på sidene som bidrar ytterligere til turbulens. Inni testkammeret er det et uniformt tverrsnitt som ikke endrer seg, og på toppen av kammeret mot diffuseren sitter den analoge vindmåleren som brukes idag. Hvis lufthastigheten er høy nok klarer den å få den gamle vindmåleren til å rotere og gi et utslag på vindmålerinstrumentpanelet på konsollen. Etter testkammeret blir luften trukket gjennom et kort og forsøkt avrundet tverrsnitt som skal fungere som en diffuser som er litt større enn viften. I diffuseren skal luften ha mulighet til å utvide seg og minke hastigheten sin. Rett før viften er det også en rist som skal hindre folk å



Figur 4.11.: Trykk og volumstrøm for vifta.

røre den. Etter dette blir luften trukket gjennom viften og ut i et påfestet åpent rør med rist som skal hindre folk i å ha tilgang til baksiden av viften. Til slutt kommer luften ut i rommet igjen.



Figur 4.12.: Laminær- og turbulent flyt gjennom rør.

Tunnelen er ikke tett nok til å få en god luftflyt gjennom og det er mange steder det ikke er tett nok som bidrar til ekstra turbulens. Den turbulente luftstrømmen er vanskelig å måle i forhold til en laminær strømning. Vindmåleren er unøyaktig og selv om viften kjører jevnt gir den et vekslende utslag. Dette utslaget stemmer da dårlig med målinger som har blitt gjort separat

med anemometer under denne tidlige fasen.

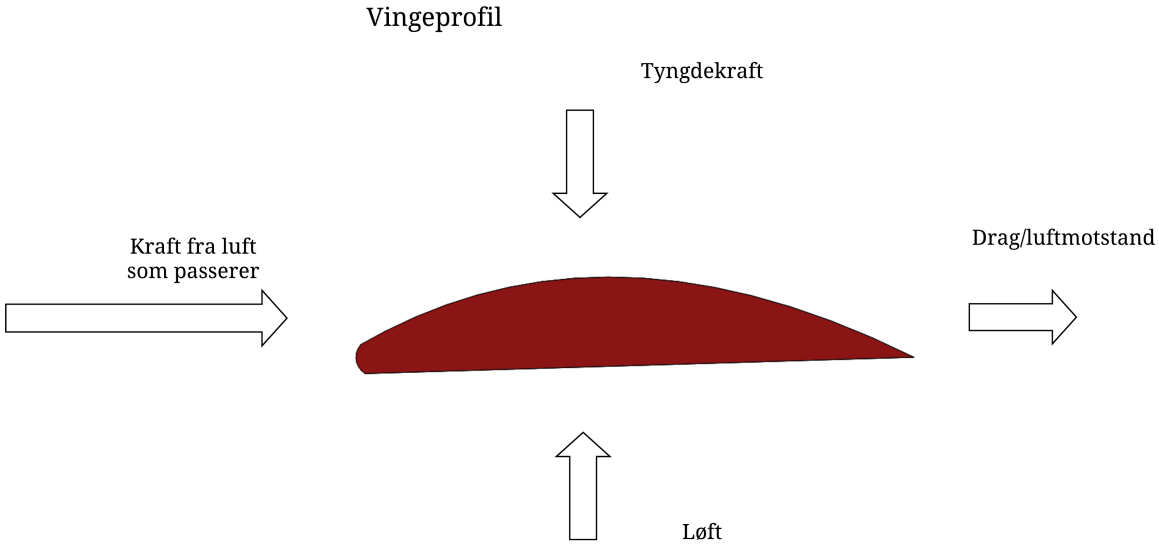
4.3. Testmodeller

Tegn	**betydning**
d_M	dimensjoner til modell.
d_P	dimensjoner prototype.
Reynoldstall	Mål for hvor jevn eller kaotisk ordnet en fluidstrømning er.
ρ	massetetthet (vekt per volum).
C_D	Enhetsløs koeffesient for luftmotstand avhengig av form.
A_M	Tverrsnittsarealet for modellen.
A_P	Tverrsnittsarealet for prototypen.
U	Lufthastighet.
U_M	Lufthastighet for modellen.
U_P	Lufthastighet for prototypen.
F_M	Kraft på modellen.
F_P	Kraft på prototypen.

Tabell 4.5.: Ytelsesparametere for testmodell.

For å undersøke kreftene som noe blir utsatt for bruker man ofte en referansemodell om man ikke har en vindtunnel som er stor nok til å ha plass til det som skal undersøkes. I noen tilfeller kan man også ha en innretning som er liten nok til at den har plass i en vindtunnel og kan testes direkte.

Vindtunnelen er bygget for å utsette modellene eller det som skal testes for en luftstrøm slik at man kan måle kreftene de blir påført av dette. Når modellen blir påført disse kreftene kan man med fornuftig plasserte sensorer måle kreftene modellen blir utsatt for mot vindretningen



Figur 4.13.: Testmodell med krefter.

(drag) og kraften i eventuelt løfting (lift) av modellen. Med tanke på løft kan man også måle kreftene i motsatt retning, noe som skjer når modellen blir presset ned.

Kreftene som måles på modellen er avhengig av utformingen av den. Om modellen er formet som en vingeprofil vil man kunne måle at den har noe luftmotstand og med riktig pitch-vinkel vil man også kunne måle at modellen løftes.

Den modellen som fulgte med vindtunnelen er formet som en vingeprofil av et fly. Denne har blitt brukt som en basis for videre modellering og i simuleringer av luftflyt og krefter. Den ble også brukt som et utgangspunkt for design av innfestingen som trengs for å holde fast modellen og overføre krefter fra vinden og til sensorene.

Mens hvis man har en modell som skal forestille en bil, eller noe annet som man ønsker skal holde seg nede, er det større sjanse for å måle at modellen blir trykket ned. Dette med tanke på at biler gjerne designes for å holde seg på bakken, samt at man også kan måle luftmotstanden som ytes på modellen.

Når man kan måle disse kreftene på modellen kan man også feste forskjellige modeller i testkammeret som blir utsatt for de samme forholdene. Dette gjør at modellene kan sammenlignes med tanke på hvilke krefter modellene utsettes for og hvor store disse er. Dette kan f.eks være en del av en prosess for å finne den optimale vingeformen for et fly.

Det er også viktig å huske når man tester en modell av noe vil kreftene som modellen utsettes for i forhold til originalen eller prototypen ved samme hastighet være annerledes. Dette er på grunn av størrelsesforholdet mellom modellen og den ekte utgaven av prototypen.

F.eks vil en modell som er 5 ganger mindre,

$$\frac{d_M}{dP} = \frac{1}{5} \tag{4.1}$$

der d_M står for modellen og d_P står for prototypen som i vindtunnelen utsettes for, trenge en 5 ganger høyere vindhastighet i vindtunnelen for å simulere tilsvarende vindhastighet på prototypen [10].

I vindtunnelen og utenfor er viskositeten på lufta tilnærmet lik og kan i mange tilfeller neglisjeres og man kan bruke følgende formel hvor Reynoldstall for flyt kan neglisjeres [10].

$$U_M \cdot d_M = U_P \cdot d_P \tag{4.2}$$

På denne måten kan man da ifølge formelen, der U står for hastighet på lufta i vindtunnelen, og d står for dimensjoner eller relativ størrelse da mellom M som står for modell og P som står for prototype. Kan man da se på forholdet mellom dem at man simulerer en vindhastighet skalert til prototypen som er 5 ganger lavere enn den som påføres på modellen.

$$U_M = U_P \cdot \frac{d_P}{d_M} \cdot 5U_P \tag{4.3}$$

U_M står for fart som testmodellen utsettes for, U_P står for farten prototypen utsettes for.

Motstandskraften på den mindre modellen som blir påført vind fra tunnelen i forhold til prototypen den er skalert fra kan finnes med ligningen:

$$F_M = \frac{1}{2} \cdot \rho \cdot c_D \cdot U_M^2 \cdot A_M = \frac{1}{2} \cdot \rho \cdot c_D \cdot U_P^2 \cdot \frac{d_P}{d_M}^2 \cdot A_M \tag{4.4}$$

Her står C_D for en enhetsløs koeffesient for drag som må finnes matematisk eller eksperimentelt , A_M er tverrsnittsarealet for modellen, ρ står for massetettheten (masse per volum) på lufta. Tverrsnittsarealet for modell og prototype skal være likeformet per forholdet mellom størrelsene.

$$\frac{A_M}{d_M^M} = \frac{A_P}{d_P^P} \tag{4.5}$$

Med dette kan man omskrive ligningen for motstandskraften på modellen slik

$$F_M = \frac{1}{2} \cdot \rho \cdot c_D \cdot U_P^2 \cdot A_P = F_P \tag{4.6}$$

Dette betyr da at kraften som modellen utsettes for i vindtunnelen er samme kraft som prototypen utsettes for. Men man må fortsatt da ta i betraktning hva slags hastighet modellen blir utsatt for og huske at prototypen som modellen er modellert etter her blir utsatt for de samme kreftene på seg ved 5 ganger lavere hastighet under reelle forhold.

5. Testing

5.1. Hensikten med testing

Etter at systemkrav er opprettet må man definere konkrete tester som viser om kravene har blitt oppfylt. Disse er det lettere å kunne gi klare svar på om de kvantifiseres, og man kan få definitive svar på om de er oppfylt.

5.2. Testmetoder

De fire hovedmetodene for testing har blitt delt inn i simulering, måling, review og funksjonstest. Disse har fått henholdsvis betegnelsene (S), (M), (R) og (F).

5.2.1. Simulering

Simulering innebærer bruk av dataprogrammer for å tilnærme seg forhold som vindtunnelen kan utsettes for som er både mer og mindre sannsynlige uten å fysisk teste systemet. Dette er ikke like bra som å gjennomføre testen fysisk eller i felten, men kan også gjentas igjen flere ganger med forskjellige variabler. Simulering kan brukes til å verifisere mot krav, men kan ikke validere opp mot virkeligheten. Simulering kan også brukes til destruktiv testing som i virkeligheten bare kunne blitt gjennomført en gang på vindtunnelen. Et eksempel på dette er i

bilindustrien der man kjører mange simuleringer av bilkollisjoner, men til sist må man da validere bilen sin oppførsel med å gjennomføre en fysisk kollisjonstest. Sannsynligvis blir det bare bygget en prototype og den kan ikke risikeres i tester som kan skade eller ødelegge den unødig. Simuleringer kan gi estimater, og kan brukes delvis til verifisering.

5.2.2. Måling

Måling som testmetode involverer å bruke et måleinstrument til å måle verdier som f.eks hastighet eller lengde. Dette kan gjennomføres med enkle og mer avanserte måleinstrumenter. For vindtunnelen er et eksempel på en slik test å måle hvor lang vindtunnelen er og dette kan måles med en tommestokk. Med tanke på vindhastighet kan dette testes med et håndholdt anemometer som måler dette.

5.2.3. Review

Review er en enkel form for testing som innebærer å observere om noe er tilstede eller ikke. Dette kan være i form av at man f.eks kan lese en avmålt verdi på målingene til vindtunnelen, eller se at man har en brukermanual. Review kan også innebære å sjekke datablad man har komponenter for stemmer.

5.2.4. Funksjonstest

Den siste formen for testing er funksjonstest. Her testes det om systemet gjør eller oppfyller den ønskede funksjonen. F.eks kan man skru vifta av eller på, og om man kan regulere vindhastigheten. Det er også interessant hvor nøyaktig verdiene som blir avlest er.

For å gjennomføre enkelte av testene brukes det også en kombinasjon av flere typer testing. F.eks kombineres det funksjons- og review-test, for å observere om det er lite eller ingen turbulens i testkammeret når vifta er på og man tilfører røyk.

5.3. Kriterier for en vellykket test

For at testen skal være vellykket må testen kunne bekrefte at kravene som er satt til systemet har blitt oppnådd. Dette kan være i form av å kunne regulere vindhastigheten som i Tabell 5.1 på neste side eller at luftstrømmen går laminært nok gjennom testkammeret.

Vi har valgt å sette det opp slik at det man knytter sammen er den individuelle kravID-en med den korresponderende test ID-en. På den måten kan man se at det henger sammen. Dette kan observeres i tabell 5.1 på neste side.

Test ID	**T.VT 04**
R.KF.01	Festene er satt på løst under konstruksjonen
Test utført av	Håvard
Godkjent av	Kristoffer
Dato	1.1.2020
Testmetode	F/M
Hva skal testes?	Brukerinnstilling av lufthastighet.
Hva er krav for godkjenning?	Bruker kan regulere vindhastigheten i testkammeret.
Test	Forsøke å regulere vindhastighet mens vindtunnelen blåser og verifisere med anemometer.
Testnummer	1
Status	Godkjent

Tabell 5.1.: Eksempel på gjennomført test.

6. Risikoanalyse

6.1. Intro til risiko

Ved bruken av risikoanalyse kan vi få et overblikk over sannsynligheten av mulige hendelser samt konsekvensene av dem. På denne måten kan vi forebygge mot risikoelementer og vektlegge hvor vi må legge fokus. Slik det er satt opp, er dette delt inn i seks deler.

1. Beskrivelse
    - Beskriver hva som kan gå galt.
2. Sannsynlighet
    - Hva er sannsynligheten for at hendelsen inntreffer.
3. Alvorlighetsgrad
    - Hvor stor grad av alvorlighet/påvirkning har det som inntreffer.
4. Risiko
    - Hvor stor risiko hendelsen har som et produkt av sannsynlighet og konsekvens. Dette vil gi en verdi som indikerer hvor stort fokus den bør ha på å forebygges.
5. Konsekvens
    - Beskrivelse av hva som skjer om hendelsen inntreffer.

6. Forholdsregel

    - Hvordan kan vi forebygge mot denne hendelsen.

6.1.1. Risikooversikt

For at vi skal kunne identifisere hvilke risikoelementer som trenger umiddelbar oppmerksomhet vektlegges sannsynlighet og alvorlighetsgrad fra 1 til 5 hvor:

Sannsynlighet:

1 = Liten sannsynlighet

5 = Stor sannsynlighet

Alvorlighetsgrad:

1 = Lite alvorlig

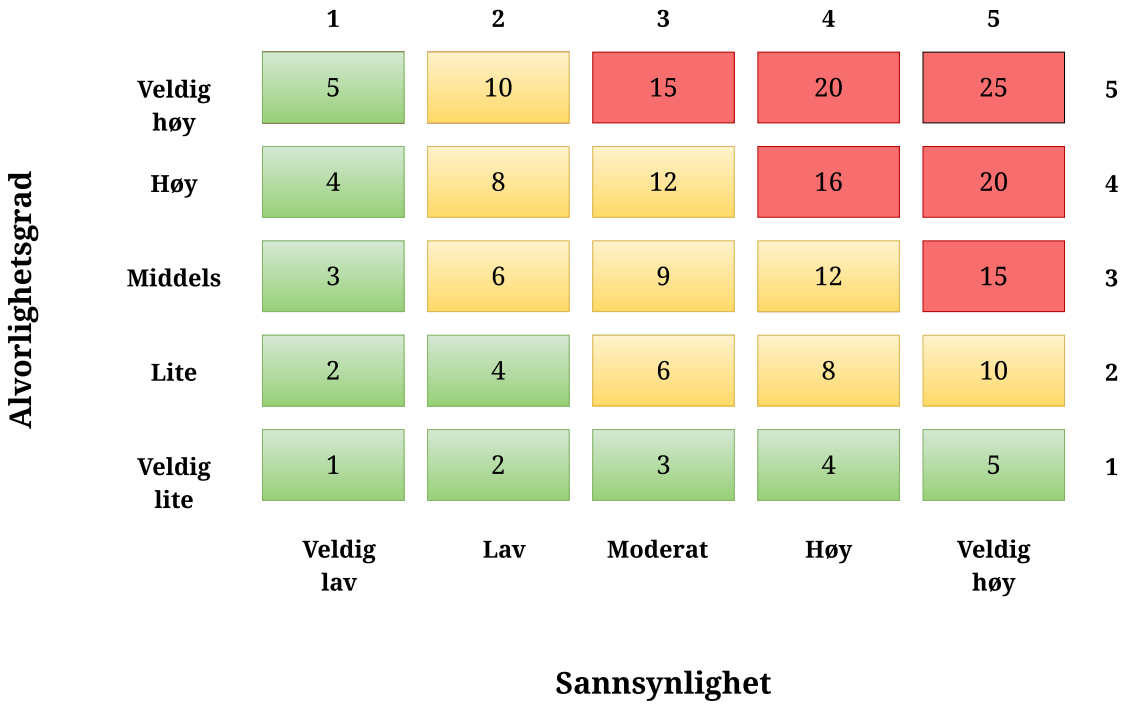
5 = Veldig alvorlig

Etter vi har veid dem opp så ganges de sammen for å vise overordnet risiko:

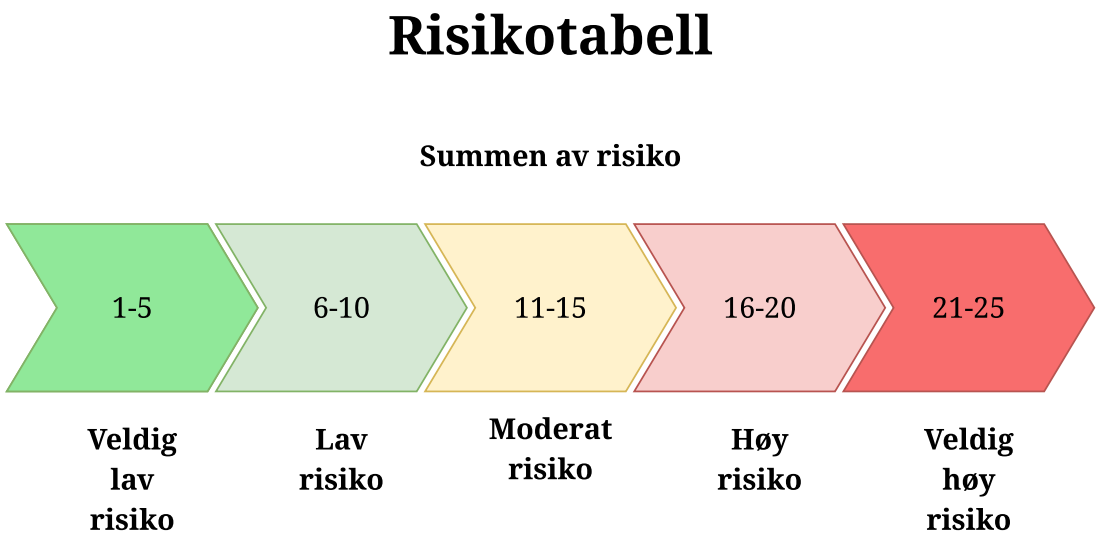
Sannsynlighet x Alvorlighetsgrad = Risiko [24]

Tabellen i figur 6.1 viser til hvordan risikoen blir regnet ut og gir tallet en fargekode slik at det er lett å kunne se nivået risikoen ligger på.

Etter at vi har vektlagt sannsynlighet og alvorlighetsgrad så blir de slått sammen til å danne risiko. Konsekvensen blir beregnet og vi kan da få tall mellom 1 og 25 (Figur 6.2).



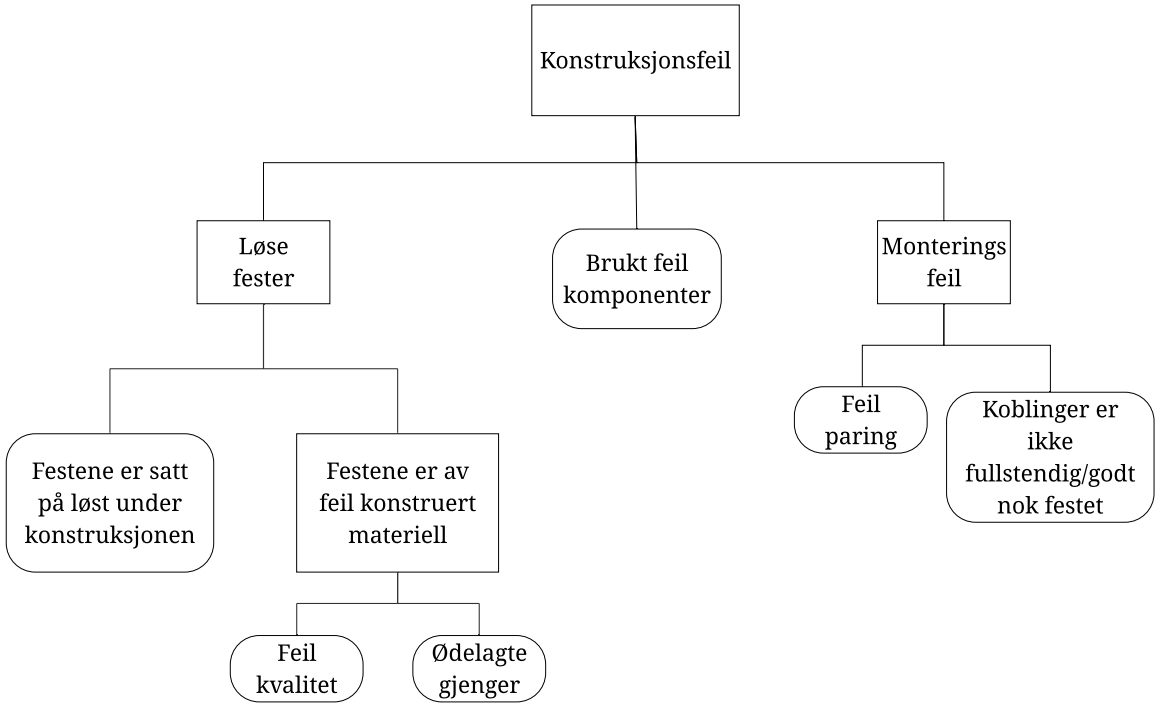
Figur 6.1.: Konsekvensmatrise.



Figur 6.2.: Risikotabell.

6.1.2. Risikoanalyse (FMEA)

Når vi skal begynne med å finne risker til FMEA (Failure Mode and Effects Analysis) [31], starter vi med å tegne et **fault tree diagram**. Dette hjelper oss med å kartlegge hva slags elementer som hører til de forskjellige risikoene. Vi identifiserer typen risiko vi ønsker å kartlegge, setter inn de forskjellige årsakene som kan lede til hendelsen, og hvor disse igjen kan ha sitt opphav.



Figur 6.3.: Utdrag fra Fault tree diagrammene.

Etter at vi har kartlagt hvilke hendelser vi antar kommer til å oppstå, setter vi det inn i en risikotabell hvor vi vektlegger sannsynlighet og alvorlighetsgraden som gir oss risikofaktoren.

ID-en har blitt veldig enkelt satt opp slik at det skal være lett å finne ut hvilken kategori hver enkel risiko befinner seg i. Vi har satt opp følgende for ID-systemet: **Risiko.Kategori.0xxx** (se Tabell 6.1).

Eksempel blir ett krav i kategorien "Feil på sensormålinger", på ID **R.FPSM.01**.

Kategoriene og ID-ene er delt opp slikt:

KF – Konstruksjonsfeil.

FPSM – Feil på sensormålinger.

FIL – Feil i luftstrømingen.

6.1.3. Prosjektanalyse

Ettersom vi da har satt opp en FMEA, så har vi også laget en analyse for prosjektet og gruppen (se Tabell 6.2). Prosjektanalyse tar for seg forskjellige risikoer/hendelser som kan skje under Bachelor-prosjektet. Ved å lage en prosjektanalyse, får vi en analyse som tar for seg alt om prosessen med prosjektarbeidet, som fortsetter til vi er ferdige med prosjektet. Vi har brukt samme system for å sette ID som i FMEA-en.

Kategoriene og ID-ene er delt opp slikt:

Gr – Gruppe

Ra – Rapporten

Pr – Prosjekt

Br – Bruker

Be – Bedrift

Øk – Økonomi

Risiko ID	Beskrivelse	Sannsyn-lighet	Alvorlig-hetsgrad	Risiko	Konsekvens	Forholdsregel
R.KF.01	Festene er satt på løst under konstruksjonen.	2	4	8	Høy vibrasjon samt farlig for bruker samt økt slitasje.	Overgå alle fester og sikre fester så godt som mulig.
R.KF.02	Brukt feil materiale til oppga-ven	1	3	3	Mulig for at deler blir ødelagt over tid raskere	Kvalitetssjekke delene.
R.KF.03	Sammensetting av systemet blir gjort feil.	3	4	12	Deler kan løsne og skade bru-keren	Overgå montering.
R.KF.04	Feil paring mellom skru-er/verktøy.	1	2	2	Parrings feil kan ødelegger gjenger å kan bli vanskelig og få ut igjen	Om det er stor motstand i å skru, så stopp.
R.KF.05	Bruk av feil komponenter.	3	4	12	Det blir et ufullstendig system eller raskere slitasje på deler.	Overgå komponentene vi har planlagt å bruke.
R.FPSM.01	Sensor får ikke strøm.	2	5	10	Sensoren får ikke strøm slik at de ikke får gjort målinger.	Se til at alle elektriske ledninger er korrekt og montering er riktig.

Tabell 6.1.: Utdrag fra risikoanalyse for gruppen.

Risiko ID	Beskrivelse	Sannsynlighet	Alvorlighetsgrad	Risiko	Konsekvens	Forholdsregel
R.Gr. 01	Gruppen splitter opp.	1	5	5	Vi får ikke fulført bachelorgraden.	Holde godt sosialt samhold.
R.Gr. 02	Gruppen blir uvenner, fullfører ikke arbeid.	3	2	6	Mindre effektivt arbeid	Holde godt sosialt samhold.
R.Gr. 03	Gruppemedlem forlater prosjektet.	2	4	8	Mindre arbeidskraft I gruppen.	Umulig å forbygge mot.
R.Gr. 04	Gruppemedlem gjør ikke arbeid.	2	3	6	Resten av gruppen blir oppgitt på personen og blir dårlig samhold.	Passe på at alle har noe å gjøre og sette tidsfrister.
R.Gr. 05	Gruppemedlem blir syk.	5	3	15	Oppgaver som den personen har blir ikke gjort.	Hjelpe hverandre om med forskjellige oppgaver om man er syk.
R.Gr. 06	Gruppemedlem blir sykemeldt I lang tid.	3	4	12	Mindre arbeidskraft I gruppen.	Holde det renslig og hygienisk på skole/hjemme.

Tabell 6.2.: Utdrag fra prosjekt risikoanalyse.

Del II.

Planlegging og design

7. Konseptutvikling

7.1. Intro til konsepter

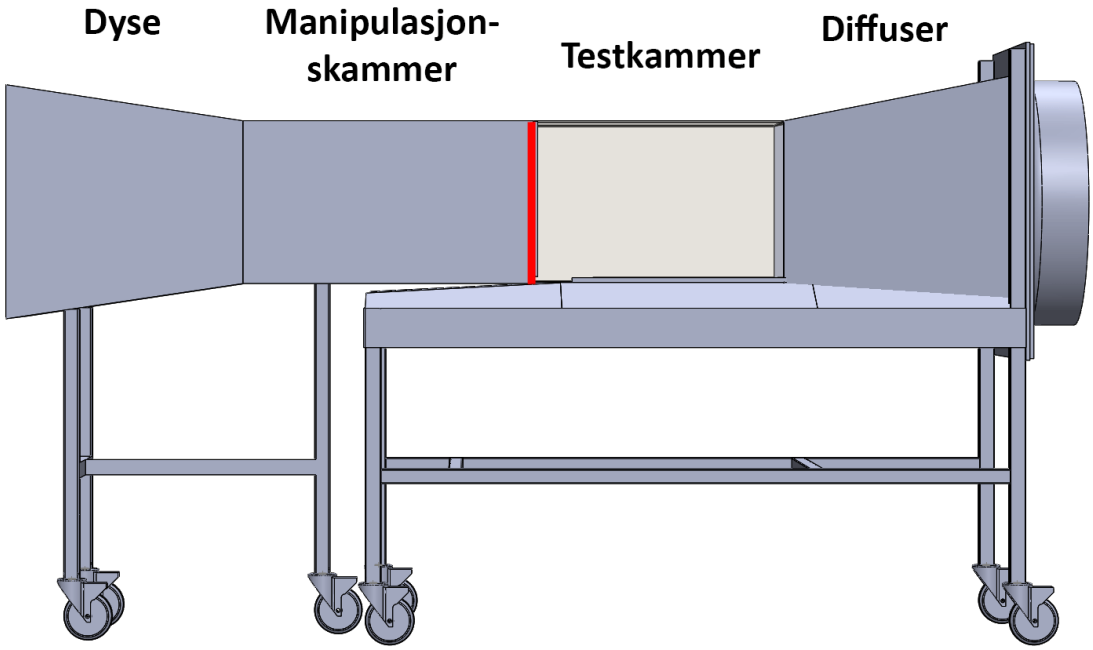
Vi har kommet opp med tre forskjellige konsepter som oppdragsgiveren kan vurdere. I denne seksjonen vil vi gå igjennom hva disse tre konseptene har av funksjoner og hvordan de er designet.

7.2. Konsept 1

Konsept 1 kan karakteriseres som det konseptet med lavest kostnad og færrest modifikasjoner for å tilfredstille krav. På grunn av dette er det også antageligvis det raskeste å gjennomføre men vil tilby lavere ytelse, presisjon og minimal funksjonalitet.

Etter å lest mange kilder for fremgangsmåter på hvordan man bør konstruere en vindtunnel har vi enda ikke sett eksempel på at strømretteren er plassert slik den er i vårt eksisterende system. Man har vanligvis et eget kammer for strømretter/nettinger med plassering foran dysen, hvor kammerets tverrsnittsareal er likt dysens inngang. Dette ville i vårt tilfelle krevd innkjøp av en langt større strømretter og netting.

I konsept 1 er tanken derfor heller å plassere disse komponentene i et mindre manipuleringskammer (manipulerer luft) mellom dyse og testkammer. Manipuleringskammerets lengde er basert på antall komponenter som blir brukt her da det anbefales en avstand på 250 mm mellom disse. I vårt tilfelle ønsker vi å gå for strømretter samt to nettinger som gir en total lengde på 750



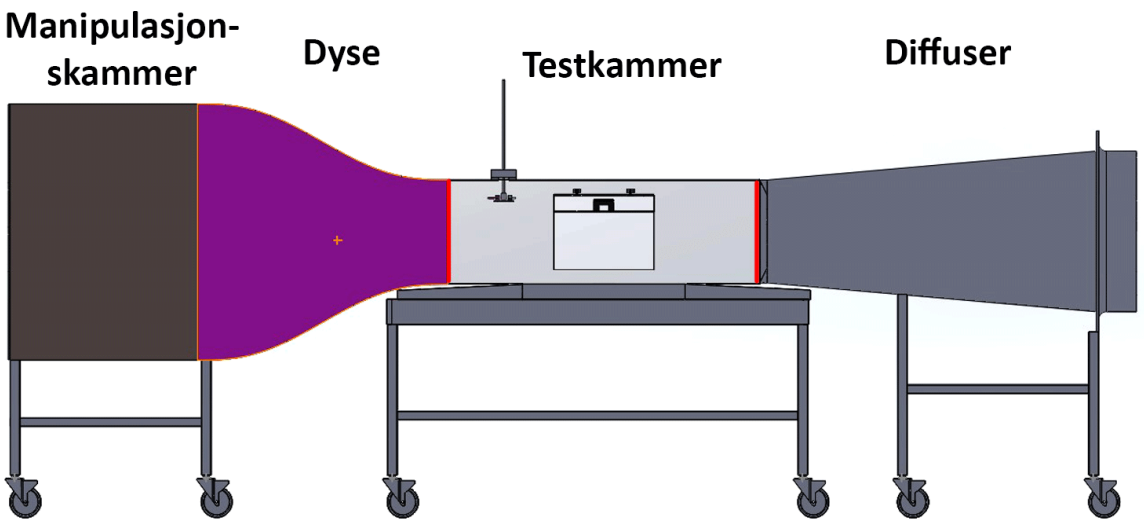
Figur 7.1.: DAK-modell av konsept 1 og grensesnitt mellom modulene markert i rødt.

mm. Denne økningen i lengde vil gjøre at systemet ikke lenger tilfredsstiller krav K.VT 09 om systemets dimensjoner og vi må derfor plassere dyse og manipuleringskammer som en separat modul med egne hjul. Eksisterende dyse brukes på nytt.

I henhold til krav K.DB 01 og K.DB 02, må vindtunnelen ha en innebygd datamaskin som kan vise måledata til brukeren og overføre dataene til en ekstern PC. Denne ble beskrevet i Avsnitt 10.1.

I konsept 1 er det minimalt med funksjonalitet. Kontroll av lufthastighet, for eksempel, utføres manuelt av brukeren ved å justere vifteturtallet, og avlesningen vises på en LCD-skjerm. Alle de fysiske verdiene leses heller ikke nødvendigvis av datamaskinen, hvor det legges prioritet på selve vindmålingen. Se Figur 7.5.

7.3. Konsept 2

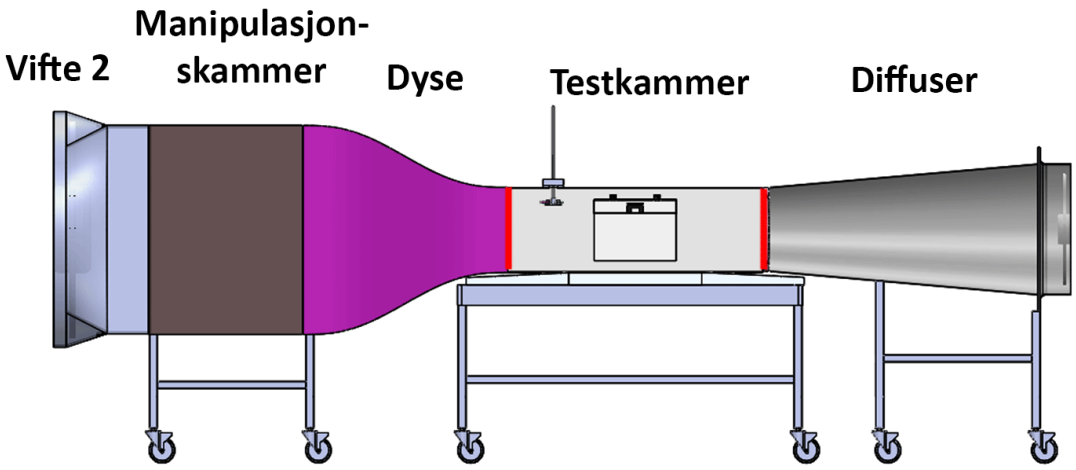


Figur 7.2.: DAK-modell av konsept 2 og grensesnitt mellom modulene markert i rødt.

I konsept 2 produseres det en ny dyse med både optimal profil og bedre kontraksjonsratio. I tillegg vil manipuleringskammeret nå få plassering foran dysen og ha et større tverrsnitt. Testkammeret blir forlenget og får derfor forbedret forhold mellom størrelse på innløp og lengde. En lengre diffuser er også inkludert. Disse endringene gir mer uniform luftstrøm samtidig som at man unngår uønskede fenomener (separasjon av strømning, økning i grensesjikt tykkelse etc). Konsept 2 vil tilby økt mobilitet på sensorer og målere.

I konsept 2 utvides funksjonaliteten til IDM-en og brukergrensesnittet. Skjermen blir nå en touchskjerm, slik at brukeren kan kontrollere IDM-en på en mer sentral måte. IDM-en kan nå styre viftekontrolleren ved hjelp av ett lukket sløyfe reguleringssystem, slik at brukeren kan sette ønsket vindhastighet i testkammeret. Dette grensesnittet er beskrevet i Avsnitt 10.1.3. Det vil også være mulig å kunne sette pitch-vinkelen igjennom brukergrensesnittet ved bruken av en step motor som endrer stillingen slik at det ikke er nødvending å gjøres manuelt. Det skal også kunne være mulig å overføre data over Wi-Fi, hvor i det tidligere konseptet det bare var planlagt å gå med kabel.

7.4. Konsept 3

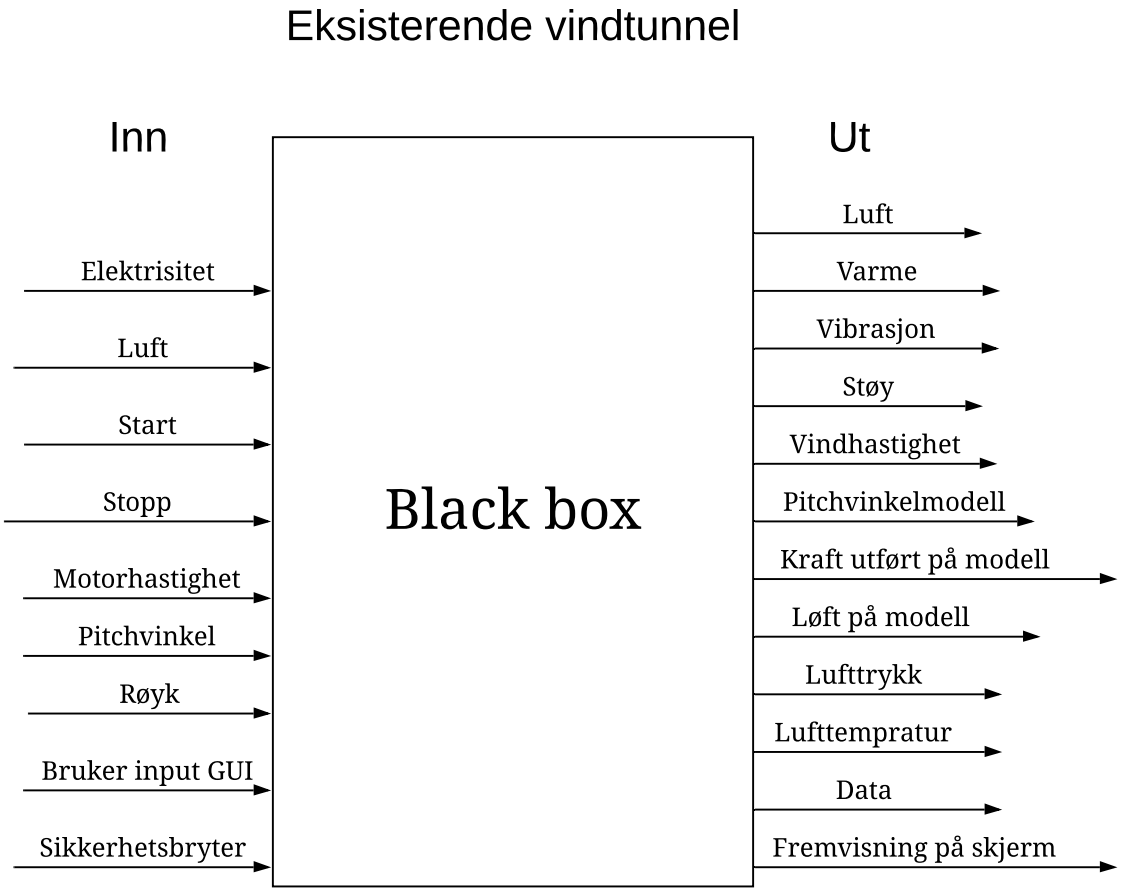


Figur 7.3.: DAK-modell av konsept 3 og grensesnitt mellom modulene markert i rødt.

Konsept 3 er helt identisk til konsept 2 bortsett fra to forskjeller. Den første er at vi kommer til å bruke et "Hot Wire Anemometer" til å måle lufthastigheten. Avhengig av modell vil denne kunne måle hastigheten både aksielt og lateralt (vertikalt/sideveis). Den andre forskjellen blir den største, og det er at vi kommer til å ha en vifte foran og en vifte bak på vindtunnelen som vil gjøre det mulig å regulere trykk i vindtunnelen. Men med en ekstra vifte på tunnelen vil dette øke systemets kostnad, strømforbruk samt tidsbruk på modifikasjoner til vindtunnelen for å støtte den andre viften. Det vil også kreve et motordriver kort til for å kunne styre viften og gi strøm til den, som kobles på det eksisterende elektriske delsystemet. Se Figur 7.5.

7.5. Morfologisk diagram

Et morfologisk diagram lager vi for å få en oversikt på forskjellige måter vi kan generere konsepter på. Et morfologisk diagram hjelper oss med å visualisere måtene man kan skape et



Figur 7.4.: Input/Output for konsept 2 og 3.

Konsept 1			
ID	**Prioritet**	**Funksjoner**	**Løsning**
K.M 01	A	**Måle lufthastighet**	
	B	**Lufthastighetsmåler mobilitet**	Stasjonær
		Metode for posisjonering av lufthastighetsmåler	N/A
		Brukergrensesnitt for mekanisk endring av lufthastighetsmåling	N/A
K.VT 08	A	**Transportering**	
	A	**Start/Stopp**	
K.VT 04	A	**Styre lufthastighet**	
K.VT 04	A	**Justering av lufthastighet (av bruker)**	

Konsept 2			
ID	**Prioritet**	**Funksjoner**	**Løsning**
K.M 01	A	**Måle lufthastighet**	
	B	**Lufthastighetsmåler mobilitet**	Flyttbar
		Metode for posisjonering av lufthastighetsmåler	Mekanisk
		Brukergrensesnitt for mekanisk endring av lufthastighetsmåling	
K.VT 08	A	**Transportering**	
	A	**Start/Stopp**	
K.VT 04	A	**Styre lufthastighet**	Kontrollsløyfe (feedback controller)
K.VT 04	A	**Justering av lufthastighet (av bruker)**	

Konsept 3			
ID	**Prioritet**	**Funksjoner**	**Løsning**
K.M 01	A	**Måle lufthastighet**	
	B	**Lufthastighetsmåler mobilitet**	Flyttbar
		Metode for posisjonering av lufthastighetsmåler	Mekanisk
		Brukergrensesnitt for mekanisk endring av lufthastighetsmåling	
K.VT 08	A	**Transportering**	
	A	**Start/Stopp**	
K.VT 04	A	**Styre lufthastighet**	Kontrollsløyfe (feedback controller)
K.VT 04	A	**Justering av lufthastighet (av bruker)**	

Figur 7.5.: Konsepter fra Morfologiske.

konsept, det vil også hjelpe oss med å fastsette hvilke type funksjoner vi skal ha på vindtunnelen, samt de forskjellige type løsninger vi kan komme opp med til funksjonene. Funksjonen kan komme fra krav eller funksjoner som trengs å være med utenom krav, som en start/stopp bryter.

Måten vi setter opp et morfologisk diagram er da slikt:

1. List opp funksjoner for systemet.
2. List opp løsninger til funksjonene.
3. Vektlegg viktigheten for funksjonene.

Vi starter med å få et overordnet bilde av hvordan disse funksjonene fungerer ved hjelp av sekvensdiagram. Så vi begynner med noe som kalles for "Operational" som blir å kartlegge funksjoner med sekvensdiagrammer slik at vi forstår funksjonen (se Figur 7.10).

- Graf animasjon Figur 7.6.
- Endring av pitch Figur 7.7.
- Styre lufthastighet Figur 7.8.
- Start og stopp session Figur 7.9.
- Eksportering av filer Figur 7.10.
- Eksportering tids valg og oppdatering Figur 7.11.

Deretter når vi har listet opp alle funksjoner og løsninger, kan vi begynne å konstruere konsepter utifra de forskjellige løsningene til funksjonene (se Figur 7.13). For at vi skal gjøre det enkelt og sporbart, har vi valgt å gi hvert konsept sin egen figur og plass i valget av løsninger. Så konsept 1 har en gul firkant og kan alltid bli funnet oppe til venstre i løsningen i tabbelen,

konsept 2 har en blå trekant og kan bli funnet øverst til høyre, mens konsept 3 er en rød sirkel og ligger ved bunnen til høyre, se Figur 7.12.

Når vi har fullført det morfologiske diagrammet, så kan vi begynne å planlegge hva de forskjellige konseptene våres skal være. Vi kom frem til at vi ønsker å vise oppdragsgiver hva slags forskjellig funksjonalitet han kan få ved konseptene vi kommer med, samt hva de koster. Derfor etter at vi satt opp konseptene, så lagde vi et budsjett på hva prisen på de forskjellige konseptene ville blitt, og en Pugh-matrise der hvor vi bruker ett konsept (konsept 2) som et grunnlag, og veier opp de andre konseptene opp mot den for å se om de er bedre eller dårligere.

7.6. Pugh-matrise

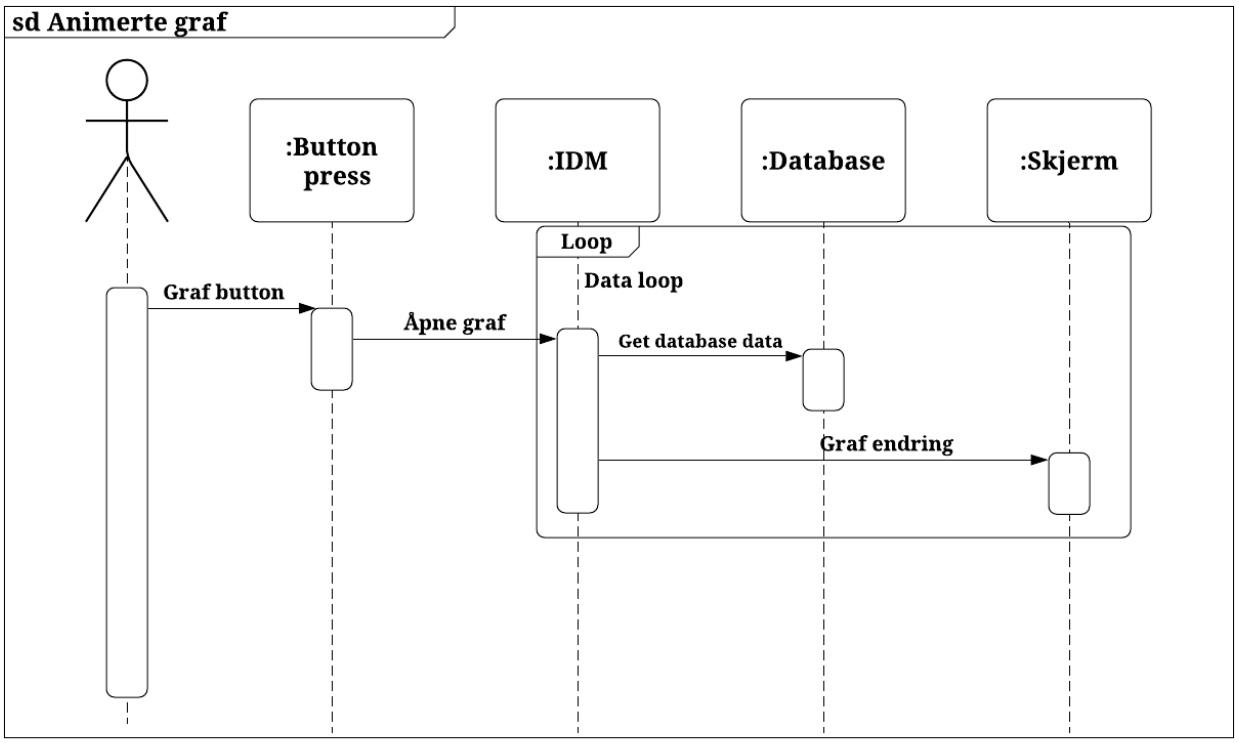
For å velge hvilket konsept å føre videre, har vi satt kandidatene opp i en Pugh-matrise (Tabell 7.1). Vi har valgt å bruke konsept 2 som grunnlinje og sammenligner konsept 1 og konsept 3 i forhold til denne. Dersom et av kriteriene løses bedre eller dårligere enn grunnlinja, markeres det med pluss eller minus, eller null hvis det ikke er noen forskjell. Poengsummen brukes til å rangere konseptene mot hverandre.

Kriteriene vi har valgt er følgende:

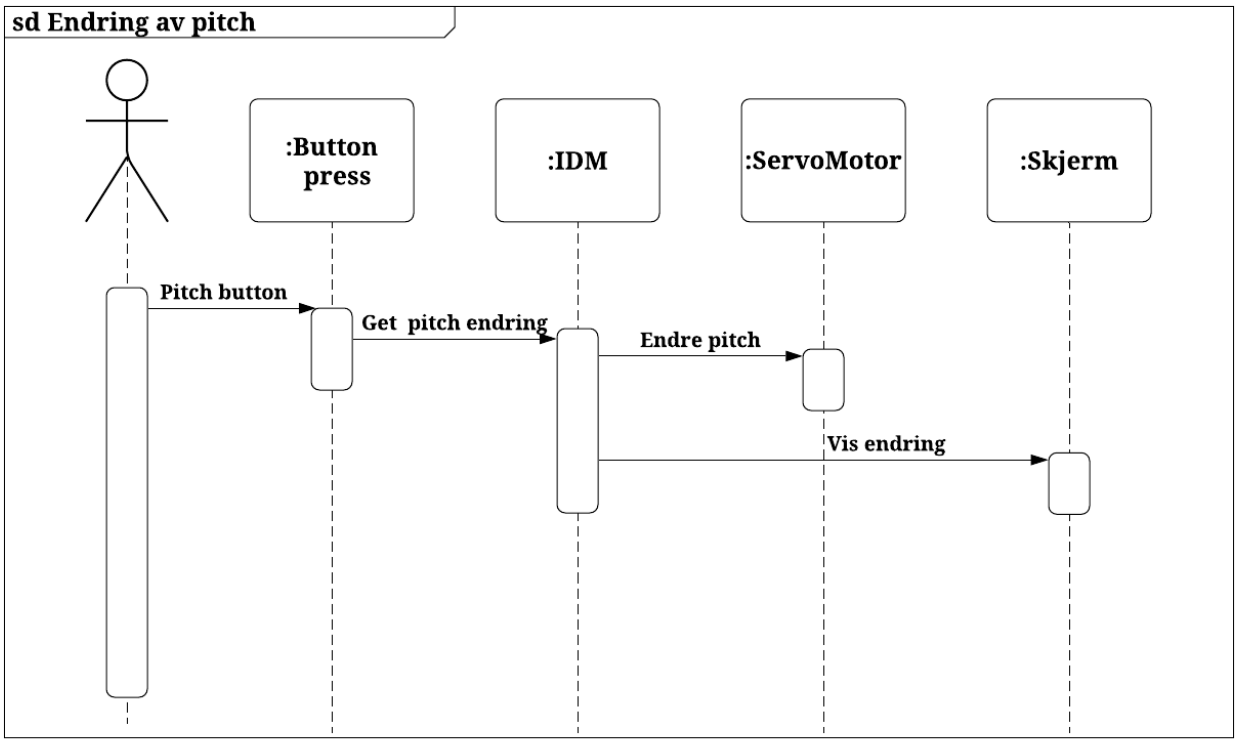
Kostnad – Forventet pris på komponenter som må kjøpes.

Konstruksjonstid – Hvor fort kan systemet ferdigstilles. Inklusivt hvor lang tid det vil ta å bestille eller maskinere deler.

Brukersikkerhet – Hvor godt er systemet sikret for at skader skal forebygges.



Figur 7.6.: Graf animasjon.



Figur 7.7.: Endring av pitch.

Styringspresisjon – Hvor presist kan systemet blir styrt, for eksempel styring av lufthastighet.

Målepresisjon – Hvor presise er målingene som kan gjøres i testkammeret.

Vedlikeholdbarhet – Hvor godt er systemet desginet for å kunne vedlikeholdes over lengre tid. For eksempel hvor ofte må slitasjedeler byttes ut, eller hvor mye generelt vedlikehold som må gjøres.

Mobilitet – Hvor lett det er å flytte vindtunnelen, med tanke på størrelse og vekt, eventuelt å måtte dele den i flere komponenter for transport.

Brukervennlighet – Hvor enkelt det er å bruke systemet og sette seg inn i hvordan det virker.

Modifiserbarhet – Hvor enkelt vil det være å kunne gjøre fremtidige endringer i systemets livssyklus.

Strømningskvalitet – Kvalitet på luft i møte med testmodell; homogenitet for lufthastighet, mer laminær, turbulens intensitet.

7.7. Valgt konsept

Etter at vi presenterte de forskjellige konseptene til oppdragsgiver med både morfologisk diagram (Avsnitt 7.5), Pugh-matrise (Avsnitt 7.6) og budsjett (Avsnitt 7.8) ønsket oppdragsgiver å velge **konsept 2**, som vi også mener er det beste konseptet av de tre. Konsept 2 dekker kravspesifikasjonen basert på oppdragsgivers ønsker, er innenfor budsjettrammen og er det konseptet som gir mest funksjonalitet i forhold til pris. Konseptet for dette designet har vi laget en modell for, se Figur 7.14

Kriterie	**Konsept 1**	**Konsept 2**	**Konsept 3**
Kostnad	+	0	-
Konstruksjonstid	+	0	-
Brukersikkerhet	0	0	0
Styringspresisjon	-	0	0
Målepresisjon	0	0	+
Vedlikeholdbarhet	+	0	-
Mobilitet	0	0	-
Brukervennlighet	-	0	0
Modifiserbarhet	-	0	0
Strømningskvalitet	-	0	0
Sum og rank			
Sum:	-1	0	-3
Rank:	2	1	3

Tabell 7.1.: Pugh-matrise.

7.8. Budsjett

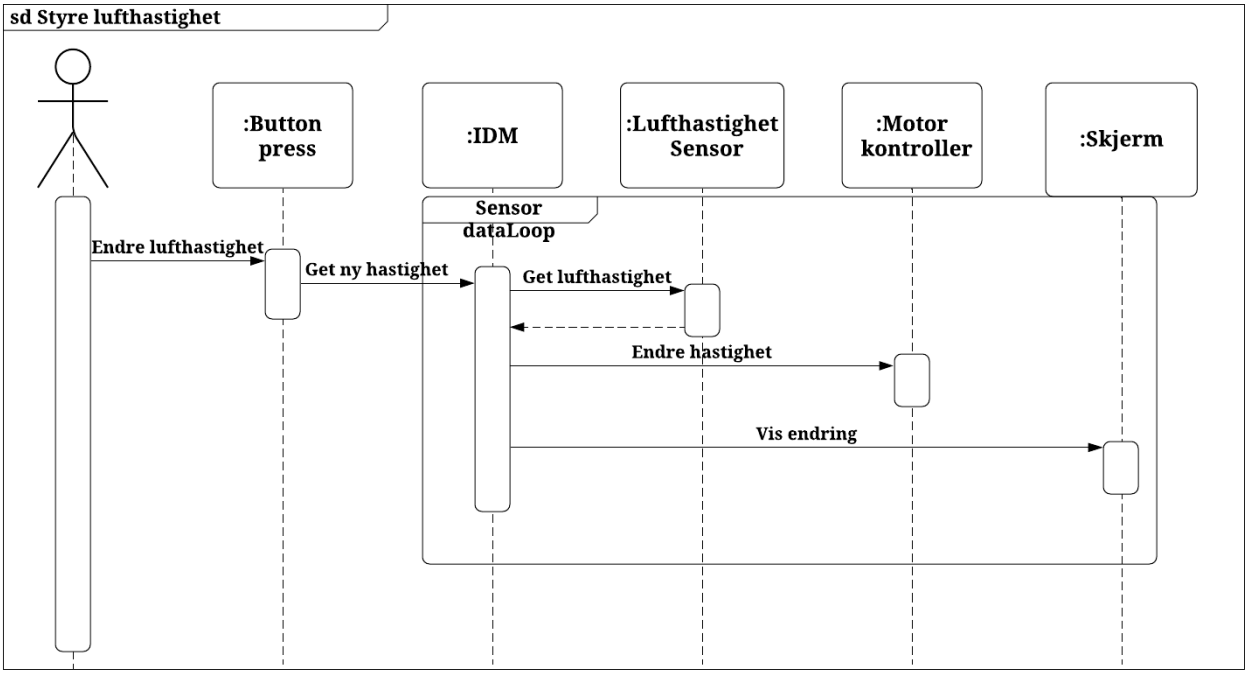
Etter at vi fant ut hvilke konsepter vi ønsker å presentere for oppdragsgiveren, så må vi finne ut av hvor mye hvert konsept vil koste med komponentene som vi har valgt ut Vedlegg D. Det vil kunne hjelpe oppdragsgiver med å få et overblikk på hva de forskjellige konseptene vil ende opp med å koste. Det er derfor det er viktig for oss å kunne vise til hvilket konsept som gir oppdragsgiveren best funksjonalitet til sitt bruk som kommer i formen av en Pugh matrise (seTabell 7.1).

7.9. Funksjonsblokkdiagram (FBD)

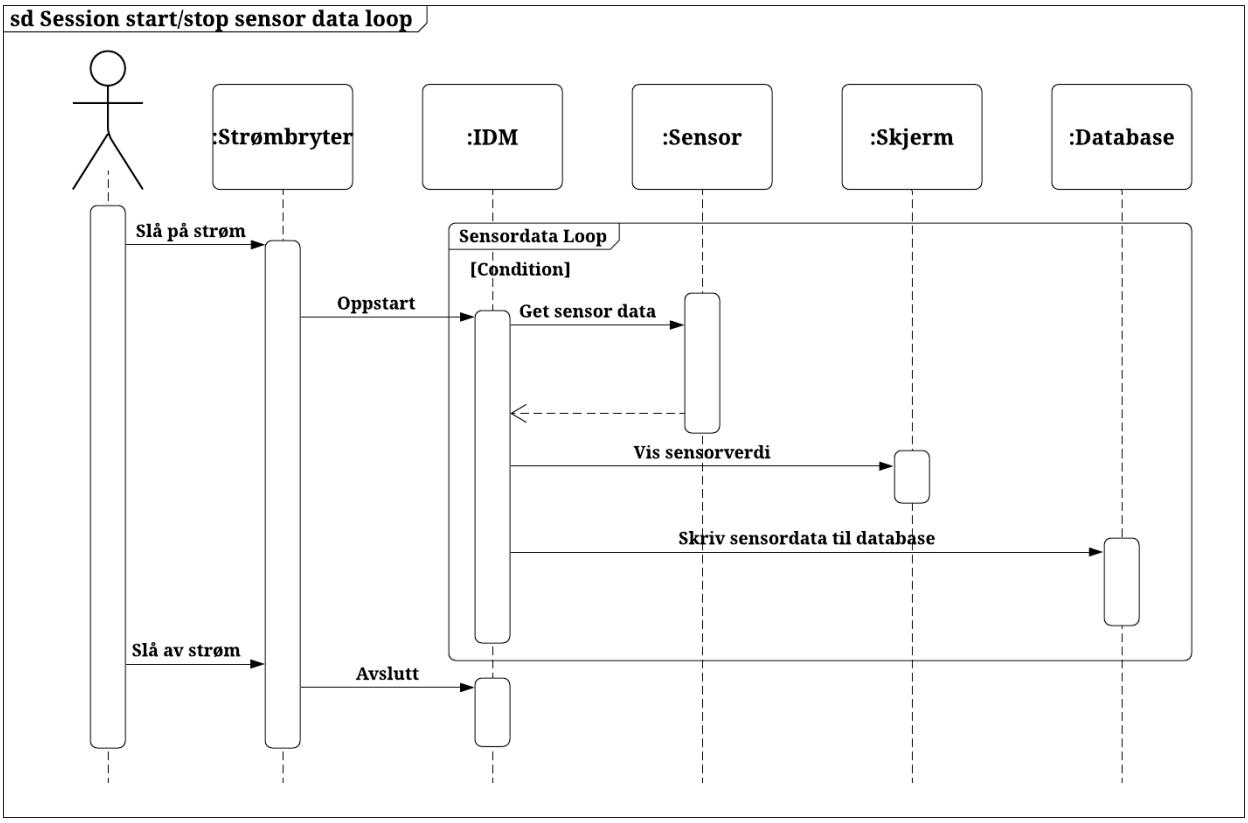
Etter som at vi har valgt å gå fremmover med konsept 2. Så setter vi opp en FBD for systemet vårt slik at vi bedre kan forstå hvordan funksjonene i systemet skal utføres Figur 7.15.

7.9.1. Funksjonsdiagram

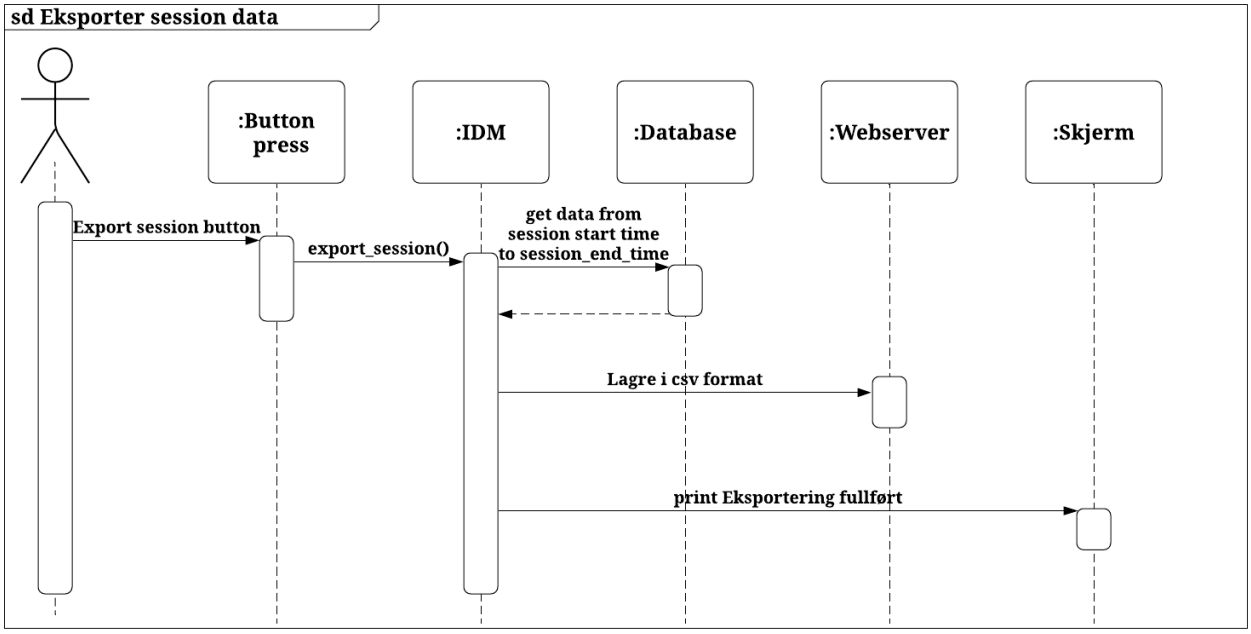
Funksjonsdiagram viser oss hvilke av systemets delfunksjoner som må utføres for at hele systemfunksjonen skal utføres. Som i eksemplet vi har tatt ved “Måle krefter på modellen i testkammeret” som da blir hele systemfunksjonen, så må vi vite hva systemet vårt må gjøre for at denne funksjonen skal bli utført (se Figur 7.16).



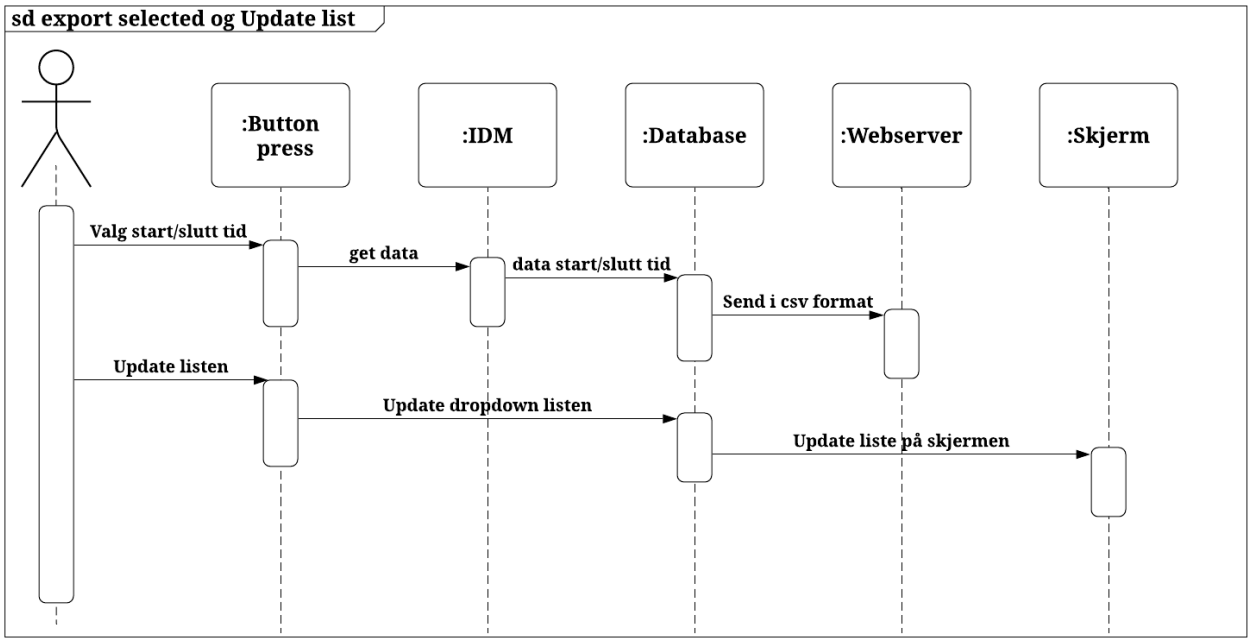
Figur 7.8.: Styre lufthastighet.



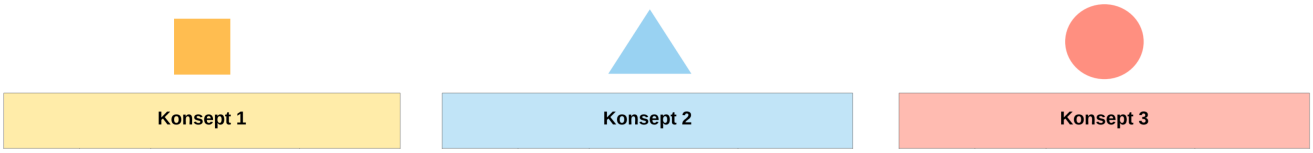
Figur 7.9.: Start og stopp session.



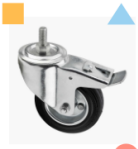
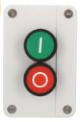
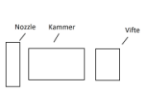
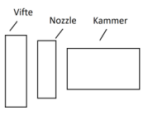
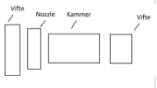
Figur 7.10.: Eksportering av filer.



Figur 7.11.: Eksportering tids valg og oppdatering.

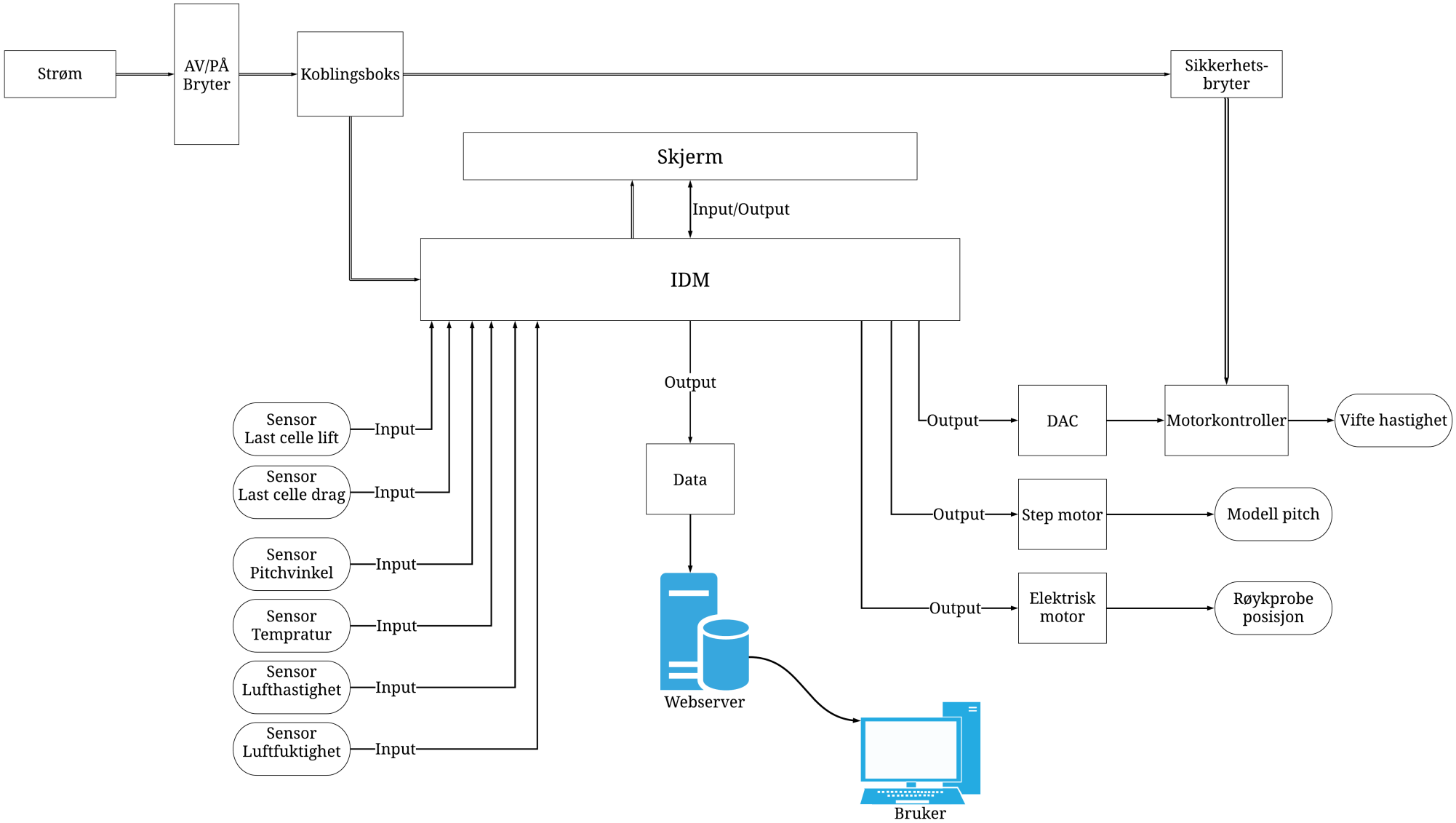


Figur 7.12.: Figurer til konsepter.

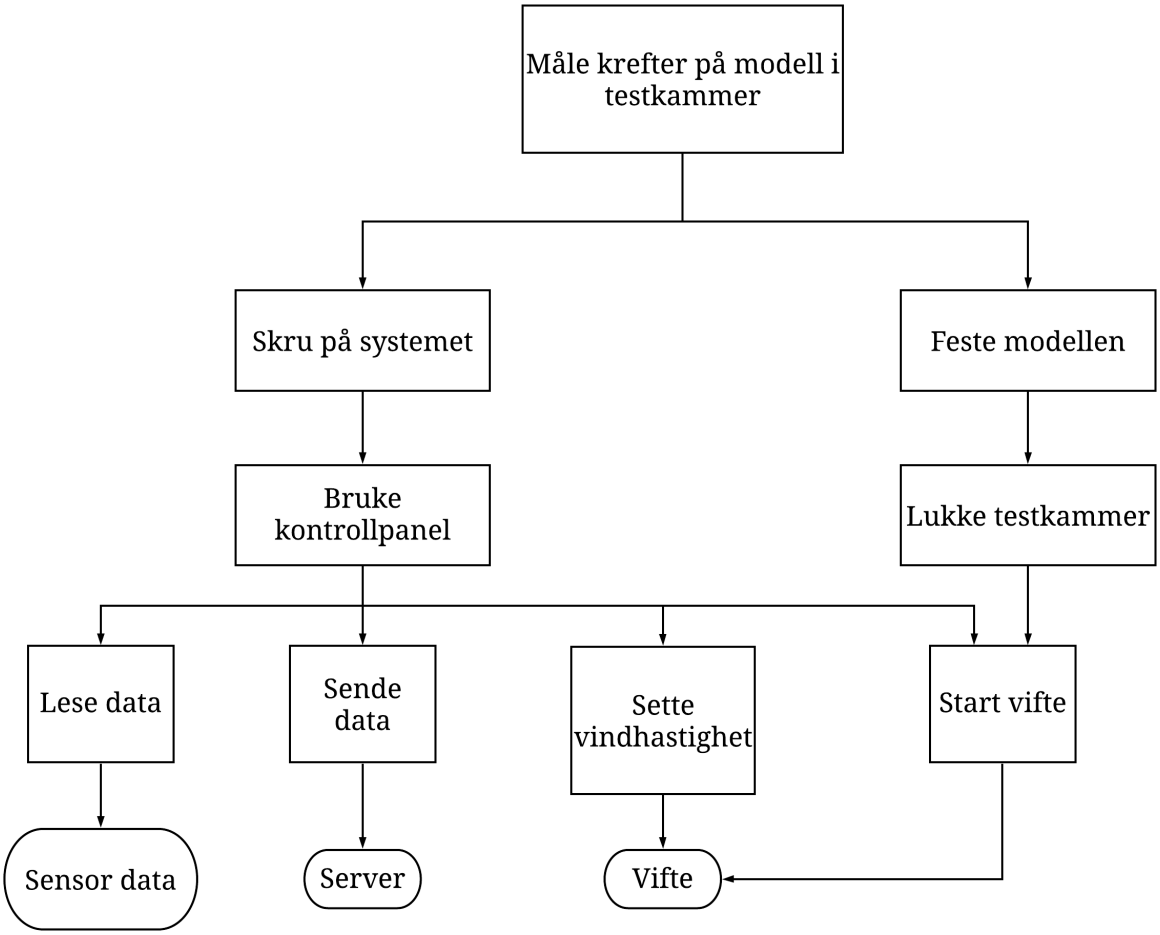
ID	Prioritet	Funksjoner	Løsning 1	Løsning 2	Løsning 3	Løsning 4	Løsning 5	Løsning 6
K.M 01	A	**Måle lufthastighet**	 Pitorør	 Hot Wire Anemometer	 Skålkorsanemometer			
	B	**Lufthastighetsmåler mobilitet**	Stasjonær	Flyttbar				
		Metode for posisjonering av lufthastighetsmåler	Manuelt	Mekanisk				
		Brukergrensesnitt for mekanisk endring av lufthastighetsmåling						
K.VT 08	A	**Transportering**		Dele vindtunnelen i biter(modulær)				
	A	**Start/Stopp**						
K.VT 04	A	**Styre lufthastighet**	Kontrollsløyfe (feedback controller)	Vifteturtall vindhastighetstabell				
K.VT 04	A	**Justering av lufthastighet (av bruker)**						
	A	**Generere luftstrøm**	 Vifte bak	 Vifte foran	 Vifte foran og bak	 Jetmotor	 Turbin	

Figur 7.13.: Utklipp fra Morfologiske diagrammet.

Figur 7.14.: Rendret design av konsept 2.



Figur 7.15.: Funksjonsblokkdiagram.



Figur 7.16.: Funksjons diagram.

8. Elektroteknisk design

8.1. Om sensorer

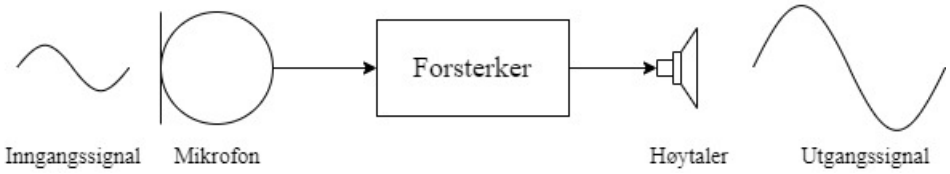
Denne seksjonen er ment for å dekke den grunnleggende bakgrunnsteorien om sensorer for å gi en bedre forståelse til å kunne velge riktige sensorer til delsystemene som skal designes. Flere av våre delsystemer er avhengige av sensorer for å kunne utføre de nødvendige funksjonene som er spesifisert i kravene.

8.1.1. Introduksjon

For at vindtunnelen skal være istand til å gjøre de målingene som er spesifisert i kravene er systemet vårt helt avhengig av sensorer. Sensorene har som oppgave å måle variable tilstander fra den fysiske verdenen og generere et elektrisk signal som er proporsjonalt med størrelsen på den fysiske tilstanden. Dette gjøres ved at en typisk sensor tar imot et inngangssignal fra den fysiske verdenen som omformes til et utgangssignal som tas imot og leses av en ekstern enhet. Figur 8.1 viser et eksempel på en sensor som fungerer som en lydforsterker. Den tar imot et inngangssignal i form av en lydbølge, dette lydsignalet forsterkes og sendes videre til en høytaler som spiller av lyden på et høyere lydnivå.

Det elektriske signalet som sensoren genererer må være kraftig nok til at det kan avleses på en måler, eksternt display eller en datamaskin. Sensor signalet må være kodet i riktig format avhengig av hvordan det skal overføres og avleses. Ved overføring av sensordata er man interessert i å få lest mest mulig nøyaktige måleresultater som stemmer overens med de virkelige

målingene. Uønsket data som skulle finne på å bli inkludert i dataoverføringen må filtreres bort.



Figur 8.1.: Eksempel på sensor som lydforsterker.

8.1.2. Kategorisering

En sensor kan kategoriseres som ordinær eller smart avhengig av hvordan den er designet.

En ordinær sensor er en analog sensor. En analog sensor kan for eksempel være en temperatursensor. Temperatur endres kontinuerlig over tid og ved endring av temperatur så vil sensoren generere en endring i elektrisk motstand som vil skape et elektrisk signal som er proporsjonalt med størrelsen på temperaturen.

En smart sensor har ekstra innebygde funksjoner, en innebygd funksjon kan for eksempel være analog til digital signalkonvertering slik at vi får en digital sensor, det vil si at den enten sender et signal med en fast størrelse, eller ingen signal i det heletatt (binær av/på kommunikasjon).

8.1.3. Ytelsesterminologi for sensorer

For å kartlegge ytelsen på en sensor, så eksister det spesifikke nøkkelterminologi begreper som man kan hente ut fra sensorens datablad eller spesifikasjonsoversikt. De mest sentrale begrepene som inngår i nøkkelterminologien som dekkes i dette avsnittet er hentet fra [6].

1. Range og span

Range definerer grenseområdene på inngangssignalet som sensoren kan måle. **Span** definerer absoluttverdien på begrenset måleområde. En temperatursensor kan for eksempel ha en **range** fra -25 °C til +25 °C som gir et **span** på 50 °C.

2. Error

Error er differansen mellom det målte resultatet og det virkelige resultatet:

$$\textbf{Error} = \text{Målt verdi} - \text{Virkelig verdi}$$

3. Accuracy

Accuracy definerer sensorens målenøyaktighet. En temperatursensor kan for eksempel ha en målenøyaktighet med et avvik på ± 2 °C.

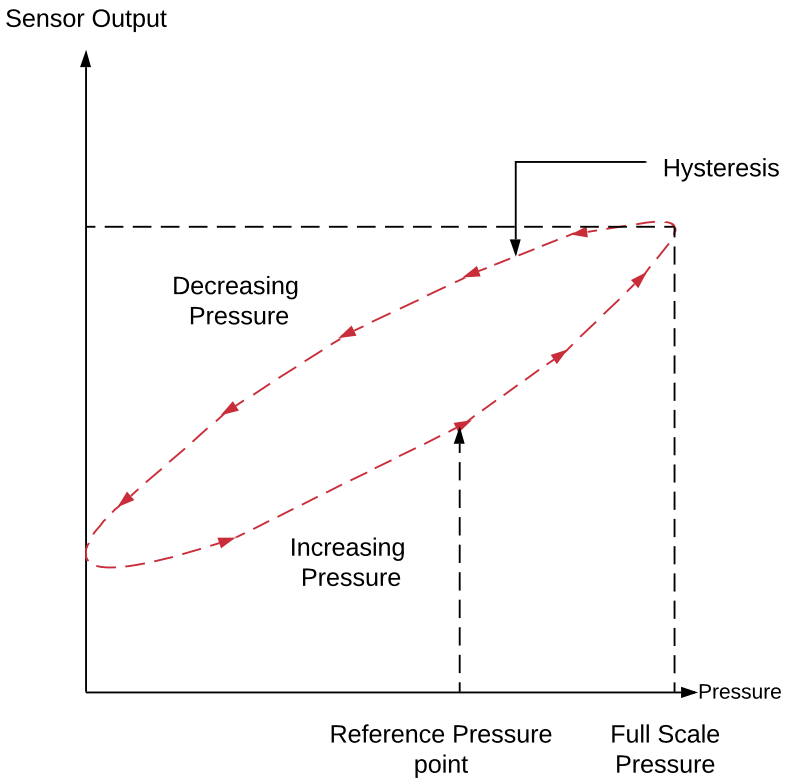
Det betyr at den målte verdien kan ligge et sted mellom ± 2 °C av den virkelige verdien. Det mest vanlige er å definere målenøyaktigheten som en prosent av sensorens **range**. En sensor kan for eksempel være spesifisert til å ha målenøyaktighet (oppgitt i prosent) som $\pm 5\%$ av total **range** til sensoren. La oss si det er snakk om en temperatursensor med range 0-200 °C, en målenøyaktighet på $\pm 5\%$ vil gi et måleresultat som ligger et sted mellom ± 10 °C av den virkelige verdien.

4. Sensitivity

Sensitivity defineres som forholdet som indikerer størrelsen på output per enhet av input, altså output/input. En temperatursensor kan for eksempel gi $0.7\,\Omega/°C$.

5. Hysteresis error

Sensorer kan gi forskjellig output fra samme verdi målt avhengig av om denne verdien ble nådd av en kontinuerlig økende eller synkende ladning, denne effekten er kalt for **hysteresis error**. Figur 8.2 viser en output der **hysteresis error** er maks differansen i output for økende og synkende verdier.



Figur 8.2.: Graf som illustrerer **hysteresis error**.

6. Non-linearity error

For mange sensorer er det antatt at det er et lineært forhold mellom input og output over fungerende range. Et lineært forhold vil si at en grafisk representasjon av output plottet i forhold til input vil gi en rett linje. I virkeligheten er det få sensorer som har et virkelig lineært forhold, noe som fører til error som følge av antagelsen av lineært forhold. **Non-linearity error** er definert som maks avstand fra den rette linjen.

7. Repeatability/Reproducability

Repeatability og **reproducability** er terminologier som beskriver sensorens egenskap for å produsere samme output verdi for den samme repeterende input verdien. Error-verdien som følge av dette beregnes som en prosentandel av rangen til output verdiene:

$$\textbf{Repeatability} = \frac{\text{Max verdi} - \text{Min verdi}}{\text{Full range}} \times 100$$

8. Stability

En sensors **stability** beskrives som egenskapen til å gi samme output ved en konstant input over et visst tidsintervall. Begrepet **drift** brukes til å beskrive endringen av output over tid og beskrives som en prosentandel av full range output. Begrepet **zero drift** brukes for endringer over tid når det er 0 input.

9. Dead band/time

Begrepet **Dead band** brukes til å beskrive intervallet av input verdier som ikke gir noe output. For eksempel et anemometer som består av en rotor som ikke vil gi noe utslag før det har blitt oppnådd en viss vindhastighet som overgår hvilefriksjonen til rotoren. Begrepet **dead time** brukes for å angi tiden det tar fra input gir noe output.

10. Resolution

Når en input varierer kontinuerlig over rangen, så vil output signalet endres i små steg av gangen. Begrepet **resolution** beskriver sensorens oppløsning, det vil si den minste endring av input som kreves for å få en observerbar endring av output.

11. Output impedance

Begrepet **output impedance** betyr utgangsimpedans på norsk. Når en sensor skal kobles til en elektronisk krets så er det nødvendig å vite sensorens utgangsimpedans. Man ønsker å vite utgangsimpedans verdien til sensoren, siden denne kan påvirke den elektroniske kretsen avhengig av om sensoren kobles i serie eller parallell med inngangsimpedansen til den elektroniske kretsen.

8.1.4. Statiske og dynamiske karakteristikker

Følgende avsnitt er basert på teori i [6].

Statisk karakteristikk er verdien man får når "steady-state" tilstand er oppnådd, det vil si når output verdien til sensoren har nådd en stabil verdi etter å ha fått en input verdi tilført. Dynamisk karakteristikk refererer til oppførselen til sensoren i tidsintervallet for en endring av input

verdi i forhold til tiden det tar for utgangsverdien å oppnå "steady-state" tilstand. Dynamiske karakteristikker er definert som sensorens respons i forhold til input på spesifikke former. Typiske input former er "step-input" der input verdien går fra 0 til en konstant verdi, "ramp-input" der input verdien øker lineært over tid eller en sinusformet input som opererer på en spesifikk frekvens.

1. Response time

Response time er tiden som går fra en konstant input er tilført sensoren og til den har oppnådd en output som tilsvarer en viss prosentandel av input verdien, for eksempel 95%. Figur 8.3 viser et grafisk eksempel på step-responsen til en sensor.

2. Time constant

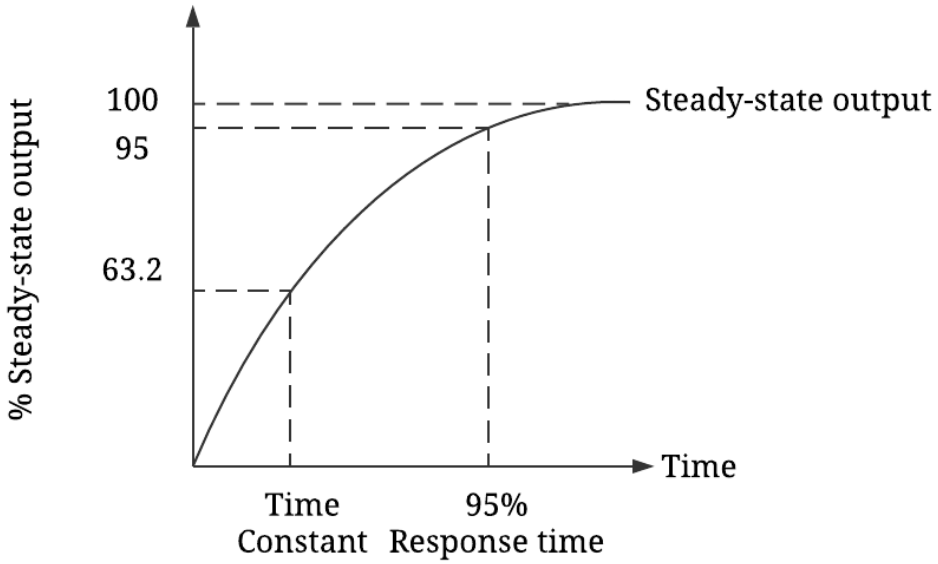
Time constant er 63.2% responstiden for input verdien. Tidskontanten definerer tregheten til sensoren, som sier noe om hvor fort sensoreren reagerer til endring i input verdien.

3. Rise time

Rise time er tiden det tar for output verdien å nå en spesifikk prosentandel av steady-state verdien. Stigningstiden måles gjerne i intervallet 10-95% av steady-state verdien.

4. Settling time

Settling time er tiden det tar for en output å slå seg til ro ved en viss prosentandel av steady-state verdi, gjerne innenfor 2% av steady-state verdien.



Figur 8.3.: Grafisk illustrasjon av step-respons.

8.1.5. Om støy

Uønsket støy kan forekomme i sensorkretser som kan negativt påvirke sensormålingene slik at man får unøyaktige eller ufullstendige måleresultater. Støy kan oppstå i ulike former fra diverse støykilder, man skiller mellom intern og ekstern støy.

Intern støy er støy der støykildene kommer fra fenomener som forekommer inne i selve kretsen som tar imot inngangssignalet som sensoren måler, for eksempel en komponent som er en del av sensorkretsen kan ha en egenskap som negativt påvirker signalet vi ønsker å måle. Ekstern støy kommer fra støykilder utenfor systemet, et eksempel kan være en sensor som tar imot et uønsket signal fra en støykilde utenifra som vil negativt påvirke signalet vi ønsker å måle. Eliminering av støy er ekstremt viktig for å oppnå korrekte måleresultater. Riktig plassering av sensoren og kartlegging av systemets kontekst er derfor viktig å ta i betrakning slik at man kan gjøre forebyggende tiltak med tanke på støy.

8.2. Valg av sensorer

Valg av sensorer har en nær tilknytning til kravene for prosjektet. Hvilke sensorer vi velger å bruke vil derfor ha en stor påvirkningskraft for å avgjøre om et krav er oppfylt eller ikke, da spesielt krav som omhandler målinger, men også krav for funksjonalitet. Det er derfor veldig viktig å gjøre grundig research på de sensorene vi vurderer å kjøpe inn for å forsikre oss på forhånd at sensorene er kompatible med systemkravene og delsystemet de skal integreres i. Konsekvensene av å velge feil sensor vil øke risikoen betraktelig for at kravene ikke blir oppfylt. Andre risikoer som vil øke er tap av budsjett og arbeidstid.

Seksjon 8.1 dekker grunnleggende teori og begreper man burde ha kjennskap til for å kunne hente ut informasjon om sensorer fra tilhørende produktdatablad. Denne seksjonen tar for seg hva man bør ta hensyn til iforhold til systemet som sensoren skal ha et grensesnitt for samt beskrivelse av spesifikke sensorer vi har valgt å bruke til dette prosjektet.

8.2.1. Sensortyper

I vårt prosjekt kan vi velge å gå for basis eller smarte sensorer. Seksjon 8.1 gikk såvidt innom konseptet med smarte sensorer, denne underseksjonen tar for seg fordelene og ulempene med begge typene.

Basis sensorer

En basis sensor inneholder ingen ekstra funksjoner utenom den grunnleggende funksjonen som innebærer avlesning av en fysisk analog kvantitet som transformeres til et analogt elektrisk signal som er proporsjonalt med den fysiske kvantiteten.

Fordelen med en basis sensor er at den er enkel å masseprodusere, dette gir en rimeligere pris. En basis sensor gir også mer frihet til å lage egendefinerte ekstrafunksjoner som gir mer fleksibilitet, baksiden med dette er at å implementere ekstrafunksjoner på egenhånd er mer komplisert og tidkrevende da man er nødt til å designe funksjonene selv og gå til innkjøp av nødvendige ekstrakomponenter som skal behandle det innkommende signalet.

Smarte sensorer

En smart sensor inneholder vanligvis en rekke innebygde ekstra funksjoner for dataprosessering som tar for seg støyfiltrering, signalforsterkning og signalomformning slik at det innkommende signalet er ferdig klartgjort til seriell overføring til enheten som skal ta imot signalet.

Fordelen med en smart sensor er den innebygde flerfunksjonaliteten som eliminerer behovet for ekstra komponenter til dataprosessering. Smarte sensorer frigjør også minnekapasiteten til for eksempel en mikrokontroller. En sensor kan ikke klassifiseres som “smart” hvis den ikke består av en egen innebygd prosessor. Ulempen med smart sensorer er at de er dyrere i pris og gir mindre frihet til å stille inn ekstra funksjoner.

Sensorer til vindtunnelen

Vi har bestemt oss for å benytte oss av analoge basis-sensorer for dette prosjektet så langt det lar seg gjøre. Årsaken til det er fordi de er rimeligere og fordi mikrokontrolleren som sensorene skal ha grensesnitt mot skal fungere som en felles dataprosesseringsenhet for alle sensorene. Mikrokontrolleren er istand til å lese av analoge signaler og oversette dem til diskrete verdier for digital avlesning.

Det er viktig å studere databladene til de sensorene som er aktuelle for dette prosjektet, der kan vi hente ut sensorenes operative egenskaper, blant annet de vi har nevnt i denne underseksjonen. Vi kan også knytte dem opp mot kravene for å bedømme om de er tilstrekkelige nok.

8.2.2. Valg av sensor til vindhastighetmålinger

Dette avsnittet er basert på teori fra [11].

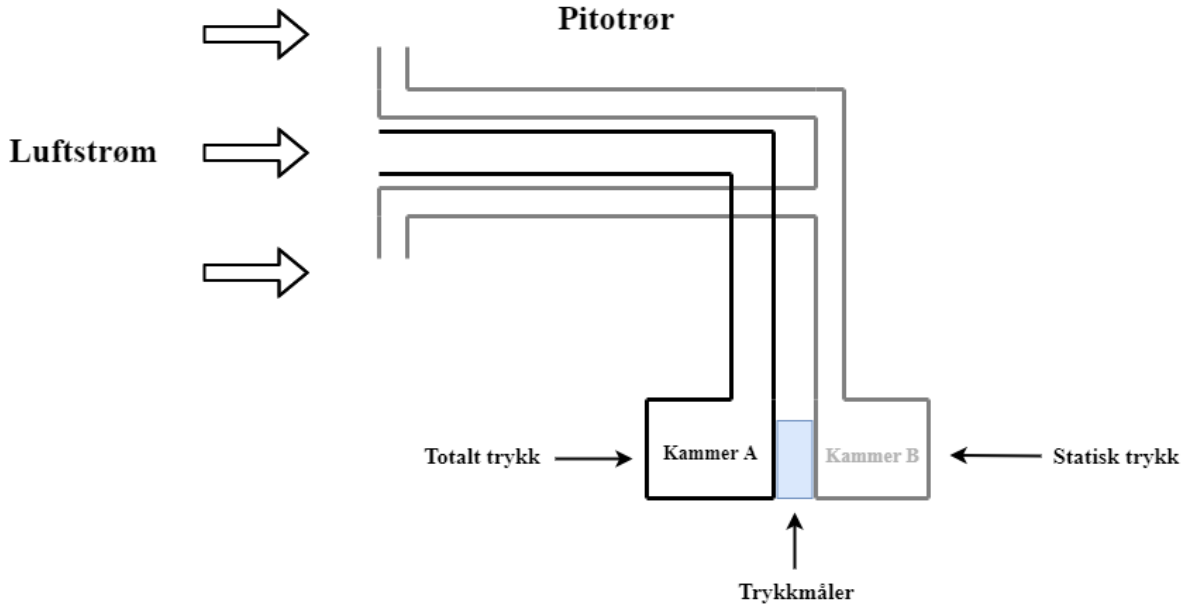
For måling av vindhastighet og statisk og dynamisk lufttrykk i testkammeret for å oppfylle krav K.M 01 og krav K.M 02 har vi valgt å bruke en differensial trykk sensor. Differensial trykk sensorer brukes mye på luftfartøy og gir mer nøyaktige vindhastighetsmålinger enn et typisk anemometer, og er derfor et svært godt valg å bruke til vindtunnelen. Vindhastighetsmålingen er en nødvendig funksjon der vi utvikler et delsystem for å utføre denne spesifikke oppgaven. Vi er også avhengig av at dette delsystemet kan kommunisere med styringssystemet for vindhastigheten.

En trykk sensor kommer som regel med et pitotrør. Et pitotrør er en L-formet tube som brukes til å samle det totale luftrykket og det statiske lufttrykket i hvert sitt kammer. Pitotrøret plasseres rett på luftstrømmen som vist på Figur 8.4. Differansen mellom total trykk og statisk trykk gir oss dynamisk trykk, ved hjelp av Bernoulli's likning kan vi beregne lufthastigheten.

Bernoulli's prinsipp beskrives med likningen:

$$\frac{1}{2}\rho a V^2 a + Pa + \rho agha = \frac{1}{2}\rho b V^2 b + Pb + \rho bghb \tag{8.1}$$

Der ρ er væsketetthet, V er hastighet, P er trykk og h er høyden til flow seksjonen.



Figur 8.4.: Strukturdiagram for pitotrør.

Hvis vi antar ingen endring av gravitasjon og null hastighet ved stagneringspunktet, så kan Bernoulli's likning forenkles til:

$$\frac{1}{2}\rho a V^2 a + Pa = Pb \tag{8.2}$$

På pitotrøret har vi en separat seksjon som tillater kun statisk trykk å bli målt ved hjelp av en port som står vinkelrett på luftstrømretningen (statisk port). Det er dette som gjør at vi kan måle total trykk og statisk trykk separat:

$$P_{\text{total}} = P_{\text{statisk}} + \frac{1}{2}\rho a V^2 \tag{8.3}$$

$$P_{\text{total}} = P_{\text{total}} \tag{8.4}$$

Formelen for dynamisk lufttrykk blir da:

$$P_{\text{total}} - P_{\text{statisk}} = \frac{1}{2}\rho V^2 \tag{8.5}$$

Finner lufthastighet ved å snu på likningen og løse med hensyn på V:

$$V = \sqrt{\frac{2(P_{\text{total}} - P_{\text{statisk}})}{\rho}} \tag{8.6}$$

Pitotrøret kobles til portene på differensial trykksensoreren som leser av det totale trykket og statisk trykk, disse dataene sendes så til mikrokontrolleren som gjør de nødvendige utregningene.

Til prosjektet vårt har vi valgt å bruke MPXV7002DP som differensial trykksensor med pitotrør som tilbehør. Denne sensoren er spesialtilpasset til bruk for mikrokontrollere og gir et analogt utgangssignal som er proporsjonalt med tilført trykk. Figur 8.5 viser et bilde av sensoren med pitotrør inkludert.

Figur 8.5.: MPXV7002DP sensor med pitotrør.

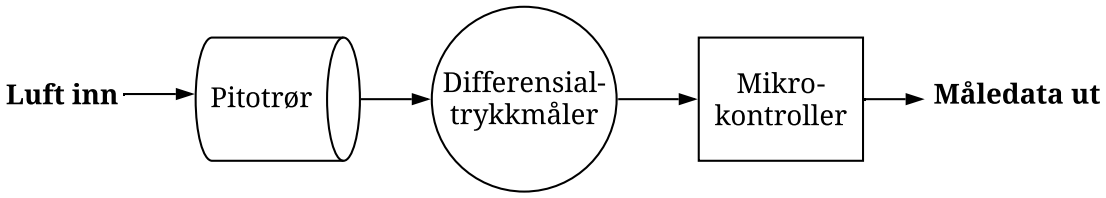
Tabell 8.1 viser utdrag fra sensorens produktdatablad[30] som beskriver sensorens operative egenskaper.

Characteristic	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
Pressure Range	Pop	-2,0	-	2.0	kPa
Supply Voltage	Vs	4,75	5.0	5.25	Vdc
Supply Current	I0	-	-	10	mAdc
Pressure Offset (10–60 °C)	Voff	0.25	0.5	0.75	Vdc
Full Scale Output (10–60 °C)	Vfso	4.25	4.5	4.75	Vdc
Full Scale Span (10–60 °C)	Vfss	3.5	4.0	4.5 V	Vdc
Accuracy (10–60 °C)	-	-	±2.5	±6.25	%Vfss
Sensitivity	V/P	-	1.0	-	V/kPa
Response Time	tR	-	1.0	-	ms
Output Source Current at Full Scale Output	Io+	-	0.1	-	mAdc
Warm-Up Time	-	-	20	-	ms

Tabell 8.1.: Operative egenskaper for MPXV7002DP [30].

8.2.3. Systemstruktur for vindhastighet-måling

Blokkskjemaet i Figur 8.6 viser systemstrukturen for delsystemet for vindhastighet-måling. Luft samles inn i pitotrøret som skaper et statisk og et totalt lufttrykk som samles i hvert sitt kammer i trykksensoren. Disse trykkmålingene sendes så til en mikrokontroller som tar differansen mellom trykkmålingene og regner ut lufthastighet målt i m/s som vises på et eksternt display.



Figur 8.6.: Delsystemet for vindhastighet-måling.

8.2.4. Valg av sensor til kraftmålinger

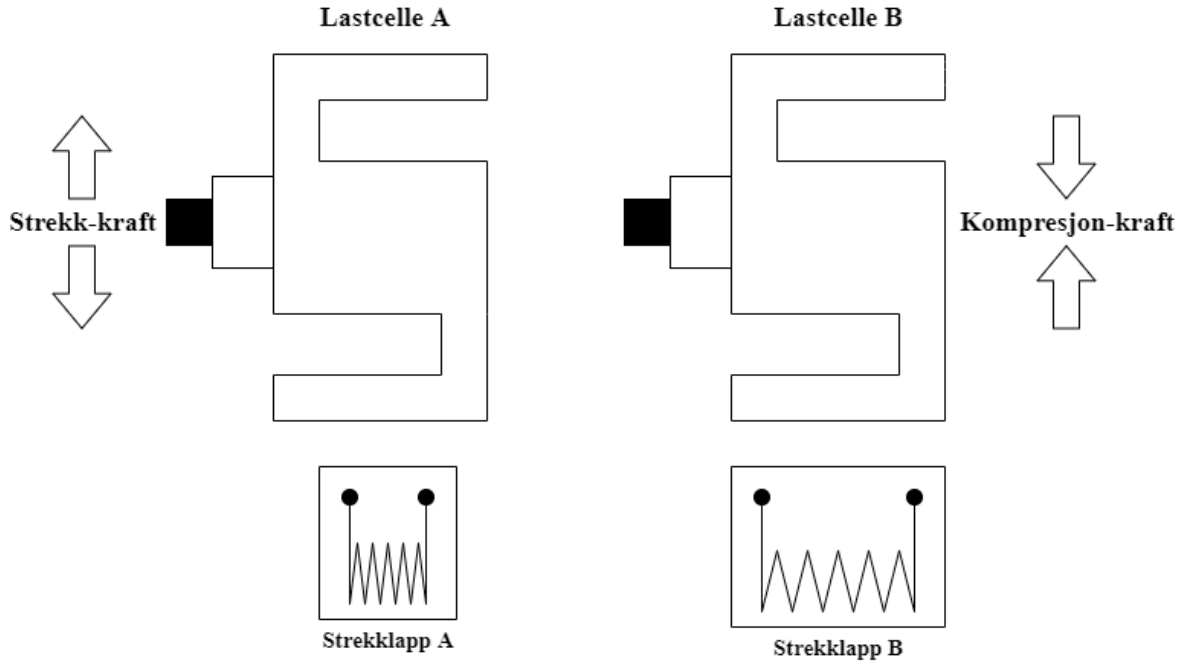
Dette avsnittet er basert på teori fra [35].

Til vårt prosjekt er vi interessert i å måle to krefter, lift og drag på testmodellen i vindtunnelen for å oppfylle krav K.M 08. Lift måles i vertikal retning og drag i horisontal retning, begge kreftene har måle-enheten Newton (N). Lastceller brukes til å måle trekkkraft, kompresjon, trykk eller moment. Det finnes mange forskjellige typer lastceller, men til vårt prosjekt har vi valgt å bruke strekkmåler lastceller som er den typen som er mest standard brukt i industrien, og som er velegnet til å måle de kreftene vi er ute etter. Denne typen lastcelle er kjent for å være veldig nøyaktig, allsidig og rimelig.

En strekkmåler lastcelle består av en metall kropp som er lagd av aluminium, legert stål eller rustfritt stål som sikrer strekklappene som befinner seg inni metall kroppen. Når en kraft

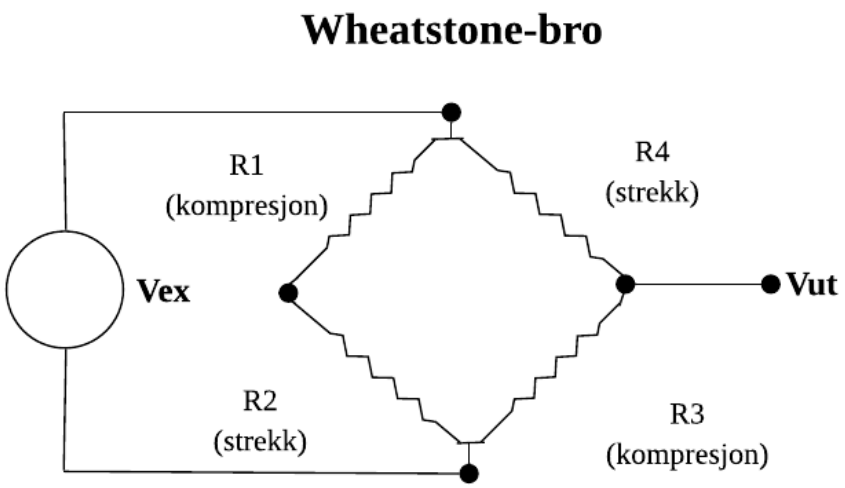
virker på lastcellen, vil den bli deformert. Deformasjonen vil medføre at strekklappene inni lastcellen endrer form. Endring av form vil føre til en endring av elektrisk motstand som kan måles som en endring i spenning som er proporsjonalt med mengden av påført kraft på lastcellen.

Selve strekklappene består av metall tråd eller metallfolie stripe som er satt opp i et rutenett mønster. Når formen på strekklappen endres, så skjer det også en endring i elektrisk motstand. Hensikten med rutenett mønsteret er for å arrangere strekklappen slik at man får en økning i elektrisk motstand når strekklappen strekkes, og reduksjon av elektrisk motstand under kompresjon. Figur 8.7 viser en illustrasjon av strukturen til en typisk strekkmåler lastcelle. Her ser man at ved strekk så blir ledningene lengre med tynnere mellomrom som vist på strekklapp A, dette vil medføre til en økning i motstand. Ved kompresjon blir ledningene kortere med større mellomrom som vist på strekklapp B, dette vil medføre en reduksjon i motstand.



Figur 8.7.: Lastcelle struktur.

En lastcelle består som regel av flere strekklapper, da endring i elektrisk motstand av en enkel strekklapp er veldig liten. Det mest vanlige er å bruke fire strekklapper i en wheatstone-bro som vist på Figur 8.8.



Figur 8.8.: Fire strekklapper arrangert i en Wheatstone-bro.

En wheatstone-bro består av fire balanserte motstander og en konstant eksitasjonspenning som blir tilført (Vex). Utgangsspenningen Vut er variabel og avhenger av formen på strekklappene. Utgangsspenningen er lik 0 hvis motstandene er balansert:

$$\frac{R1}{R2} = \frac{R4}{R3} \tag{8.7}$$

Så fort en av motstandene endrer verdi, så vil også utgangspenningen endre verdi. Endring av utgangsspenning kan måles og tolkes via Ohm's lov som sier at strømmen som går gjennom en leder (I målt i Ampere) mellom to punkter, er direkte proporsjonal med spenningen (V målt i Volt) over de to punktene. Motstanden (R målt i Ohm) er introdusert som en konstant i

forholdet, uavhengig av strømmen. Ohm's lov uttrykkes som:

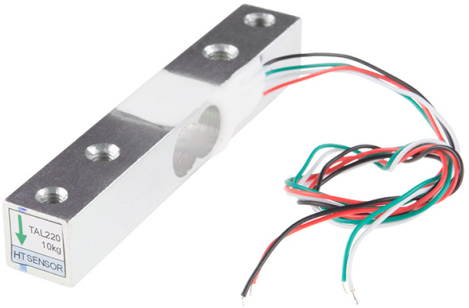
$$I = V/R \tag{8.8}$$

Ved å anvende prinsippet med Ohm's lov til Wheatstone-broen får vi følgende likning:

$$V_{ut} = (\frac{R3}{R3 + R4} - \frac{R2}{R1 + R2})V_{ex} \tag{8.9}$$

I en lastcelle så erstattes de fire motstandene med strekklapper som fungerer som variable motstander i et alternerende mønster for å måle strekk og kompresjon som vist på Figur 8.8. Når trykk blir påført så endres formen og motstanden på strekklappene som gir en utgangsspenning som er proporsjonal med påført kraft på lastcellen.

Det finnes forskjellige typer underkategorier av strekkmåler lastceller. I vårt prosjekt er det mest aktuelt med Single Point last celler, en for å måle lift i vertikal retning, og en for å måle drag i horisontal retning. Til vårt prosjekt har vi valgt å bruke TAL220 lastcelle som kapasitet på 10 kg last. Figur 8.9 viser bildet av lastcellen og Tabell 8.2 viser utdrag fra databladet til lastcellen.

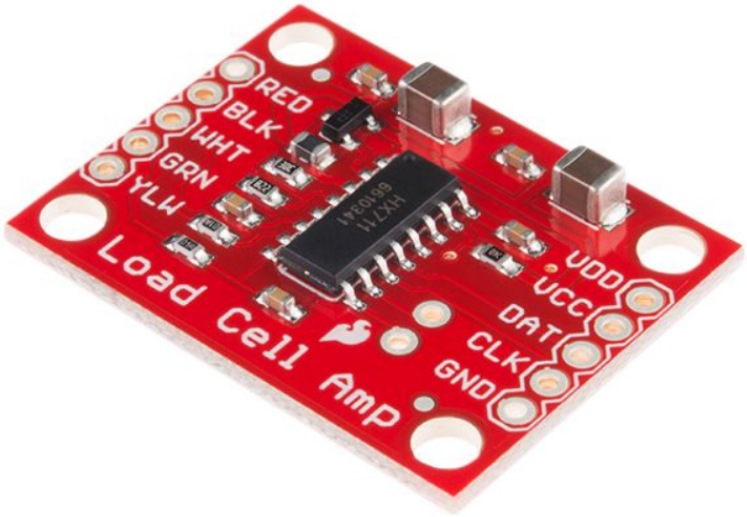


Figur 8.9.: TAL220 lastcelle [12].

Characteristic	**Unit**	**Value**
Capacity	kg	3.5, 5, 10, 20, 25, 30 (aluminum); 80, 100, 120, 200 (alloy steel)
Safe overload	%FS	120
Ultimate overload	%FS	150
Rated output	mV/V	1.0 ± 0.15
Excitation voltage	Vdc	5-10
Combined error	%FS	± 0.05
Zero unbalance	%FS	± 0.1
Non-linearity	%FS	± 0.05
Hysteresis	%FS	± 0.05
Repeatability	%FS	± 0.03
Creep	%FS/3min	± 0.05
Input resistance	Ω	1000 ± 15
Output resistance	Ω	1000 ± 10
Insulation resistance	M Ω	≥ 2000
Operating temperature range	°C	-10 - +55
Compensated temperature range	°C	-10 - +40
Temperature coefficient of SPAN	%FS/10°C	± 0.05
Temperature coefficient of ZERO	%FS/10°C	± 0.05
Electrical connection	cable	4 color wire (standard) or 4 shielded PVC cable (0.8 x 220 mm)

Tabell 8.2.: Operative egenskaper for TAL220 lastcelle [12].

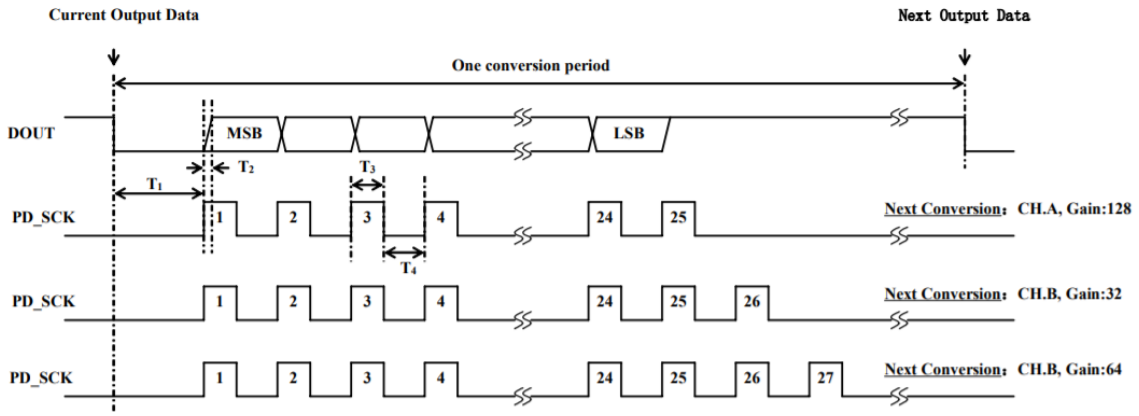
Utgangssignalet som denne lastcellen genererer er for svakt til å kunne leses av en mikrokontroller, så vi trenger dedikerte lastcelle forsterkere. Det finnes allerede en løsning for det ute på markedet. Til vårt prosjekt bruker vi en HX711-lastcelleforsterker som har tilkoblingsmuligheter for to lastceller. Figur 8.10 viser bilde av kretskortet. Figur J.1, Figur J.2 og Figur 8.11 viser utdrag fra databladet [29] til lastcelleforsterkeren. Se også datablad vedlegg Vedlegg J.



Figur 8.10.: HX711-lastcelleforsterker kort.

HX711 lastcelleforsterkeren er en presis 24-bit Analog til Digital konverter (ADC), noe som gjør den mer presis enn om vi hadde sendt de analoge måledataene fra lastcellene direkte til Arduino kortet som bare har 10-bit Analog til Digital konvertering. Kommunikasjonsprotokollen til HX711 forsterkerkortet er ikke standardisert (se Figur 8.11), men bruker følgende prosess: Forsterkeren har tre forskjellige trinn som kan velges. Når data ikke er klar for overføring, så er digital utgangsport DOUT høy, seriell klokke input PDSCK burde være lav. Når DOUT blir lav, så indikerer det at data er klar for overføring. Ved å anvende 25, 26 eller 27 positive klokke pulser på PDSCK pin, så blir data skiftet ut fra DOUT utgangsporten. Hver PDSCK puls skifter ut et databit av gangen med den mest signifikante bit først helt fram til alle 24 bits er skiftet ut, etter det vil DOUT pin gå tilbake til høy. Hvem av klokkepulsene man

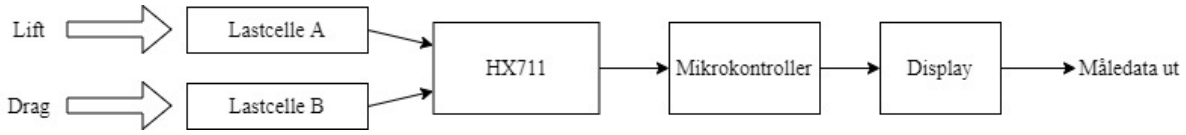
velger avhenger av hvilken forsterkertrinn som er valgt som input.



Figur 8.11.: Kommunikasjonsprotokoll for HX711 for data output, input, valg av forsterkertrinn, timing og kontroll [29].

8.2.5. Systemstruktur for kraftmålinger

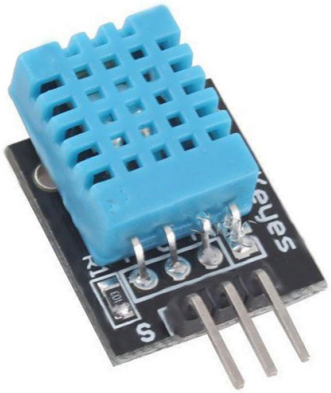
Blokkskjemaet i Figur 8.12 viser systemstrukturen for delsystemet for kraftmålingene. Lastcelle A måler lift kraften i vertikal retning og lastcelle B måler drag kraften i horisontal retning. Når lastcellene blir påført krefter så sendes et elektrisk signal som er proporsjonalt med den påførte kraften til lastcelleforsterkeren HX711 som forsterker signalene slik at mikrokontrolleren kan lese de av og konvertere dem til måleenheten Newton som vises på displayet.



Figur 8.12.: Delsystemet for måling av lift og drag.

8.2.6. Valg av temperatur/fuktighet sensor

Til vårt prosjekt har vi valgt å gå for DHT11 sensoren for måling av temperatur og fuktighet inni testkammeret. DHT11 er en smart sensor som gir et kalibrert digitalt signal på utgangen. Denne sensoren kan kobles direkte til en Arduino mikrokontroller. Det finnes også eksisterende Arduino software bibliotek som gjør det enkelt å få lest av måledataene til en skjerm. Sensoren består av et resistivt måleelement for fuktighet og en termistor for måling av temperatur som kobles til en 8-bit mikrokontroller som gir god kvalitet, kjapp respons og anti-interferens. Figur 8.13 viser et bilde av sensoren og Figur I.1 viser sensorens spesifikasjoner. Se også datablad Vedlegg I.



Figur 8.13.: DHT11 temperatur og fuktighet sensor.

8.2.7. Systemstruktur for temperatur/fuktighet måling

Blokkskjemaet i Figur 8.14 viser systemstrukturen for delsystemet for temperatur/fuktighet måling. DHT11 sensoren leser av både temperatur og fuktighet i rommet, denne informasjonen sendes videre til mikrokontrolleren som konverterer de elektriske signalene om til temperatur målt i °C og fuktighet målt i % som skrives ut til displayet.

Figur 8.14.: Delsystemet for måling av temperatur og fuktighet.

8.3. Styringssystem for vindhastighet

Denne seksjonen er basert på teori fra [9]. Hensikten med denne seksjonen er å dekke den nødvendige bakgrunnsteorien som kreves for å designe et styringssystem som er et av kravene til vindtunnelen.

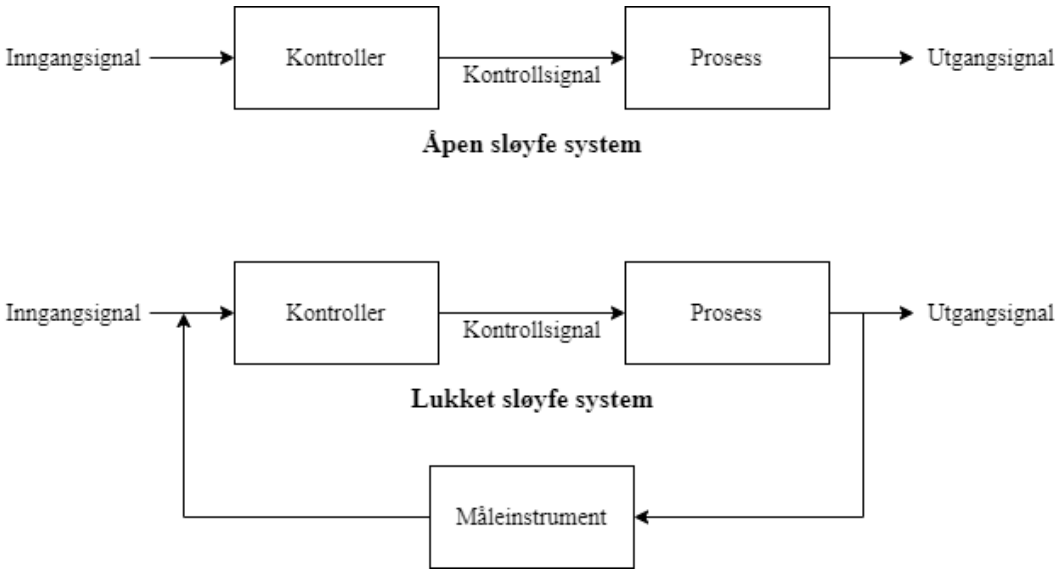
8.3.1. Introduksjon

Ifølge krav K.VT 04 så skal brukeren kunne stille inn vindhastigheten selv. For å oppfylle dette kravet skal vi implementere et digitalt lukket sløyfe reguleringssystem som automatisk justerer rotasjonshastigheten på AC vifta til ønsket vindhastighet er oppnådd. Vindtunnelen har fra før av kun manuell styring via en bryter man vrir på. Denne bryteren fungerer som et analogt potmeter som regulerer motstanden i kretsen som tilfører motordriver kortet spenning. Den tilførte spenningen kan sees på som et analogt inngangssignal som kan variere mellom 0–10 Volt som er proporsjonalt med den tilførte spenningen til AC vifta som kan variere mellom 0–400 Volt.

Motordriver kortet oversetter inngangssignalet til et kontrollsignal som da er driftsspenningen til AC motoren, når 0 Volt inngangssignal er tilført til kortet, så operer vifta på 0 %, når 10 Volt er tilført så opererer vifta på 100 % maks ytelse. Det eneste som trengs å endres på det eksisterende styringssystemet er pot-meteret som skal erstattes med et digitalt styringssystem som utfører samme jobb som pot-meteret, nemlig å variere kontrollsignalet basert på ønsket vindhastighet.

8.3.2. Lukket sløyfe styringssystem

For styringssystemer så skiller man mellom åpent og lukket sløyfe system. Et åpent sløyfe-styringssystem tar imot et inngangssignal som konverteres til et kontrollsignal som påvirker en prosess som produserer et utgangssignal. I et åpent system så får ikke kontrolleren mulighet til å gjøre automatiske justeringer hvis utgangssignalet viser seg å være feil. I et lukket sløyfe system derimot, så måles utgangssignalet som mates så tilbake til kontrolleren som gjør de nødvendige justeringene til vi oppnår ønsket resultat. Et lukket sløyfe styringssystem jobber utifra en referanse, i vårt tilfelle blir denne referansen vindhastigheten som brukeren har stilt inn. Kontrolleren tar å sammenligner utgangssignalet med referansen, er utgangssignalet for lavt så vil kontrolleren gi beskjed til prosessen om å øke pådraget, i motsatt tilfelle vil prosessen minke pådraget. Styringssystemet vil gjenta denne prosessen med å øke og minke pådraget helt fram til vi har oppnådd en stabil utgangsverdi som er så nærme referansen som mulig. Figur 8.15 illustrerer forskjellen på et åpent og lukket sløyfe system.



Figur 8.15.: Åpent og lukket sløyfe system.

8.3.3. Ytelsesterminologi for styringssystemer

For å kunne bedømme oppførselen til et styringssystem så er det noen nøkkelterminologi begreper man må forstå. Man tester som regel systemets oppførsel ved å sende inn et "step-signal" på inngangen som foreksempel går fra 0 til 1 Volt, så ser man på utgangen hvordan systemet oppfører seg, hva som skjer på utgangen kalles for "step-respons". Figur 8.16 viser et grafisk eksempel på step-responsen til et typisk lukket sløyfe styringssystem.

Rise time

Dette er tiden det tar for output verdien å nå en spesifikk prosentandel av steady-state verdien. Stigningstiden måles gjerne i intervallet 10-95% av steady-state verdien.

Settling time

Dette er tiden det tar for output verdien å nå steady-state verdi med 2-5 % avvik.

Overshoot

Overshoot beskriver hvor mye output verdien overstiger steady-state verdien før den stabiliserer seg og måles i % relativt til steady-state verdien.

Peak

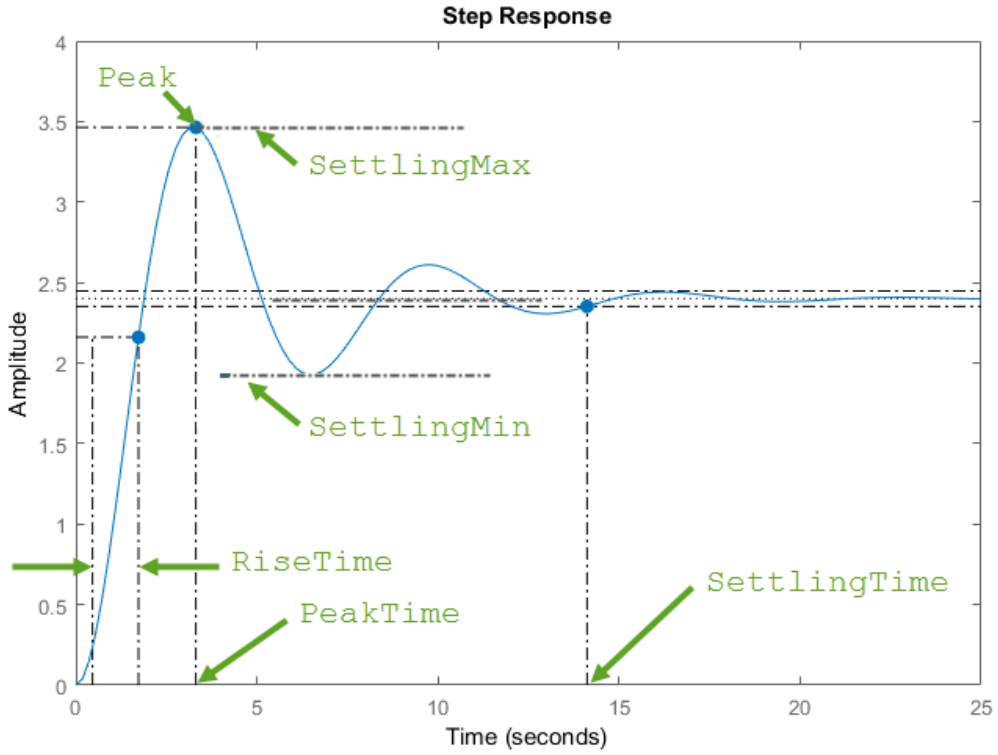
Peak er maks-verdien til step-responsen.

Error

Error er differansen mellom referanse (r) og output (y) verdi og kan beskrives med formelen:

$$e = r - y \tag{8.10}$$

Lukket sløyfe systemet jobber iterativt med å forsøke å holde error verdien så nærme 0 som mulig for å oppnå steady-state tilstand.



Figur 8.16.: Step-respons for lukket sløyfe-styringssystem [19].

8.3.4. PID regulering

Den mest vanlige algoritmen for å kontrollere en motor eller vifte er å bruke en kontroller som er basert på PID regulering. En PID regulator består av tre ledd som avgjør pådraget. P står for proporsjonalitet, I står for integrasjon og D står for derivasjon. Hvert ledd kan beskrives som:

P: Dette leddet sørger for å øke eller minke pådraget til prosessen avhengig av om error verdien er positiv eller negativ.

I: Dette leddet sjekker tidligere verdier av error verdien og integrerer dem over tid for å eliminere eventuelle avvik på utgangen som følge av P leddet.

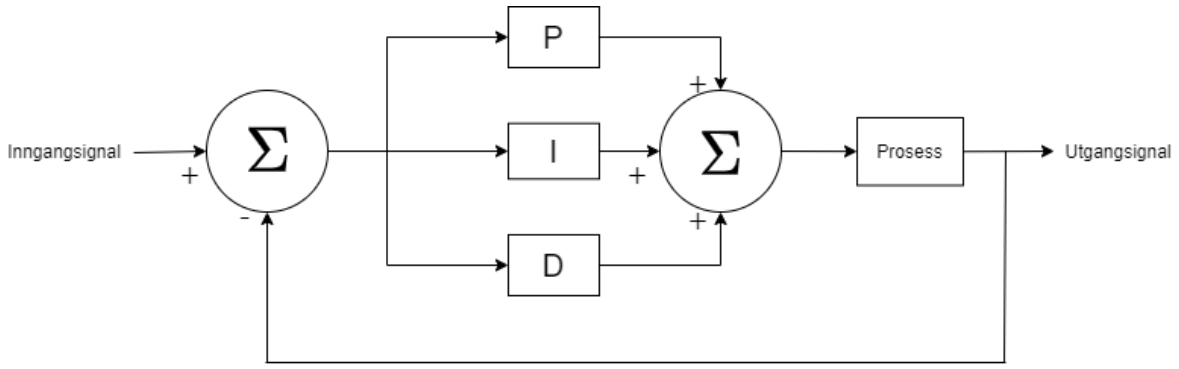
D: Dette leddet forsøker å lage et estimat av fremtidige verdier for error signalet basert på nåværende endringer. Målet er å forsøke å dempe uønskede bråe endringer (overshoot) fra I-leddet når output verdien nærmer seg ønsket referanse verdi.

En PID regulator kan beskrives matematisk med formelen:

$$u(t) = Kp \cdot e(t) + Ki \int_0^t e(t)dt + Kd \cdot \frac{d}{dt}e(t) \tag{8.11}$$

Der Kp, Ki og Kd er tuning koeffisientene til PID regulatoren. Disse koeffisientene kan man justere selv avhengig av hvordan respons man ønsker å oppnå med PID regulatoren. I Matlab/Simulink kan man for eksempel modellere et reguleringssystem og automatisk finjustere disse koeffisientene. I et reguleringssystem er kanskje kravet at man ønsker rask responstid, det fører som regel til høy overshoot. I et annet tilfelle ønsker man kanskje lav overshoot, det fører som regel til en tregere responstid. Det er dette som gjør PID regulatorer til et kraftig verktøy, det gir designeren fleksibilitet til å endre responsen til styringssystemet avhengig av hva kravene er. Man kan også selv eksperimentere med å manuelt teste ut forskjellige kombinasjoner av

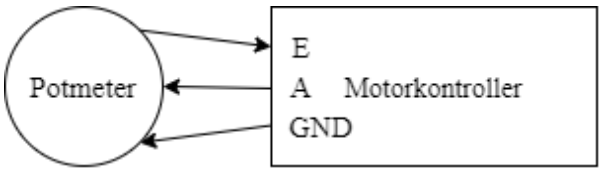
koeffisientene til man oppnår ønsket respons. Figur 8.17 illustrerer hvordan PID-regulatoren kobles i en reguleringssløyfe.



Figur 8.17.: Lukket sløyfe reguleringssystem med PID-regulator.

8.4. Design av styringssystem

I vindtunnelen står det en ZIEHL-ABEGG-FC040-4DQ vifte som roterer ved hjelp av en elektrisk 3-fas 400 Volt AC motor. Motoren styres ved hjelp av motorkontroller kortet ZIEHL-ABEGG Dcontrol PKDT5 som har et potmeter tilkoblet. Potmeteret har 3 terminaler som er tilkoblet E, G og A portene på motorkontroller kortet som vist på Figur 8.18. A tilfører potmeteret 10 Volt DC spenning, G er jord og E tar imot inngangspenningen fra potmeteret. Potmeteret fungerer som en spenningsdeler, når man vrir på bryteren med klokken, så vil motstanden i potmeteret minke slik at spenningen som E inngangen tar imot øker gradvis. Spenningen tilført E kan variere mellom 0-10 Volt.



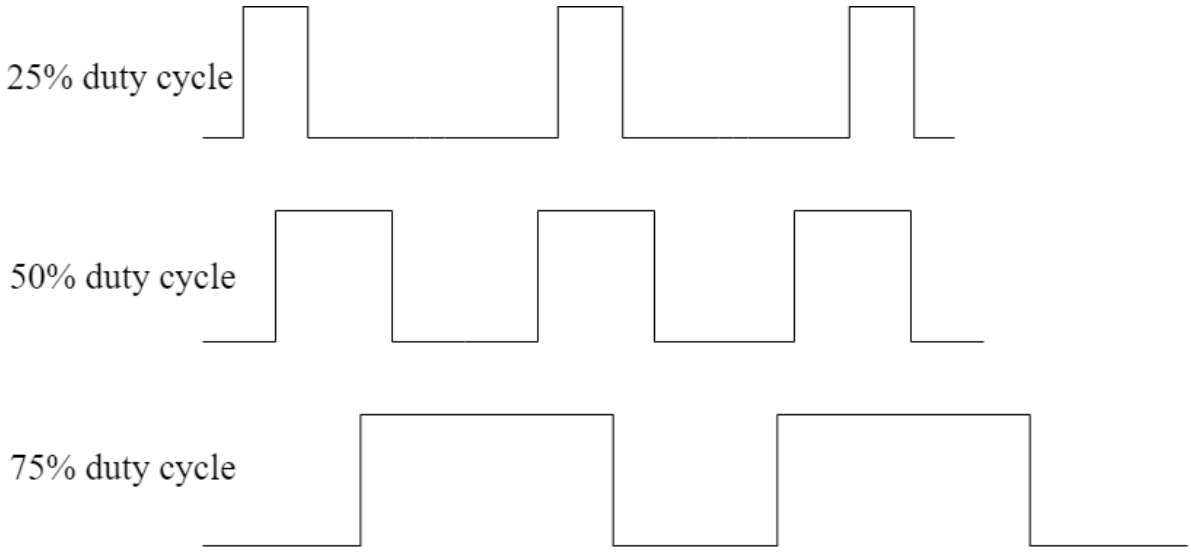
Figur 8.18.: Analogt potmeter tilkoblet motorkontrolleren.

Dette potmeteret skal erstattes med et digitalt styringssystem som utfører samme jobben som det fysiske potmeteret. Dette kan oppnås ved å erstatte potmeteret med en mikrokontroller som sender inn en variabel spenning ved hjelp av puls-bredde modulasjon (PWM) til motorkontroller kortet. Ved bruk av mikrokontroller så er det to utfordringer. For det første så kan ikke Arduino Atmega2560 kortet sende ut mer enn 0-5 Volt ved hjelp av PWM, for det andre så må PWM-signalet konverteres til et rent analogt signal for at motorkontroller kortet skal kunne klare å lese av signalet.

8.4.1. Pulsbreddemodulasjon

Pulsbreddemodulasjon er en teknikk for å få en tilnærmet analog output ved hjelp av digitalteknikk. Arduino ATmega2560 kortet har egne porter med PWM funksjonalitet. PWM skrur

et signal av og på veldig raskt slik at gjenommsnittspenningen er det man får ut. Hvor lenge signalet er på definerer pulsbredden. Duty cycle definerer av/på forholdet til signalet målt i % (se Figur 8.19). Har man en PWM spenning på 5 Volt med 50 % duty cycle så får man ut 2.5 Volt analog spenning.



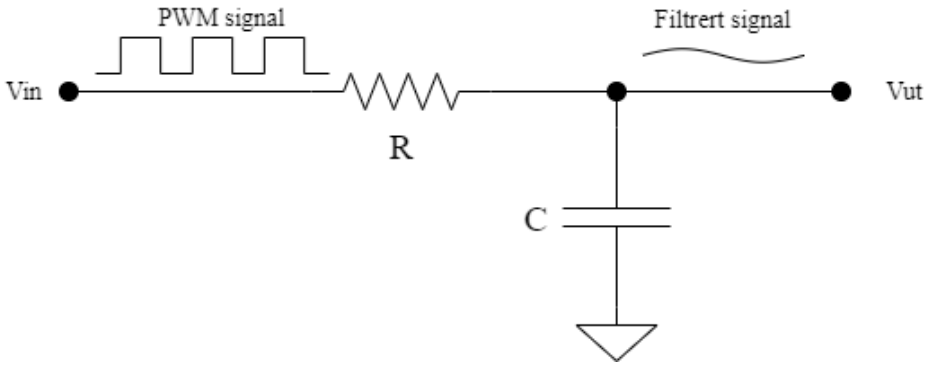
Figur 8.19.: Duty cycles.

8.4.2. Design av DAC

Dette delkapittelet er basert på kunnskap tilegnet fra [7].

For å løse problematikken angående pulsbreddemodulasjon så er man nødt til å designe en krets som består av to elementer; et RC lavpass filter som består av en resistor og kondensator. Og en ikke-inverterende operasjonsforsterker. Denne kretsen kan sees på som en digital til analog konverter (DAC).

Hensikten med RC lavpass filteret er å filtrere bort uønskede signaler med høye frekvenser og bare ta vare på lavfrekvens signalene slik at vi får et rent analogt signal på utgangen. Figur 8.20



Figur 8.20.: RC lavpass filter.

illustrerer hvordan et første ordens RC lavpass filter er koblet og hva som skjer med PWM signalet etter det har passert lavpass filteret.

Valg for kombinasjon av motstand og kondensator verdier har en direkte sammenheng med filterets respons. Tidskontanten τ definerer hvor raskt kondensatoren lader seg opp gjennom motstanden til 63.2 % av tilført spenning. Vi kan finne tidskonstanten ved hjelp av formelen:

$$\tau = RC \tag{8.12}$$

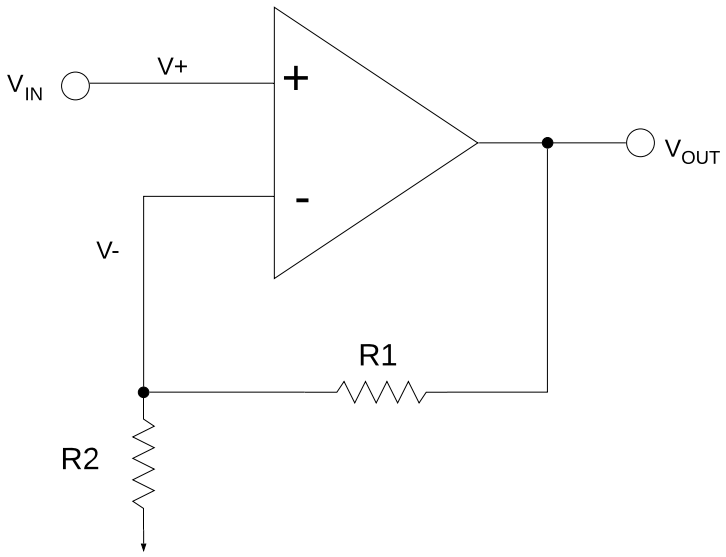
Siden vi ønsker å ønsker å kun ha med signaler som har frekvenser så nærme 0 Hertz (Hz) som mulig, så ønsker vi en veldig lav knekkfrekvens på filteret. Knekkfrekvensen er definert som grenseverdien målt i Hz der signaler som overstiger denne grensen blir dempet og filtrert bort. En annen måte å se på knekkfrekvensen på, er å se på det som båndbredden til filteret. I et første ordens filter som vi skal bruke, så blir signaler som ligger over knekkfrekvensen dempet med -20 dB per dekade. Vi kan finne knekkfrevensen til RC lavpass filteret med formelen:

$$F_c = \frac{1}{2\pi\tau} = \frac{1}{2\pi RC} \tag{8.13}$$

Etter å ha eksperimentert med forskjellige motstand og kondensator verdier så kom vi fram til et fornuftig resultat ved å velge R = 1000 Ω og C = 100 µF. Dette gir oss følgende knekkfrekvens:

$$F_c = \frac{1}{2\pi \cdot 1000\,\Omega \cdot 100\,\mu\text{F}} = 1.59\,\text{Hz} \tag{8.14}$$

Nå har vi et filter til å fjerne de høye frekvensene fra PWM signalet som blir sendt til mikrokontrolleren, men vi trenger fremdeles å kunne forsterke det analoge signalet etter filteret. Vi bruker en ikke-inverterende operasjonsforsterker. Det finnes mange forskjellige modeller av operasjonsforsterkere, til vår krets bruker vi en LM358 OP-amp som kobles slik som vist på Figur 8.21. Figur K.1 viser utdrag fra databladet til OP-ampen. Se også vedlegg for elektriske egenskaper til LM358 Figur K.1.



Figur 8.21.: Kobling av ikke-inverterende OP-amp.

Siden mikrokontrolleren bare kan operere med 0-5 Volt, mens motorkontroller kortet operer med 0-10 Volt, så ønsker vi en forsterkningsfaktor for OP-ampen lik 2. Forsterkningsfaktoren

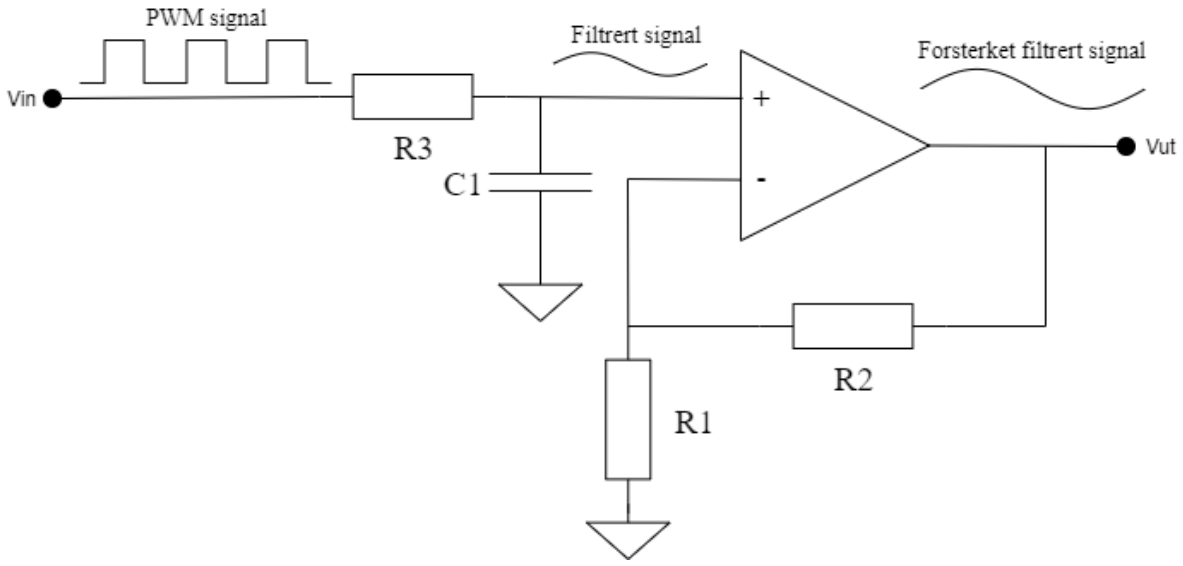
for en ikke-invertende OP-amp kan beregnes med formelen:

$$\text{Gain} = 1 + \frac{R2}{R1} \tag{8.15}$$

Vi setter både R1 og R2 lik 10 kΩ og får en forsterkning lik:

$$\text{Gain} = 1 + \frac{10\,\text{k}\Omega}{10\,\text{k}\Omega} = 1 + 1 = 2 \tag{8.16}$$

For at OP-ampen skal kunne gi en forsterkning så er den avhengig av en ekstern strømforsyning som bør ligge litt over maks spenningen vi ønsker, så en strømforsyning på 12 Volt holder. Filteret og forsterkerkretsen kobles sammen slik som vist på Figur 8.22.

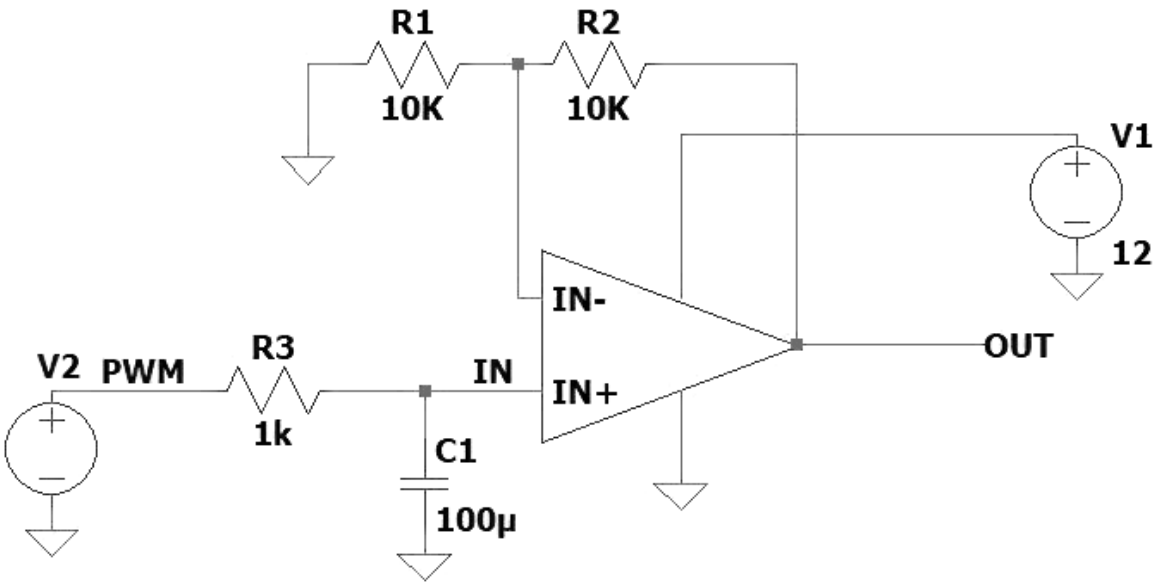


Figur 8.22.: Kombinert filter og forsterker krets.

Denne kretsen kan sees på som et bindeledd mellom mikrokontrolleren og motorkontroller kortet der Vin er PWM signalet fra mikrokontrolleren og Vut er kontrollsignalet som sendes til motorkontroller kortet. Tilsammen utgjør denne kretsen det vi kaller for Digital til Analog konverter (DAC).

8.4.3. Kretssimulering av DAC

Figur 8.23 er tegnet i kretssimuleringsprogrammet LTspice og viser fullstendig krets for DAC. Før vi begynner med bestilling av komponenter og fysisk implementering, så simulerer vi kretsen for å sjekke at vi får ønskede resultater.

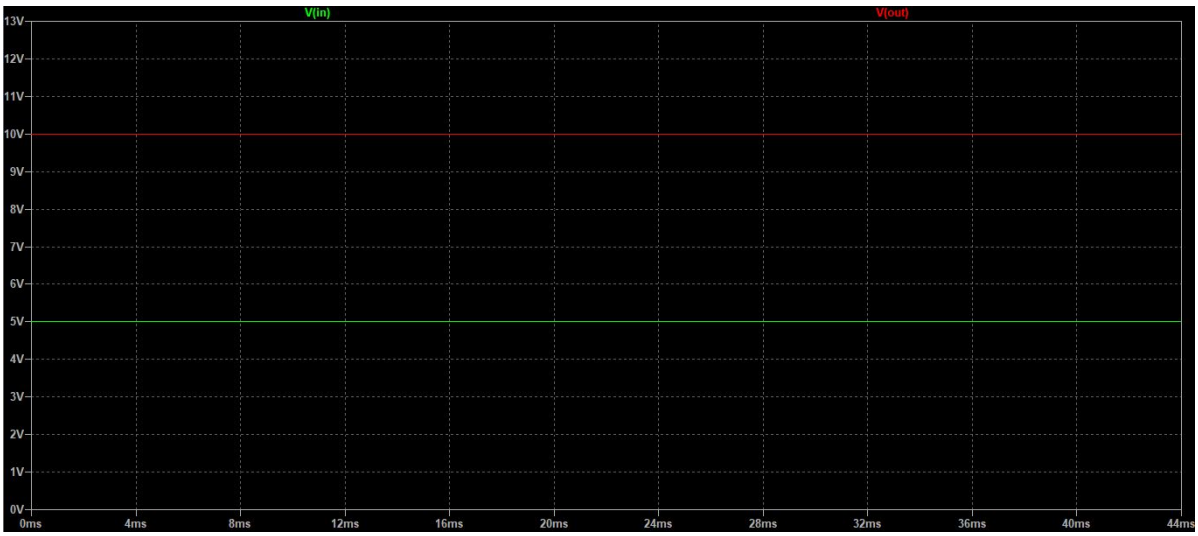


Figur 8.23.: Kretstegning for DAC i LTspice.

Gain simulering

Setter inngangsspenning lik 5 Volt DC og sjekker at dette signalet forsterkes med en faktor på 2. Resultatet er som vist på Figur 8.24. På grafen kan vi se at vi får en fordobling av inngangsspenningen på utgangen, og kan dermed verifisere at forsterkerkretsen virker som den skal.

Nå endrer vi inngangsspenningen til å etterligne PWM signal fra Arduino Atmega2560. Som standard så opererer denne mikrokontrolleren med PWM signal på 490 Hz, ved å regne ut den



Figur 8.24.: Spenningsforsterkning.

inverse finner vi tidsperioden for en syklus:

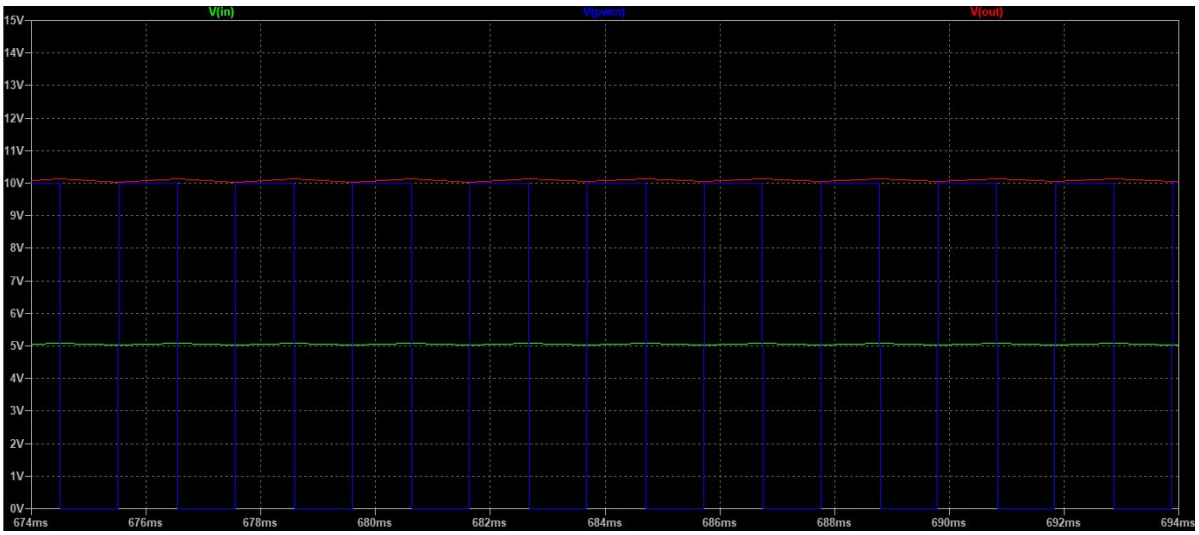
$$Tp = \frac{1}{490\,\text{Hz}} = 0.002\,\text{s} \tag{8.17}$$

Med en duty cycle på 50 % sier vi at signalet skal være på halvparten av perioden, det tilsvarer 0.001s.

Setter PWM signalet med en initial verdi på 0 Volt og en peak verdi på 10 Volt, sender så signalet gjennom kretsen og får følgende resultater som vist på Figur 8.25. Der den blå linjen representerer PWM signalet fra mikrokontrolleren, den grønne linjen representerer det filtrerte PWM signalet og den røde linjen representerer det forsterkede filtrerte PWM signalet.

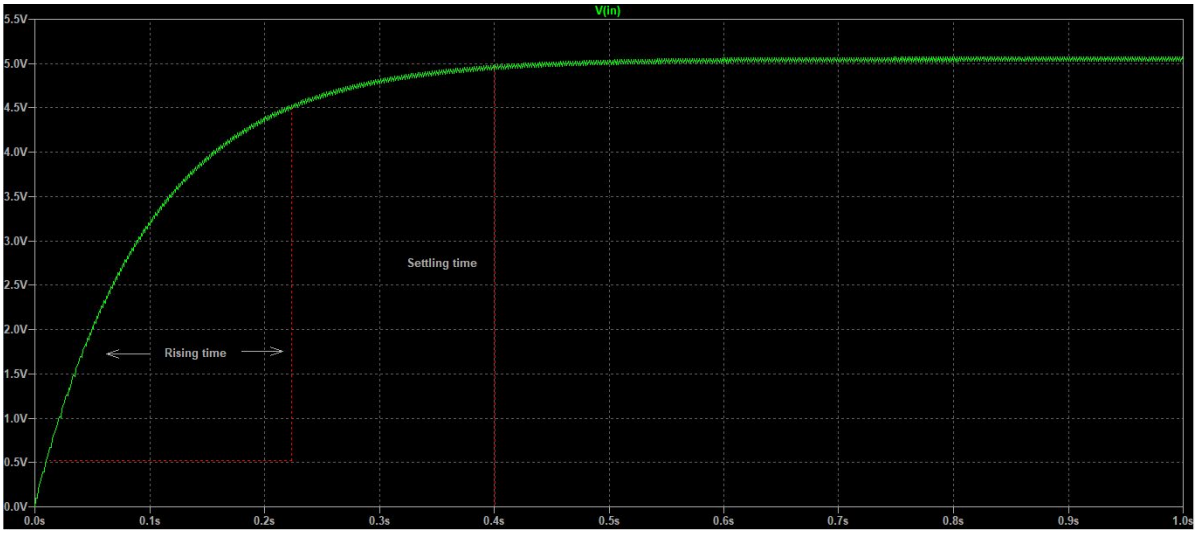
Filter respons

Måler over utgangen av RC filteret og kjører en transientanalyse i tidsdomenet for å simulere sprangresponsen til filteret som vist på Figur 8.26. Grafen viser en stigningstid på ca. 210 ms



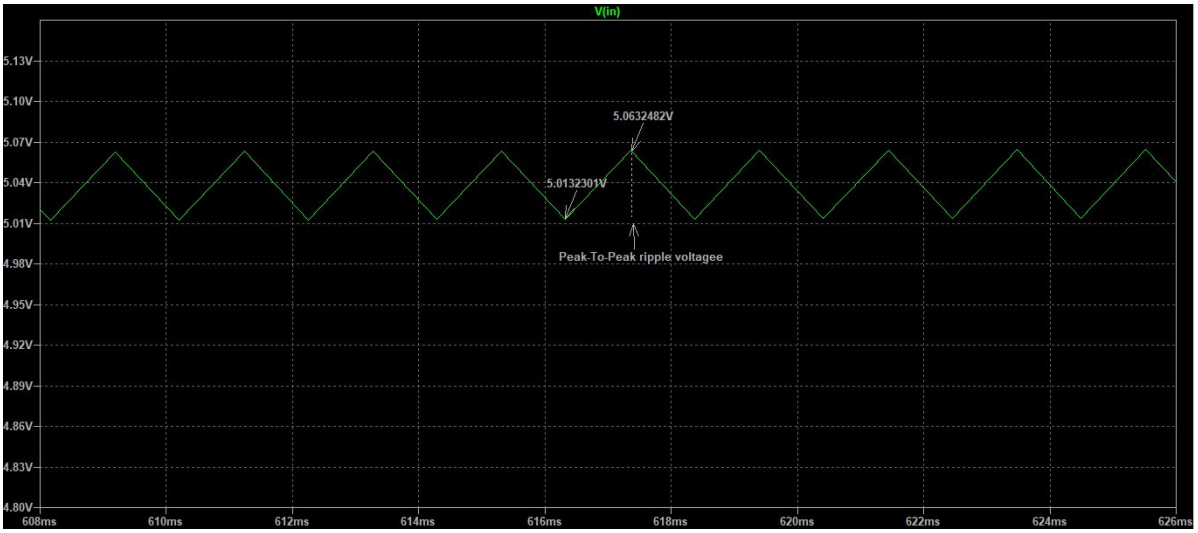
Figur 8.25.: Spenningsforsterkning med PWM signal.

og en settling time på 400 ms.



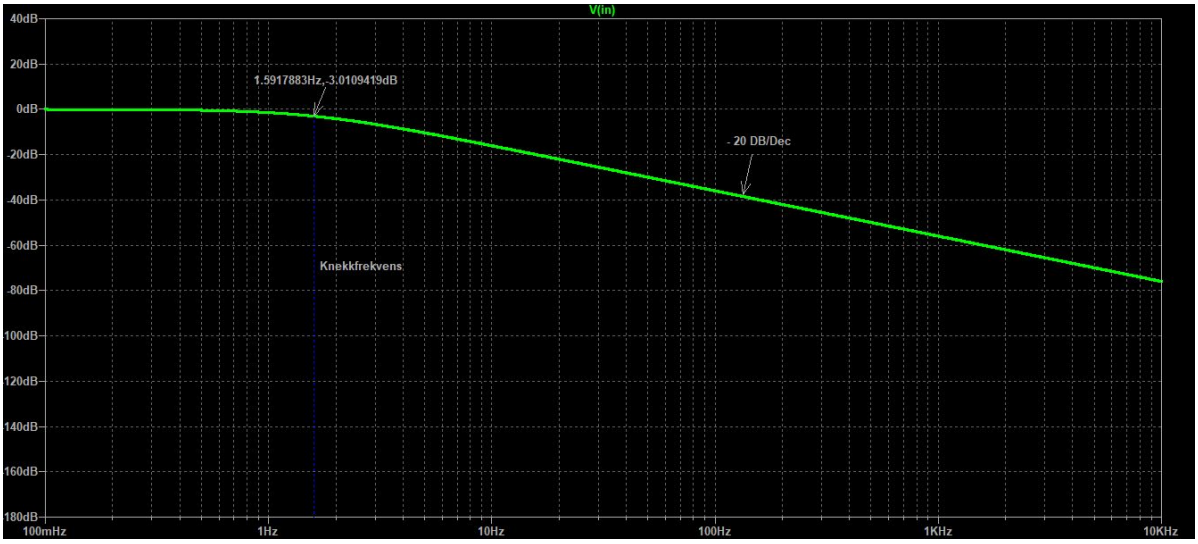
Figur 8.26.: Filterets sprangrespons.

Hvis vi zoomer inn på grafen så ser vi at vi har noen små variasjoner i form av ripple-spenning som vist på Figur 8.27, dette er helt normalt da RC filteret ikke er et ideelt filter. I de fleste tilfeller så godtar man en ripple-spenning under 100 mV, i vårt tilfelle har vi en Peak-To-Peak ripples-penning på rundt 50 mV.



Figur 8.27.: Ripple-spenning.

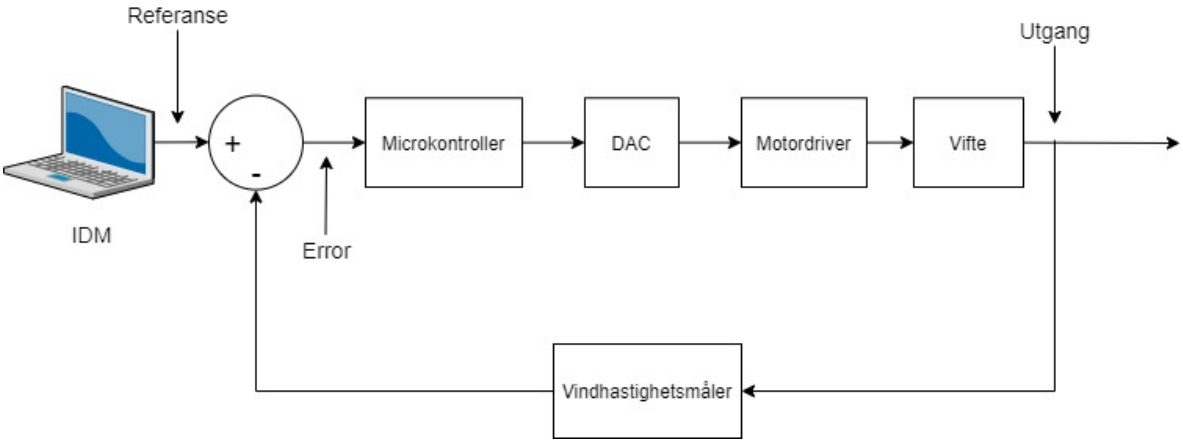
Nå går vi over til frekvensdomenet og simulerer frekvensresponsen til filteret. Vi sender inn et sinusformet AC signal med en amplitude på 1 Volt med mikrokontrollerens frekvens på 490 Hz og verifiser at den resulterende knekkfrekvensen fra simuleringen, stemmer overens med den beregnede knekkfrekvensen. Resultatet ser vi på bode-diagrammet som vist på Figur 8.28. Her ser vi at knekkfrekvensen stemmer veldig godt overens med den beregnede frekvensen, og at vi har en dempning på -20 dB/dekade som er forventet fra et 1. ordens lavpass filter.



Figur 8.28.: Frekvensresponsen til lavpass filteret.

8.4.4. Digitalisering av lukket sløyfe styringssystem

Nå som vi har lagt tilrette for at AC-motoren til vifta kan styres fra mikrokontrolleren gjenstår det å designe selve styringssystemet. Styringssystemet skal oppfylle den funksjonen som gir brukeren mulighet å taste inn ønsket vindhastighet i m/s innenfor viftens min/maks hastighet. Ønsket vindhastighet tastes inn på IDM av bruker. Inntastet hastighetsverdi skal så sendes videre til mikrokontrolleren som definerer denne verdien som referansen til lukket sløyfe styringssystemet. Styringssystemet sammenligner deretter referanseverdien med utgangsverdien som da er målt vindhastighet fra differensialtrykksensoren som sender samplet måledata inn til mikrokontrolleren i et fast tidsintervall. I starten vil vifta stå i ro og sensoren måler dermed 0 vindhastighet. Lukket sløyfe-styringssystemet vil derfor øke pådraget til målt vindhastighet på utgangen er så lik referanseverdien som mulig og vil forsøke holde seg stabilt der ved å holde error-verdien så nærme 0 som mulig. Figur 8.29 viser blokkdiagram for det digitale styringssystemet.



Figur 8.29.: Digitalt lukket sløyfe styringssystem for vindhastighet.

8.4.5. Valg av reguleringsalgoritme

All styringslogikk utføres av et dataprogram som blir lastet inn på mikrokontrolleren. I forrige seksjon var vi inne på PID-regulering. For mikrokontrollere finnes det egne programvarebiblioteker som etterligner PID-regulatorer som man kan ta i bruk. PID-biblioteket linkes til hovedprogrammet hvor man definerer ønsket hastighet som da blir referansen, inngangssignalet defineres som målt vindhastighet og utgangssignalet defineres som pådraget. Pådraget i vårt tilfelle blir PWM-signalet som mikrokontrolleren sender til motordriveren som igjen tar å øker eller minker vindhastigheten til den matcher referanseverdien så nært som mulig.

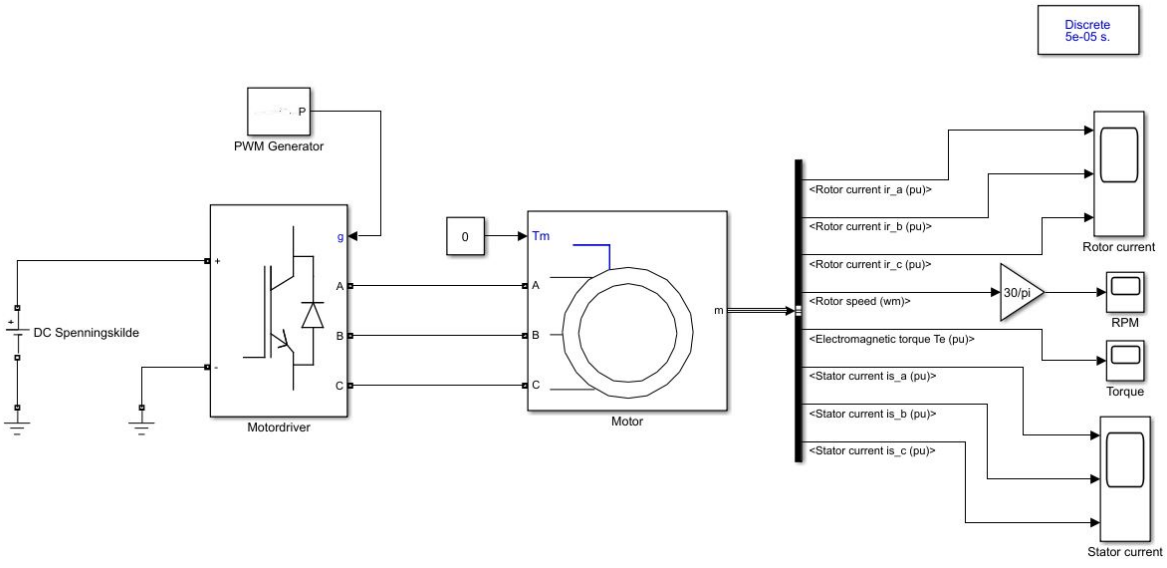
En utfordring for oss er at vifta styres av en AC-motor som vi ikke har funnet de nødvendige motordataene på. Uten disse dataene er vi ikke i stand til å gjøre en eksakt matematisk simulering i Matlab/Simulink. Hadde vi hatt disse dataene kunne vi brukt dem i Simulink for å simulere motorens oppførsel. Vi kunne også brukt ett innebygd "PID-tuning verktøy" på PID-kontrolleren, og fått ut nøyaktig de PID-parameterne vi trenger for å få en fornuftig steprespons. En AC motor er uansett mye mer kompleks å simulere enn en DC motor, så hvis vi skal ta i bruk PID-regulering så er vi nødt til å eksperimentere med forskjellige kombinasjoner av PID-parameterne til vi oppnår en fornuftig respons. Et annet alternativ er å ikke bruke PID, men heller designe en algoritme som øker pådraget stegvis til ønsket vindhastighet er oppnådd. I vårt prosjekt har vi bestemt oss for å simulere en AC motor som er så tilnærmet lik den virkelige AC motoren og teste oss fram med forskjellige PID-parametere så vi får en bedre forståelse for hvordan systemet vil oppføre seg i virkeligheten når dette skal fysisk implementeres. Vi velger derfor å bruke PID reguleringsalgoritmen i dette prosjektet.

8.5. Simulering av styringssystemet

Hensikten med å simulere styringssystemet er for å få en bedre forståelse for systemets oppførsel når det skal implementeres fysisk for å validere og fullføre krav K.VT 04. Med en slik simulasjon kan vi verifisere om delkonseptet vi har valgt er verdt å investere i. Simuleringen er også bra for å kunne teste og mate systemet med forskjellige inngangsparametere for å optimalisere systemets ytelse og verifisere at systemkravene i teorien kan oppfylles. Vi bruker Matlab/Simulink til å simulere styringssystemet til vindtunnelen.

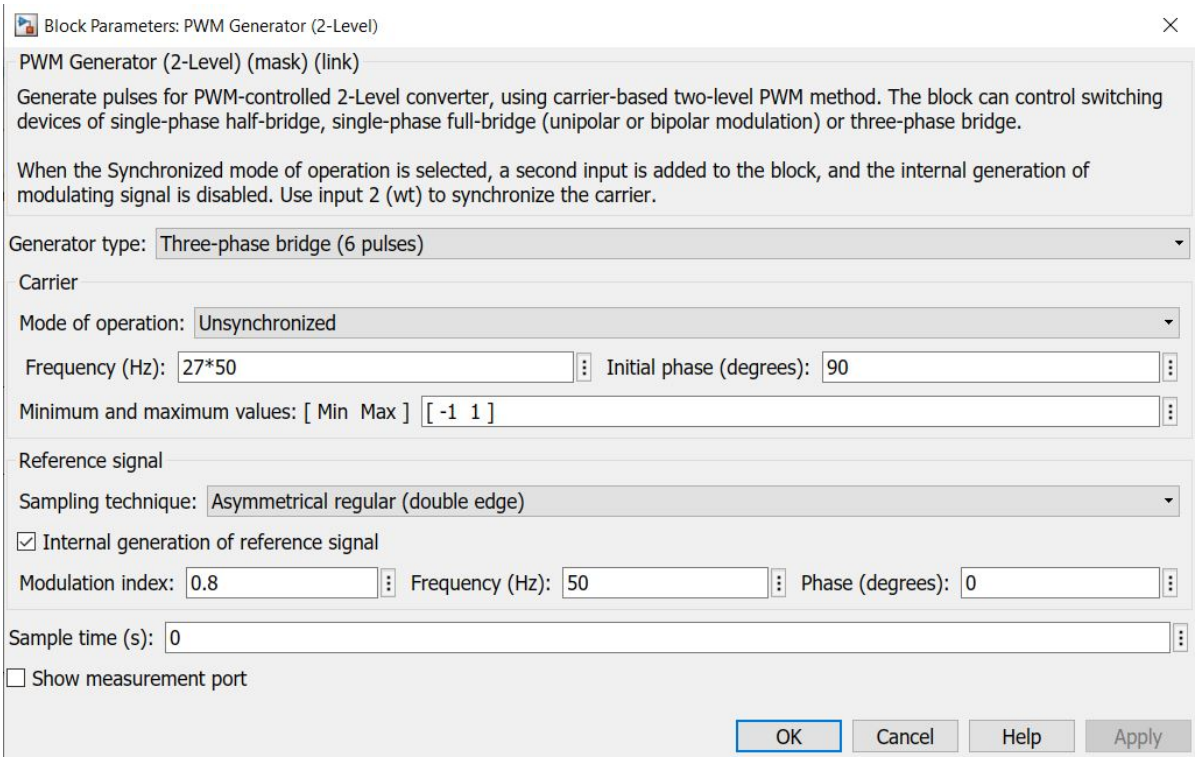
Åpen sløyfe-simulering

Styringssystemet vårt skal ha lukket sløyfe, men man tester alltid systemet med åpen sløyfe først for å sjekke at motoren produserer hastighet ut når vi mater den med spenning inn. Etter vi har verifisert at åpen sløyfe systemet virker så kan vi øke kompleksiteten med lukket sløyfe simulering. Figur 8.30 viser designet av åpen sløyfe styringssystemet og hvordan det er koblet sammen.



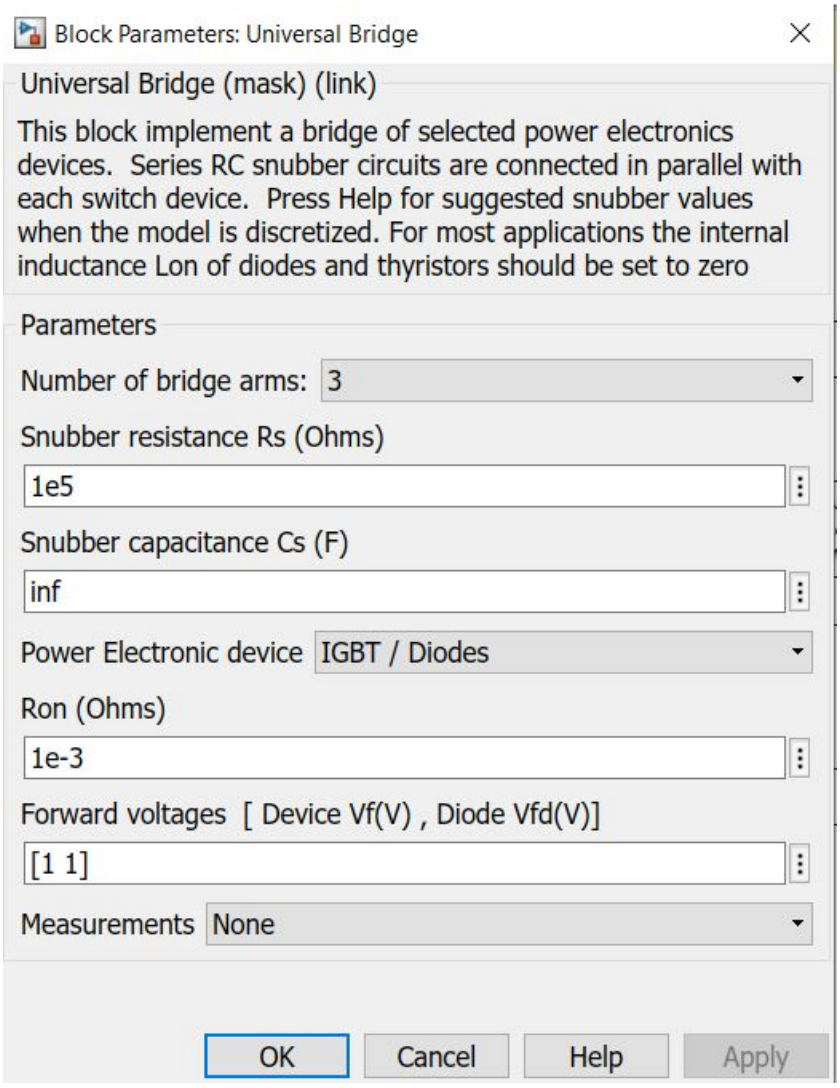
Figur 8.30.: Design av åpen sløyfe styringssystem.

Motordriveren fungerer som en universal bru som konverterer DC inngangsspenning om til 3-fase AC spenning som går til motoren som generer rotasjonshastighet. For å teste at motoren virker så måler vi rotor og stator strømmene som motoren genererer som følge av 3-fase AC spenningen. Vi måler også motorens rotasjonshastighet og torque målt i omdreininger per minutt (RPM). For at motordriveren skal klare å konvertere tilført DC spenning om til 3-fase AC spenning så kobler vi en PWM Generator til den universale broen med følgende standard-innstillinger som vist i Figur 8.31. Vi setter type generator som tre-fase bro og velger intern generering av referansesignal da vi ikke bruker noe eksternt referansesignal for denne komponenten.



Figur 8.31.: Innstillinger for PWM-generatoren.

Figur 8.32 viser standardinnstillingene for den universale broen som opererer på samme måte som det virkelige motordriver-kortet. Vi beholder standardinnstillingene men setter antall bro armer lik 3 og setter Power Electronic device som IGBT / Diodes. Da vil broa operere som en DC til 3-fase AC konverter.

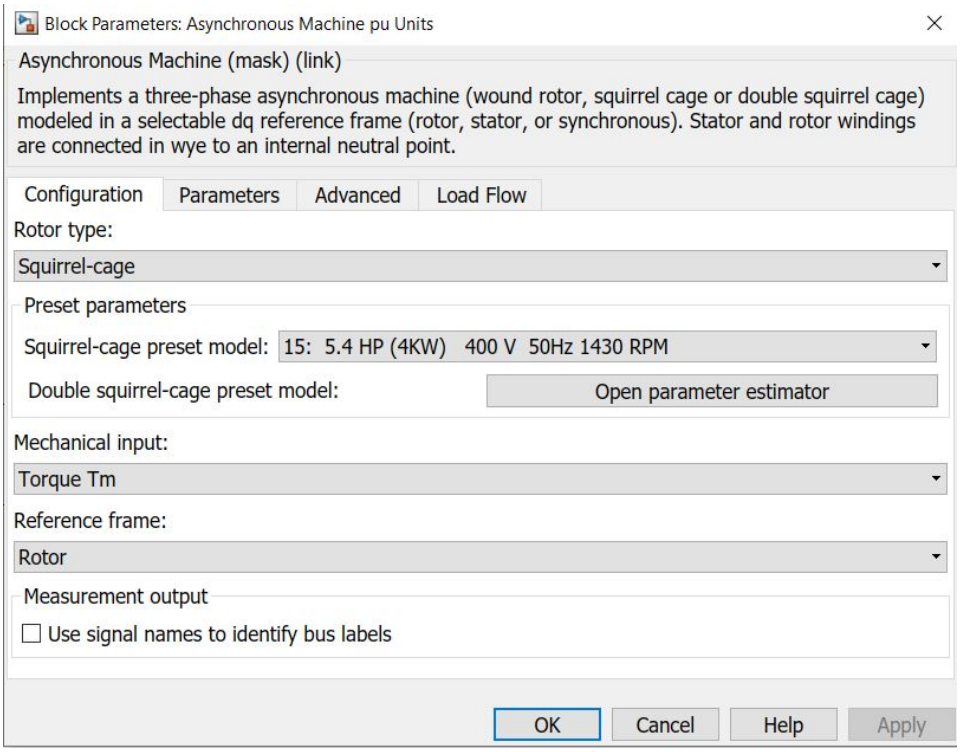


Figur 8.32.: Innstillinger for motordriveren.

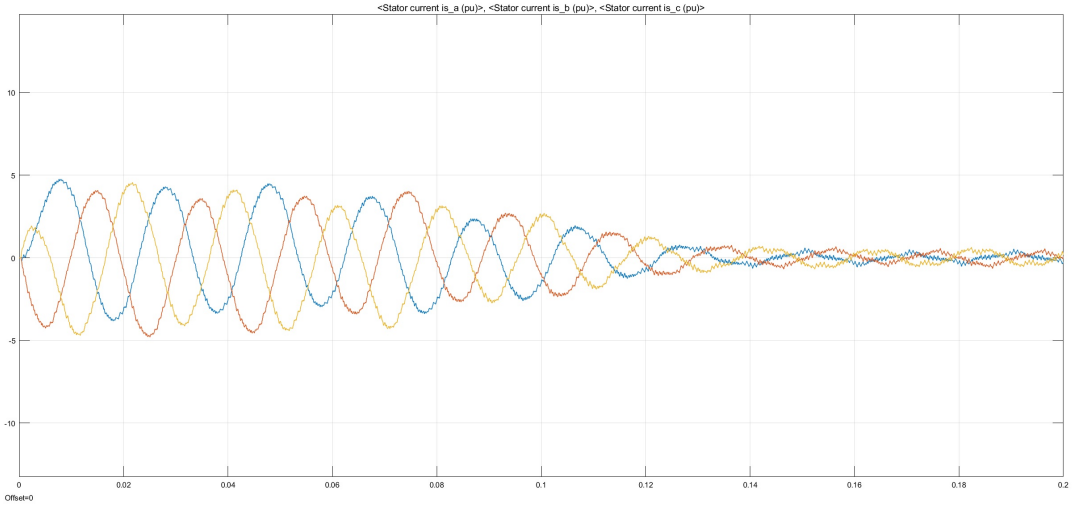
Siden vi ikke har klart å få tak i de nødvendige elektriske og mekaniske parameterne til den eksisterende motoren, og siden det er så knapt med tid igjen til prosjektets deadline, så bruker vi en alternativ motor i disse simuleringene som er det nærmeste vi kommer. Simuleringene for styringssystemet blir derfor ikke en nøyaktig representasjon av den eksisterende motoren, men heller en mer generell simulering for å demonstrere at vårt utvalgte delkonsept for lukket sløyfe-styringssystemet virker og kan anvendes på hvilken som helst motor av samme type. Hadde vi hatt motor parameterne så hadde designet av styringssystemet vært akkurat likt, den eneste endringen vi hadde trengt å gjøre i simuleringen var å endre parameterne på motorblokka lik de oppgitte parameterne på den virkelige motoren. Da kunne vi gjentatt samme prosess med å teste oppførselen med forskjellige parameter-verdier på PID-kontrolleren.

Hvis prosjektet videreføres til en annen prosjektgruppe som klarer å få tak i de riktige motor parameterne, så kan de prøve å kjøre disse simuleringene på nytt med de riktige parameterne. Den eksisterende motoren i vindtunnelen har en oppgitt effekt på 0,59 kW, den nærmeste motoren modellen vi fant i Simulink er på 4 kW men er av samme type. Figur 8.33 viser innstillingene for motoren vi bruker i simuleringen. Denne motoren går på 400 V (50 Hz) akkurat som den virkelige motoren.

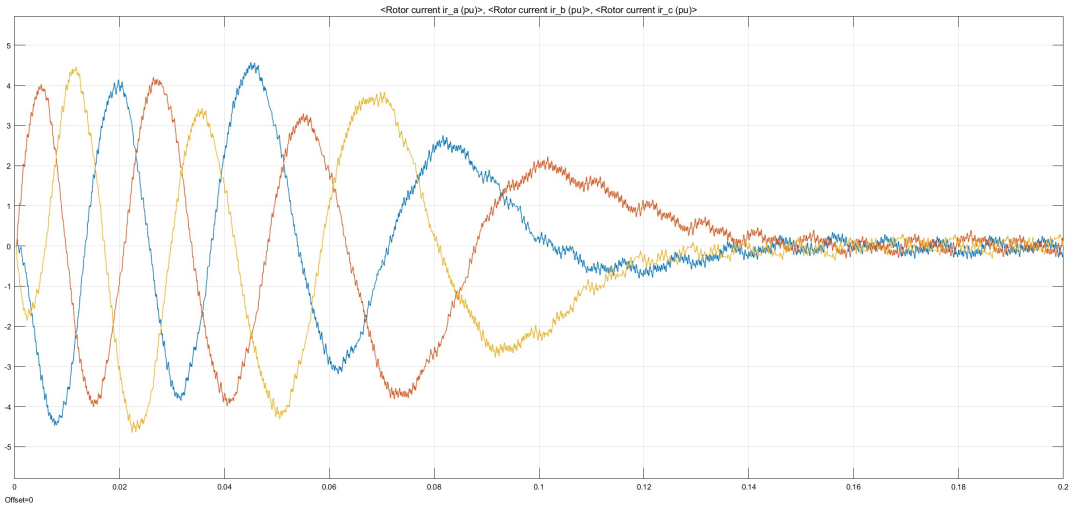
Når alle komponentene er konfigurert og alt er koblet sammen kan vi kjøre simuleringen og sjekke måleresultatene på utgangen av motoren. For å bekrefte at motoren faktisk går på 3-fase vekselstrøm så sjekker vi først måleresultatene for rotor og stator strømmene. Rotor strøm er den strømmen som går gjennom den roterende delen av motoren mens stator strøm er den strømmen går gjennom den statiske delen av motoren. Figur 8.34 viser måleresultatet for stator strømmene og Figur 8.35 viser måleresultatet for rotor strømmene i motoren.



Figur 8.33.: Spesifikasjonene til motor modellen.

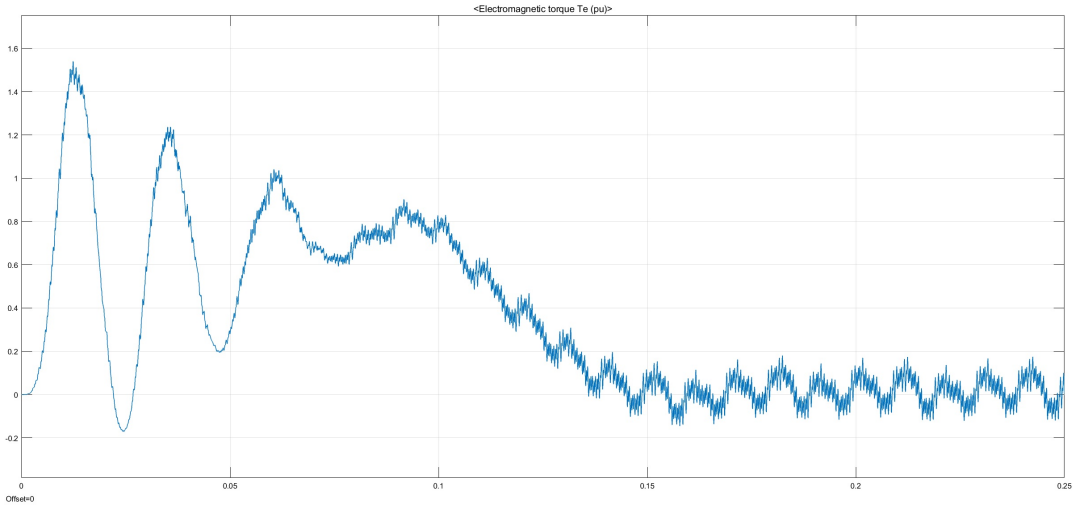


Figur 8.34.: Statorstrømmene til motoren med åpen sløyfe.



Figur 8.35.: Rotorstrømmene til motoren med åpen sløyfe.

Her kan vi verifisere at den simulerte motoren faktisk er en 3-fase AC-motor. Den har tre rotor- og stator-strømmer der hver strøm har en faseforskyvning på 120° iforhold til hverandre. Vi ser også at vi får relativt større startstrømmer i begynnelsen før dem stabiliserer seg på et lavere nivå, noe som gir mening da elektriske motorer er avhengig av en større startstrøm for å kunne generere stor nok elektromagnetisk torque for å få rotoren til å begynne å rotere. Figur 8.36 viser motorens elektromagnetiske torque, her også ser vi at den i begynnelsen er stor før den minker og blir værende på et lavere nivå.

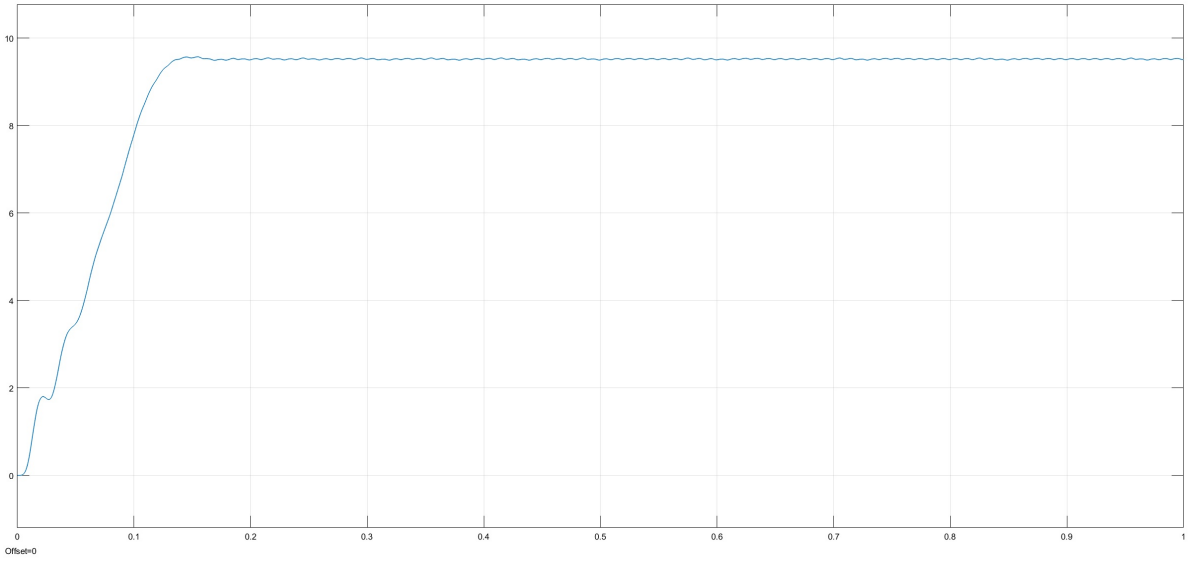


Figur 8.36.: Elektromagnetisk torque med åpen sløyfe.

Nå som vi har verifisert at det er strøm og bevegelse i motoren så ser vi på den siste målingen som er den vi er mest interessert i, nemlig rotasjonshastigheten målt i RPM. Figur 8.37 viser motorens åpen sløyfe hastighet. I realiteten er det jo vindhastigheten målt i m/s vi er interessert i, men siden vi ikke får simulert vindhastighetsmåleren i Simulink så baserer vi oss på motorens RPM isteden. Motorens RPM er jo den rotasjonshastigheten som får viftebladene til å rotere og generere vindhastighet inni vindtunnelen, så de henger sammen uansett.

Med dette verifiserer vi at det simulerte åpen sløyfe- systemet virker som det skal. Vi får rotasjonshastighet ut når vi tilfører DC spenning inn. På grafen ser vi at vi får en rotasjonshastighet

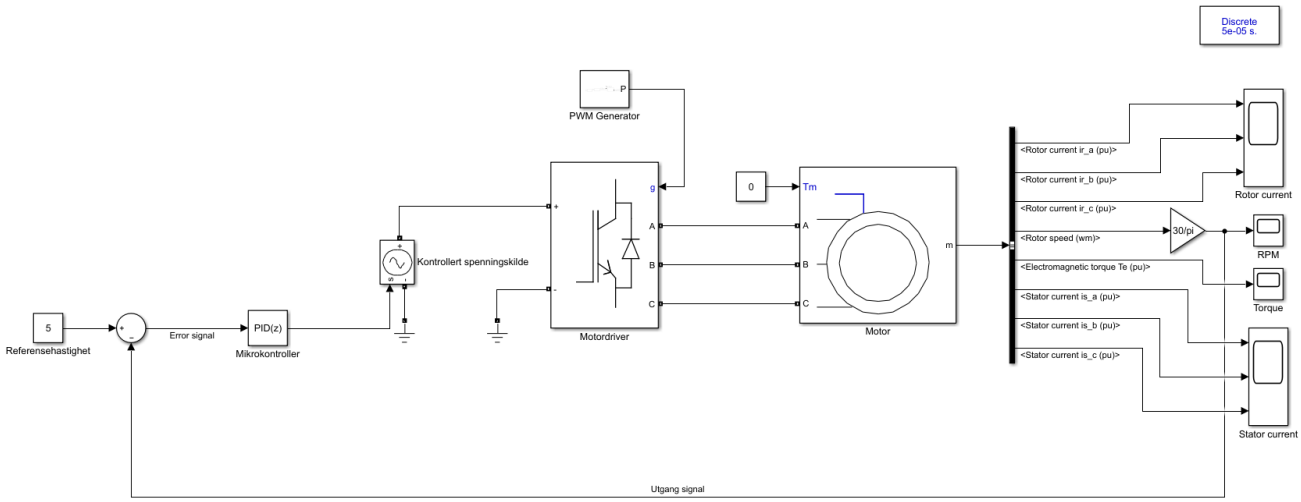
på litt under 10 RPM med tilført maks spenning på 400 volt DC. Den har en rask rise time på ca 0.1 sekunder og veldig lav overshoot. Den eneste feilen jeg kan se i systemet er at motoren burde ha fått en mye større hastighet ut, da den har en makshastighet på 1430 RPM. Koblingene skal være riktige siden de andre grafene viser at motoren oppfører seg som en 3-fase AC-motor skal. Derfor er det mest sannsynlig noe feil i konfigurasjonen av en av blokkene i Simulink-kretsen. I skrivende stund får vi dessverre ikke mer tid til å feilsøke på det tekniske, så dette er jo noe som eventuelt neste prosjektgruppe kan feilsøke videre på. Men systemet virker som det skal, den gir bare ut lavere verdi enn forventet. Med denne simuleringen så beviser vi bak-siden med åpen sløyfe, nemlig at vi har ikke noe kontroll på hva utgangshastigheten blir, det beste man kan gjøre er å manuelt regulere spenningsnivået inn og lese selv av fartsmålingen fram til ønsket resultat er oppnådd. Med lukket sløyfe system eliminerer vi dette problemet, da holder det for brukeren å bare sette ønsket vindhastighet også gjør styringssystemet resten av jobben.



Figur 8.37.: Åpen sløyfe rotasjonshastighet.

8.5.1. Lukket sløyfe simulering

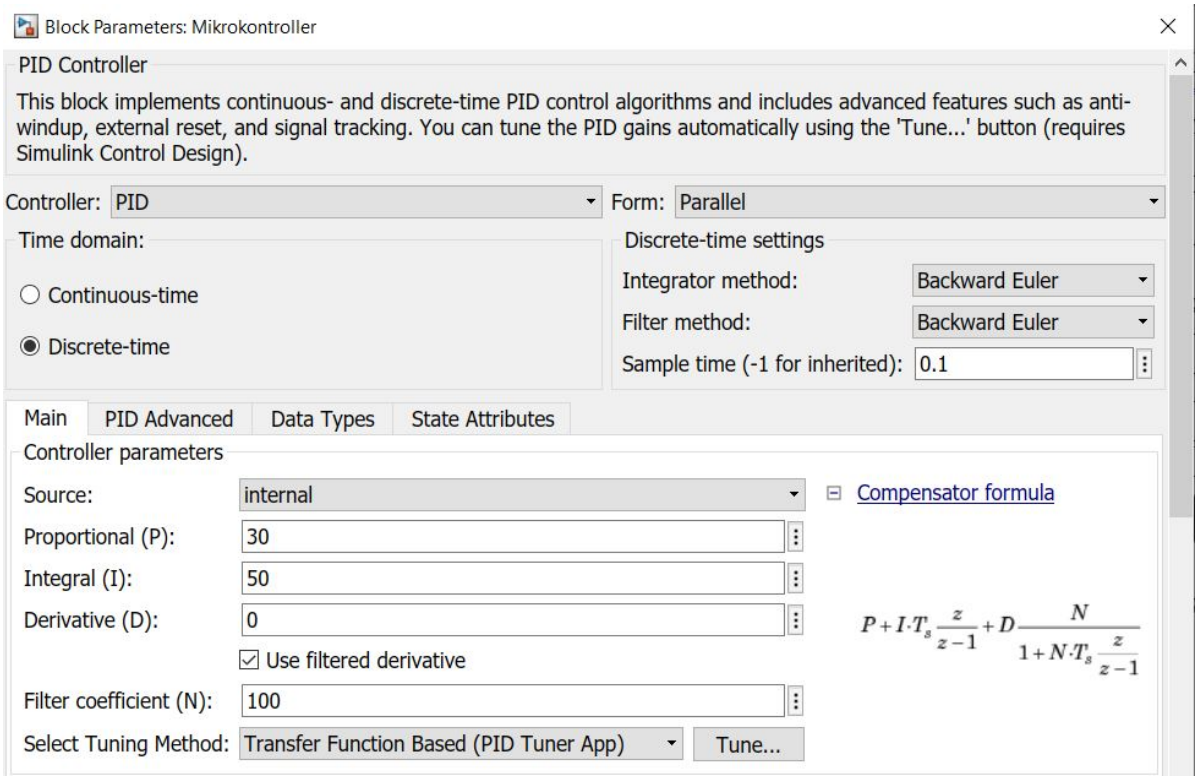
I forrige seksjon bekreftet vi at åpen sløyfe simuleringen fungerer godt nok til å øke kompleksiteten til lukket sløyfe styringssystem som er det vi er ute etter med tanke på kravet. Med lukket sløyfe styringssystem blir kretsen utvidet som vist på Figur 8.38. Her har vi byttet ut spenningskilden med en styrt spenningskilde som vil variere fram til spenningspådraget som sørger for ønsket hastighet er oppnådd. Referansehastighet blokka representerer displayet der brukeren taster inn ønsket hastighet, dette signalet sendes så videre til PID-regulatoren som representerer mikrokontrolleren som beregner spenningspådraget som sendes til motordriveren som styrer motoren. Den målte hastigheten blir så matet tilbake til mikrokontrolleren, dette vil føre til et feilsignal som er differansen mellom referansehastigheten og utgangshastigheten. Dette feilsignalet vil brukes av PID-regulatoren til å beregne et nytt pådrag som sendes til motordriveren, denne prosessen vil gjenta seg raskt gjentatte ganger i en loop fram til motoren har oppnådd en hastighet som er så lik referansehastigheten som mulig.



Figur 8.38.: Lukket sløyfe-styringssystem.

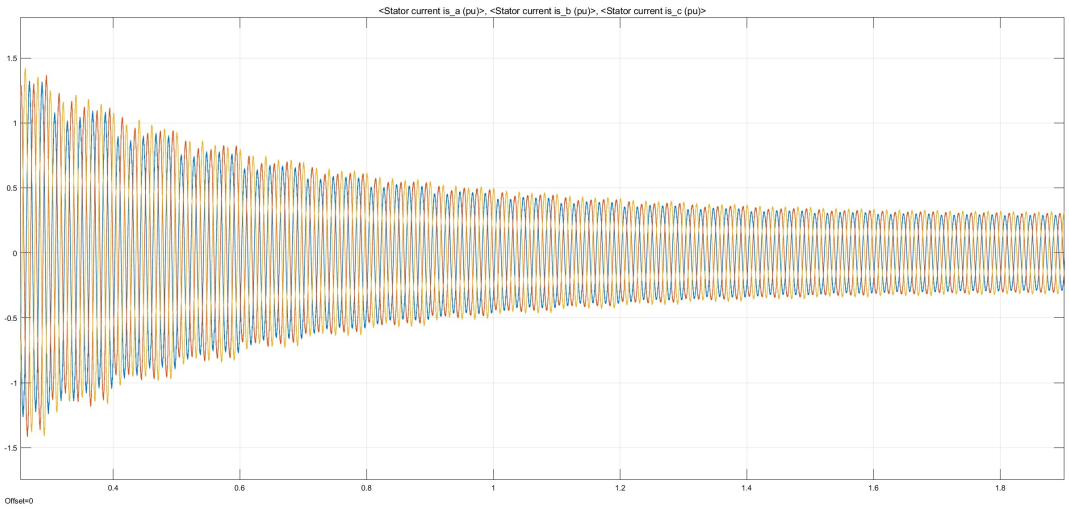
I PID-regulator blokka prøvde vi oss frem med ulike kombinasjoner av P, I og D leddene til vi fikk en fornuftig respons på utgangen. Når vi satt P=30, I=50 og D=0 så fikk vi en brukbar respons. Siden det er en diskret PID-regulator så må man også definere samplingstid, det er

hvor ofte den skal lese av den målte hastigheten, vi satt samplingstid til 0.1 sekunder. Figur 8.39 viser konfigurasjonen av PID-regulatoren.

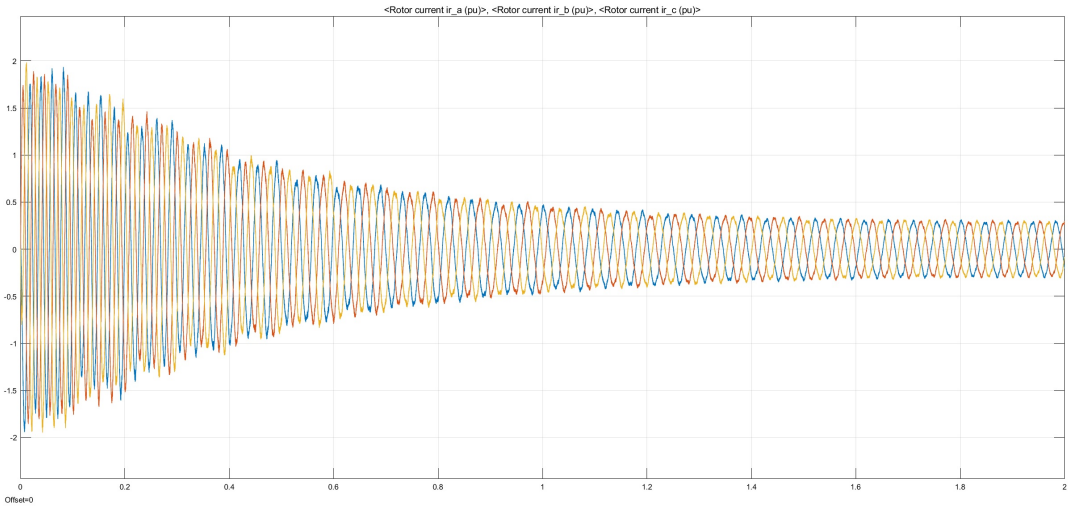


Figur 8.39.: PID-parameterne

Med disse parameterne så fikk vi stator- og rotor- strømmer som vist på Figur 8.40 og Figur 8.41. Figur 8.42 viser motorens elektromagnetiske torque. Vi ser motoren oppfører seg på samme måte som i åpen sløyfe systemet men her ser vi at signalene har mindre rippelstøy.

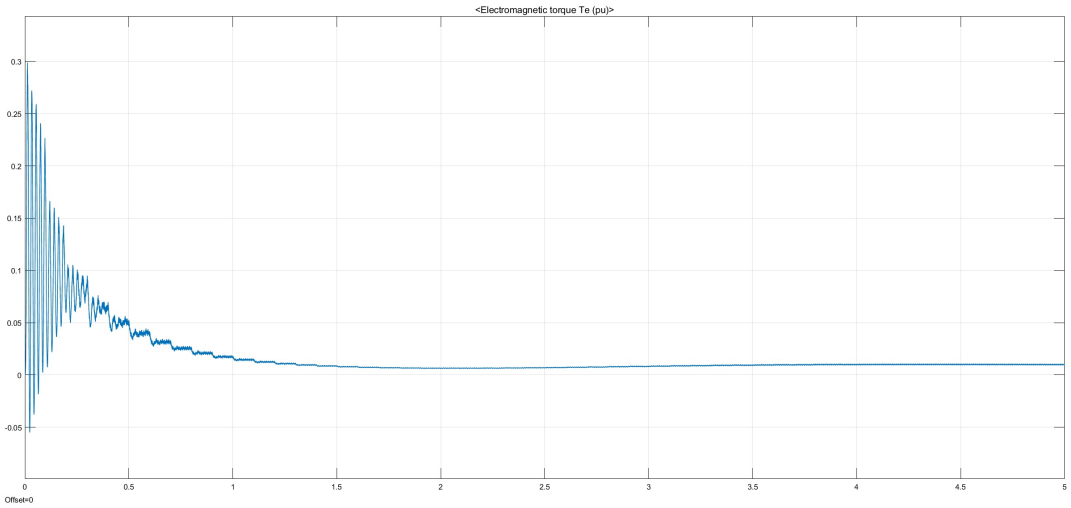


Figur 8.40.: Stator strømmene til motoren med lukket sløyfe.

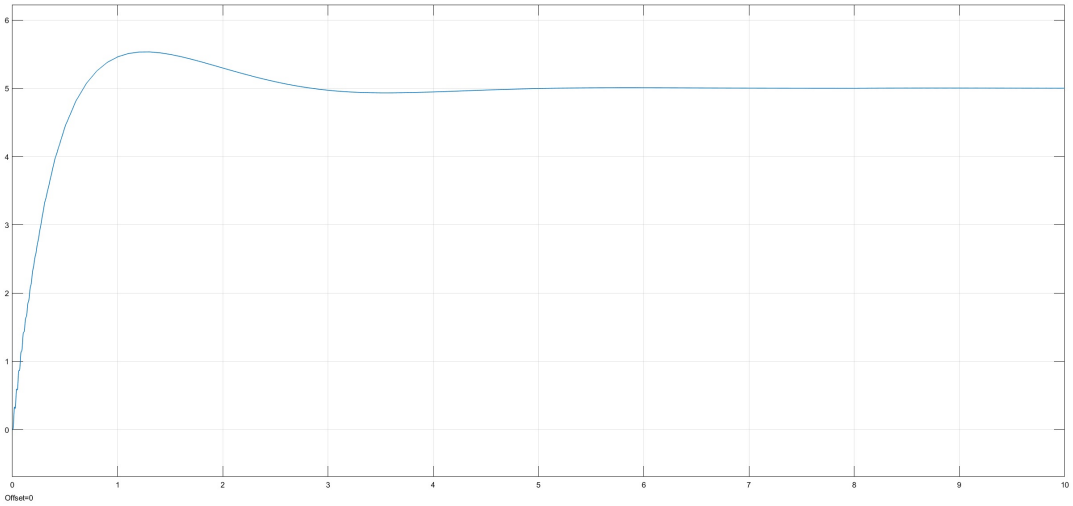


Figur 8.41.: Rotorstrømmene til motoren med lukket sløyfe.

Vi satt referansehastighet på 5 RPM, la oss nå se om motoren klarer å regulere seg selv opp til denne hastigheten på utgangen. Figur 8.43 viser resultatet.



Figur 8.42.: Elektromagnetisk torque med lukket sløyfe.



Figur 8.43.: Rotasjonshastighet med lukket sløyfe.

Fra Figur 8.43 ser vi at resultatet gir oss en stigningstid på ca 0.8 sekunder, overskyt på ca 0.5 m/s og en settlingtid på ca 4 sekunder Etter det stabiliserer hastigheten seg på 5 RPM som ønsket. Vi har fortsatt det problemet hengende igjen fra tidligere at motoren bare klarer å operere riktig på lave hastigheter, jeg testet med forskjellige referanse verdier og det ser ut som den bare klarer å operere innenfor intervallet 0-10 m/s, noe høyere enn det og systemet virker ikke lenger. Men lukket sløyfe-systemet skal være satt opp riktig, for den gir forventet respons på de lave hastighetene, så problemet har en sammenheng med det tidligere problemet fra åpen sløyfe konfigurasjonen. Vi har nå verifisert at det simulerte styringssystemet virker innenfor et visst hastighetsintervall (0-10 RPM), men ikke innenfor hele intervallet som motoren skulle være istand til å generere (0-1430 RPM).

8.6. Kretskortdesign

Denne seksjonen tar for seg designprosessen for de kretskortene vi har bestemt oss for å lage selv. Dette gjelder 2 kretskort; Digital til Analog konverteren og lastcelle-forsterkeren. Vi bruker gratisprogramvaren **KiCad** til å designe kortene. Designprosessen for kretskort kan deles opp i tre deler: skjemategning, simulasjon og PCB-design. Man tegner først opp kretsen i en programvare. Man annoterer komponentene og spesifiserer et fotavtrykk for hver type komponent, deretter simulerer man kretsen for å verifisere at man får riktig signaloppførsel og tilslutt designer man selve kretskortet i PCB-verktøyet.

PCB-design fasen er den mest sentrale og viktigste delen av designprosessen. Med PCB-verktøyet definerer vi størrelse og antall lag på brettet, plasserer fotavtrykkene til komponentene på en fornuftig måte slik at vi både sparer plass på brettet men også slik at det blir lett å rute koblingene med kobberspor. Man deler gjerne kobbersporene inn i to størrelser, en for strømforsyning og en for signal. Det er vanlig at kobbersporene for strømforsyning er betydelig bredere enn kobbersporene for signalet med hensyn til elektromagnetisk kompatibilitet.

Det er også viktig at de forskjellige kobbersporene ikke står for nærme hverandre da dette kan skape interferens som negativt påvirker kretsen.

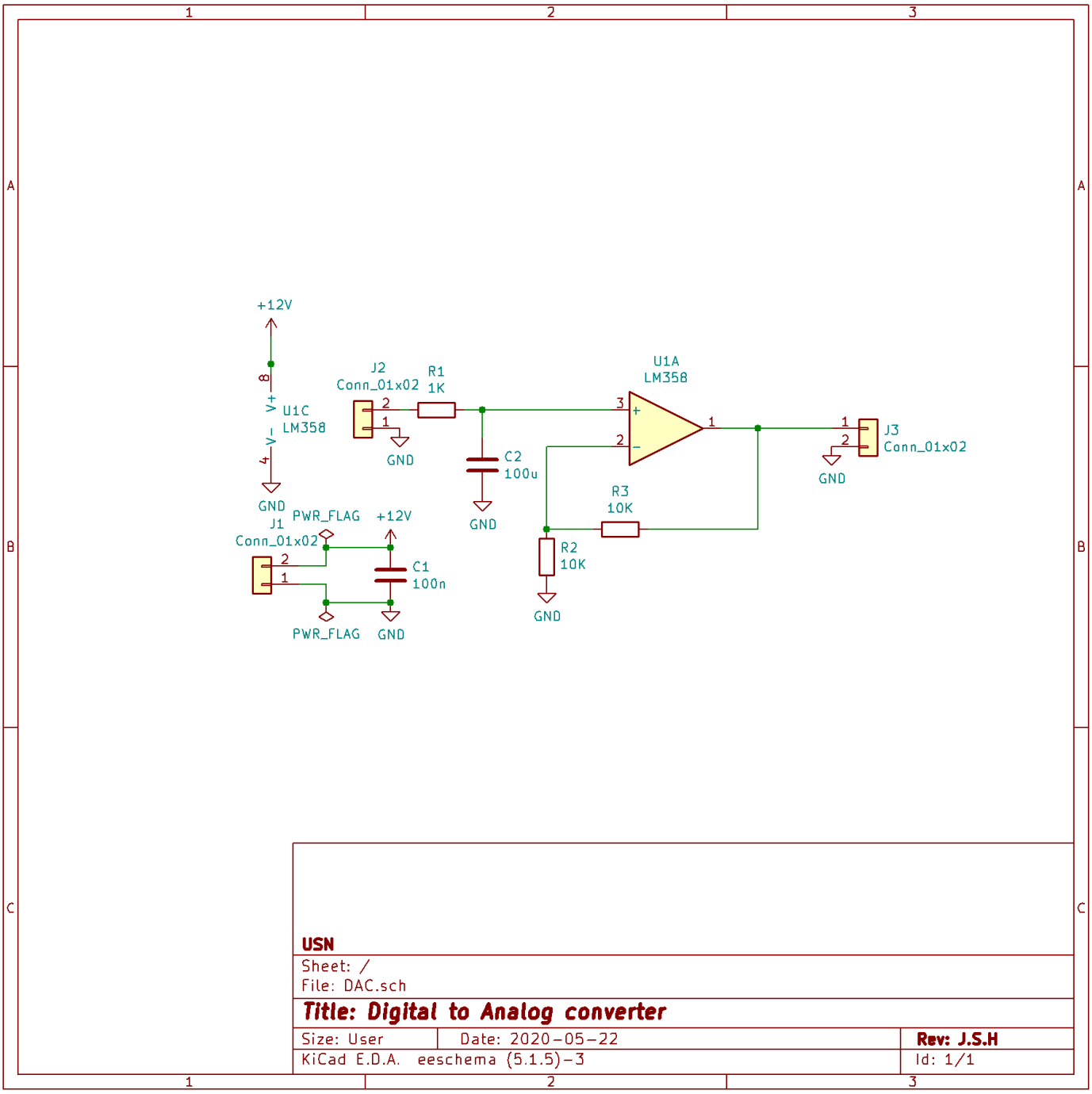
Når kretskortet er ferdig designet gjenstår det bare å hente ut gerber og drill filene fra **KiCad**, disse filene inneholder all den informasjonen som produsenten trenger for å kunne produsere kortene. Gerber-filene inneholder informasjon om hvordan hvert lag av kretskortene er designet, mens drill-filene inneholder informasjon om plassering og dimensjoner på hull som skal drilles på kortet.

8.7. Design av DAC kretskort

Design av DAC kretskortet er en komponent som er knyttet til krav K.VT 04 som hører til styringssystemet.

8.7.1. Skjemategning for DAC kortet

Figur 8.44 viser skjemategningen for Digital til Analog konverter kortet.



Figur 8.44.: Skjemategning for DAC-kortet.

Beskrivelse av skjemategning

J1 er tilkoblingen til 12V strømforsyningen som går til LM358 operasjonsforsterkeren. Mellom 12V og jord har vi lagt på en dekopling kondensator (C1) for å fjerne ripple-støy som kan komme fra strømforsyningen. J2 er tilkoblingen til inngangssignalet som kommer fra mikrokontrolleren og J3 er utgangssignalet som går videre til motorkontroller kortet.

8.7.2. Tildeling av fotavtrykk til DAC kortet

Fotavtrykk er de linjene tegnet på kretskortet som markerer hvor de ulike komponentene skal plasseres, man skiller mellom fotavtrykk for overflatemonterte komponenter og fotavtrykk for komponenter med pins som skal gjennom drillede hull i kretskortet. For motstander og kondensatorer bruker jeg overflatemonterte fotavtrykk med størrelse 0603 som står for 6 inches i lengde og 3 inches i bredde. For koblingsportene bruker jeg fotavtrykk for 1x2 skruterminaler med 3.5 mm mellomrom mellom pinsene.

Figur 8.45 viser liste over alle fotavtrykkene som brukes til kortet.

Symbol : Footprint Assignments

1	C1 -	100n	: Capacitor_SMD:C_0603_1608Metric
2	C2 -	100u	: Capacitor_SMD:C_0603_1608Metric
3	J1 -	Conn_01x02	: TerminalBlock_4Ucon:TerminalBlock_4Ucon_1x02_P3.50mm_Horizontal
4	J2 -	Conn_01x02	: TerminalBlock_4Ucon:TerminalBlock_4Ucon_1x02_P3.50mm_Horizontal
5	J3 -	Conn_01x02	: TerminalBlock_4Ucon:TerminalBlock_4Ucon_1x02_P3.50mm_Horizontal
6	R1 -	1K	: Resistor_SMD:R_0603_1608Metric
7	R2 -	10K	: Resistor_SMD:R_0603_1608Metric
8	R3 -	10K	: Resistor_SMD:R_0603_1608Metric
9	U1 -	LM358	: LM358N:DIP762W46P254L960H508Q8

Figur 8.45.: Fotavtrykk til komponentene.

8.7.3. Materialliste for DAC kortet

Tabell 8.3 viser materiallisten for alle komponentene til DAC kortet med leverandørnavn og priser i NOK. Komponenter med kvantitet større enn 1 som har lik totalpris er på grunn av at de kommer i en pakke.

Referanse	**Kvantitet**	**Verdi**	**Leverandør**	**Pris**	**Totalpris**
C1	1	100n	Elfa	1.92	1.92
C2	1	100u	Elfa	17.51	17.51
J1, J2, J3	3	Connector (01x02)	Digikey	13.56	40.68
R1	1	1k	Elfa	1.15	1.15
R2, R3	2	10k	Elfa	0.1	0.1
U1	1	LM358	Elfa	3.5	3.5

Tabell 8.3.: Materialliste for komponenter til DAC kortet.

8.7.4. Simulering av DAC kortet

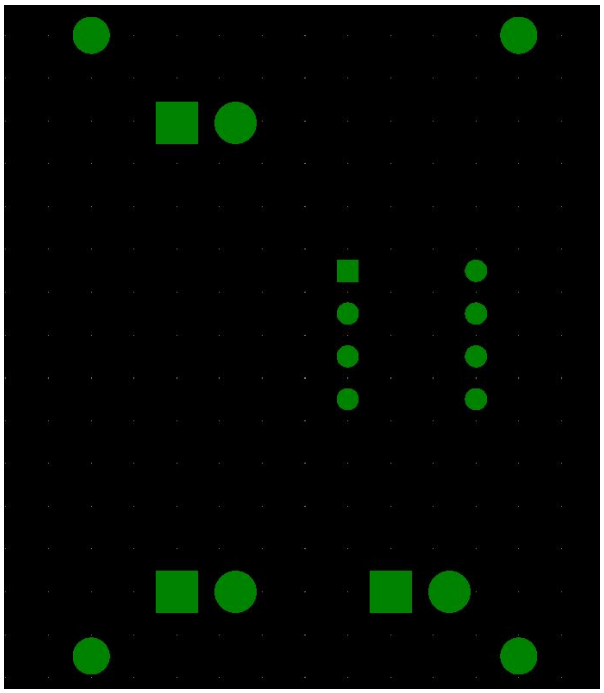
Se 8.4.3.

8.7.5. DAC PCB utlegg

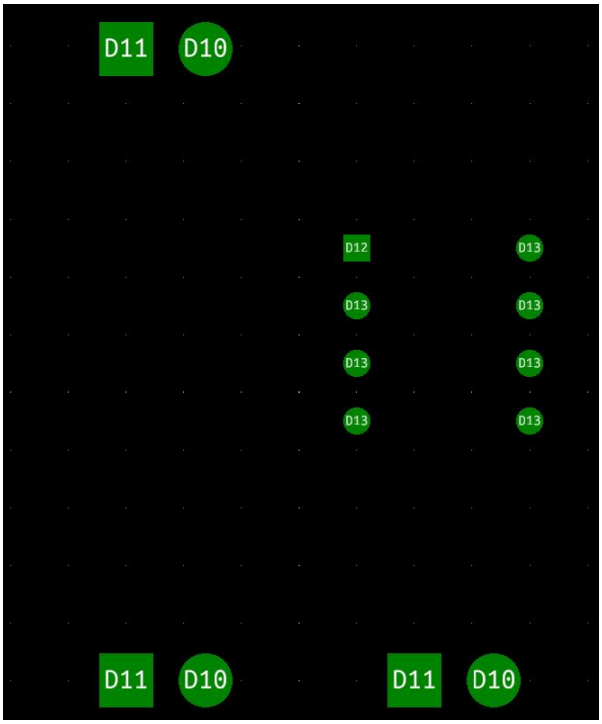
Nå som kretsen er annotert og alle fotavtrykk er tildelt for komponentene så genererer vi en netlist fil, denne filen lastes inn i PCB-design programmet. Når netlist filen er lastet inn kan vi begynne å plassere alle komponentene og rute dem. For kobberspor til strømforsyning bruker vi en bredde på 0.635 mm som tilsvarer 25 mils og kobberspor for signalet bruker vi en bredde

på 0.25 mm som tilsvarer ca 10 mils. Mils er en måleenhet som brukes som standard i PCB-utvikling industrien, 10 mils for signal og 25 mils for strømforsyning kobberspor er veldig standard brukt i industrien. Vi legger på fire monteringshull med en diameter på 2.1 mm, ett i hvert hjørne. Hele kortet har en dimensjon på 41.91 x 30.48 mm. Når vi er ferdige med å definere lengde og bredde på brettet, plassert komponenter, tekst og alle kobberspor er rutet sammen, så kan vi produsere gerber-filene for de nødvendige lagene og studere det endelige resultatet på 3D-modellen.

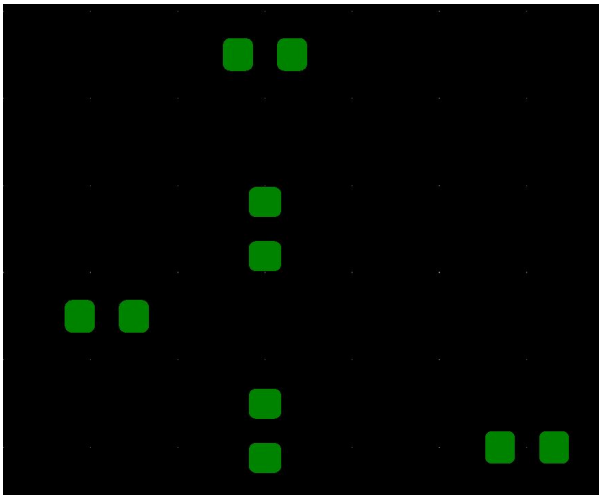
Gerber-filer



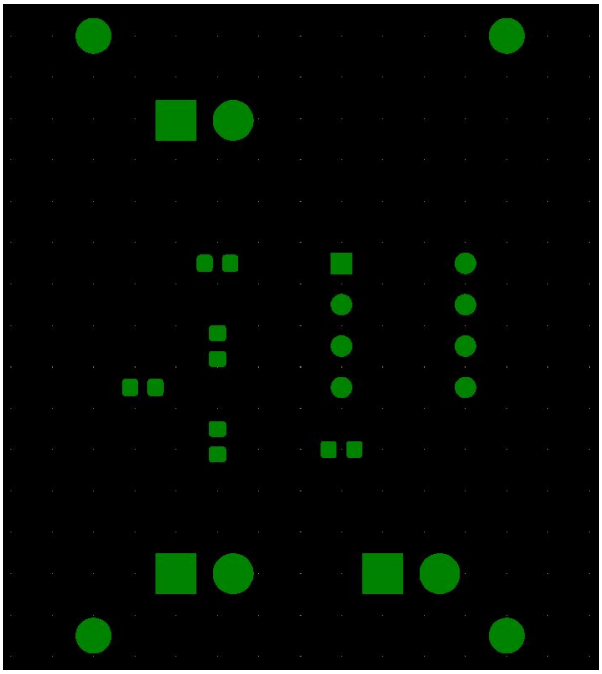
Figur 8.46.: Mask-laget på baksiden av kortet for komponenter som skal loddes.



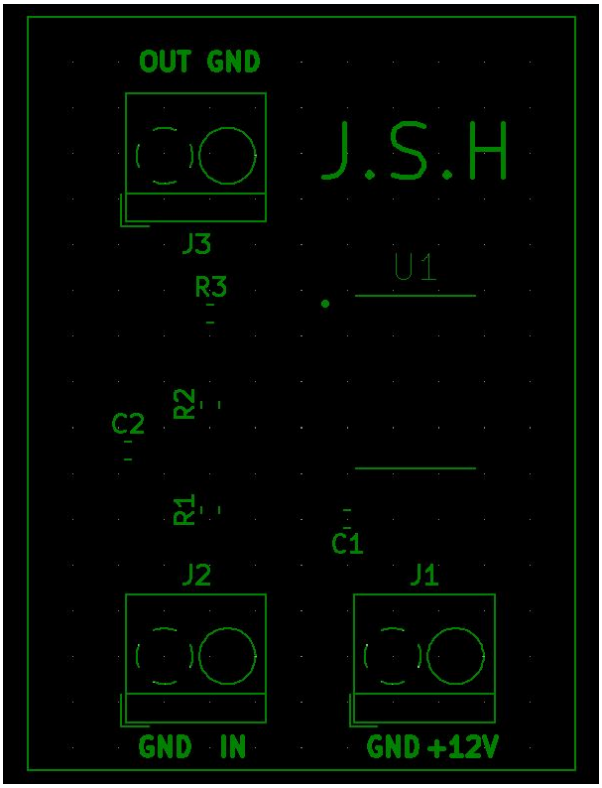
Figur 8.47.: Kobberspor-laget på baksiden av kortet.



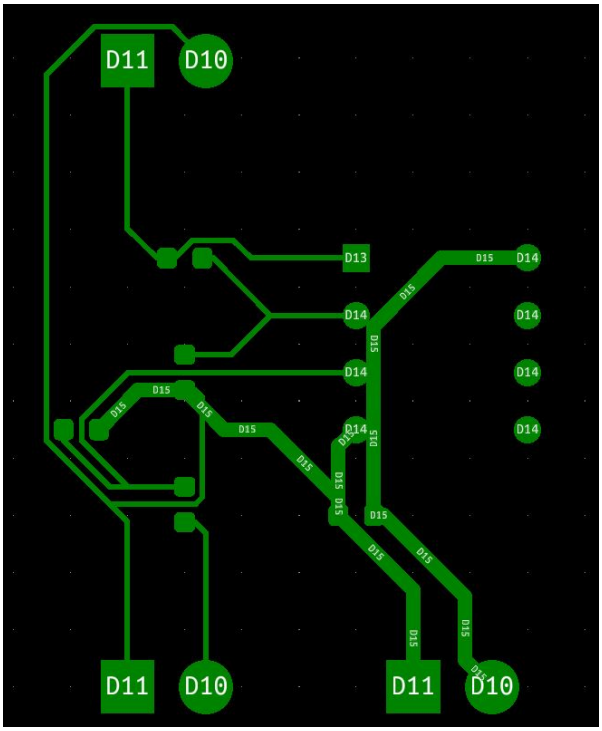
Figur 8.48.: Paste-laget på forsiden av kortet for overflatemonterte komponenter.



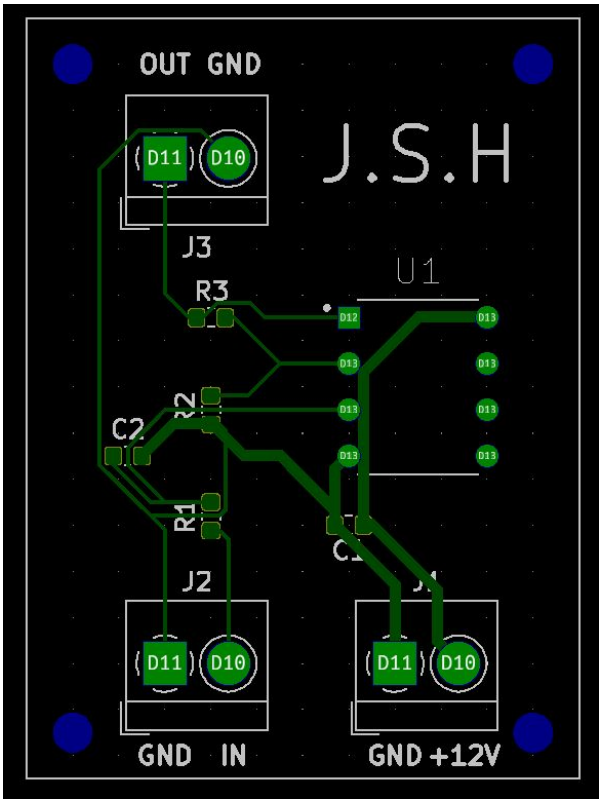
Figur 8.49.: Mask-laget på forsiden for komponenter som skal loddes på kortet.



Figur 8.50.: Silkscreen-laget på forsiden av kortet for tekst og fotavtrykk.

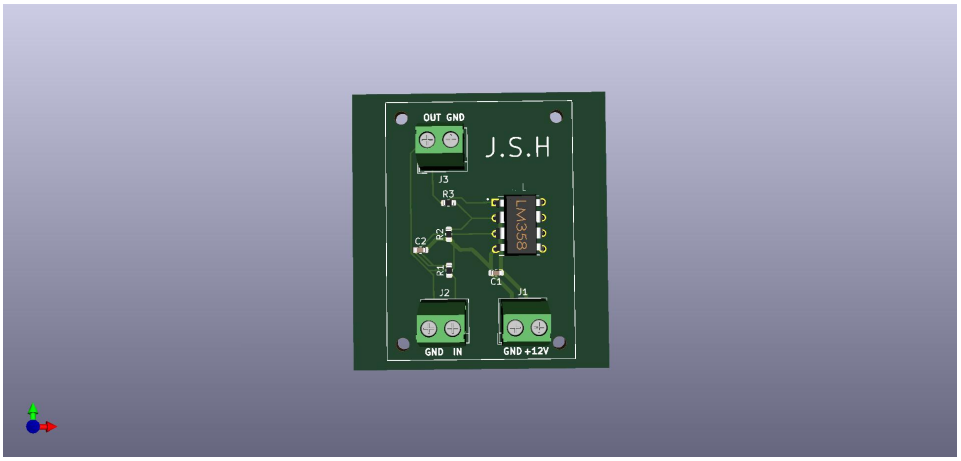


Figur 8.51.: Kobberspor-laget på forsiden av kortet.

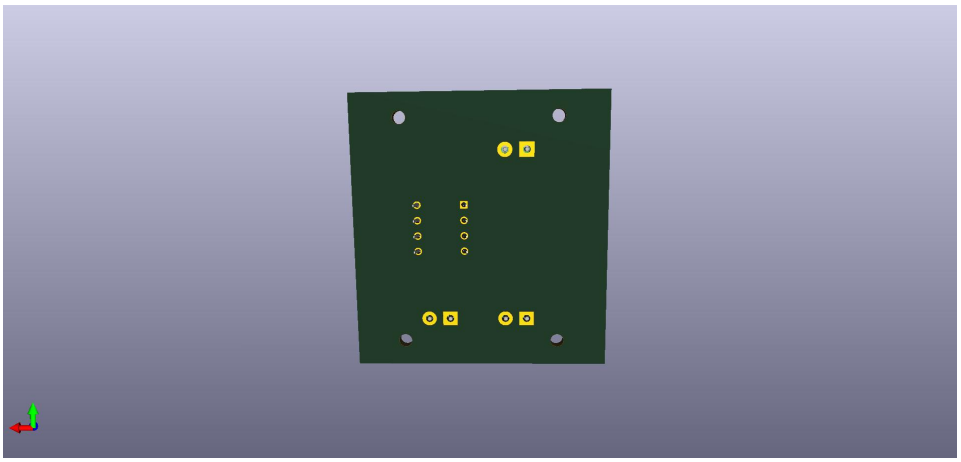


Figur 8.52.: Hele kortet med alle lagene.

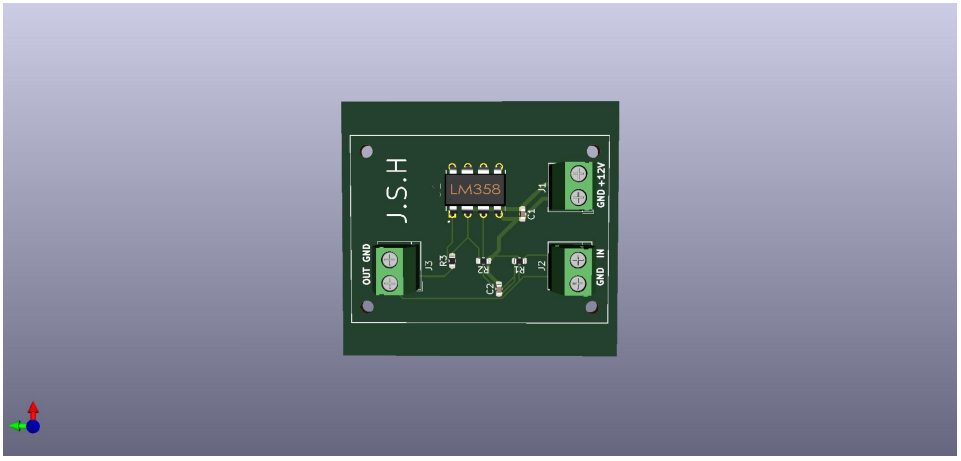
3D modell



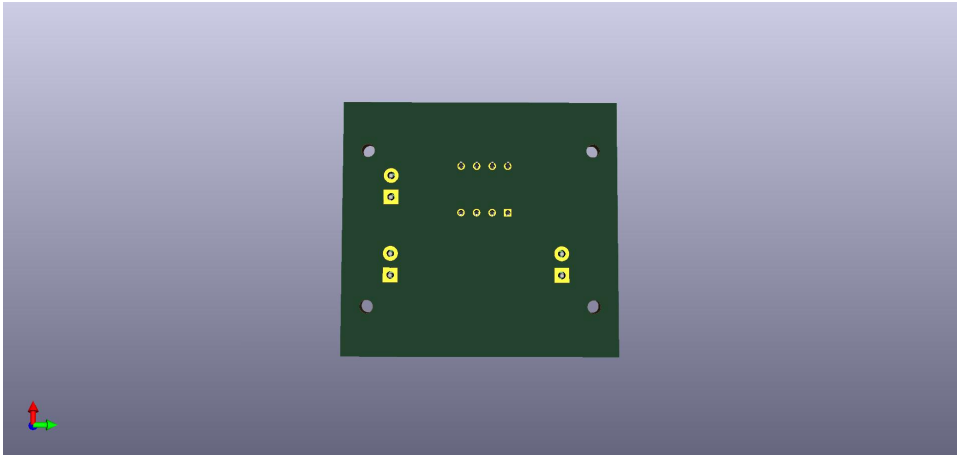
Figur 8.53.: Forsiden av kortet vertikalt.



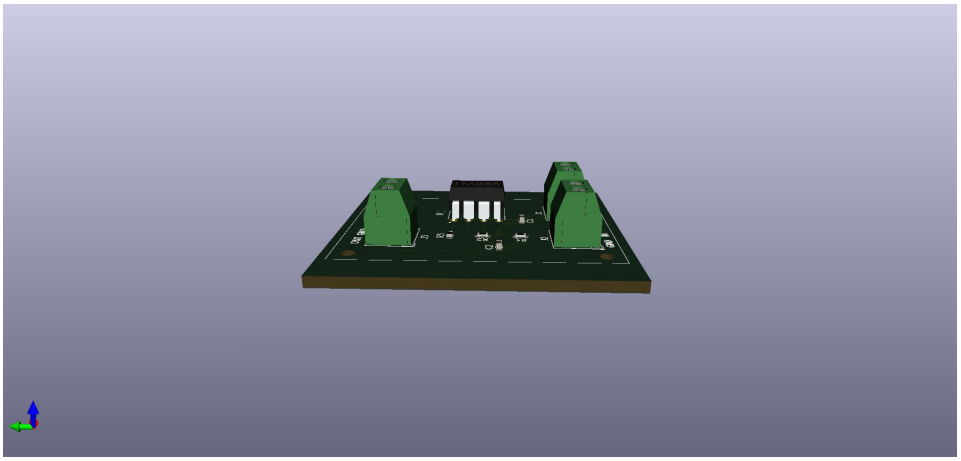
Figur 8.54.: Baksiden av kortet vertikalt.



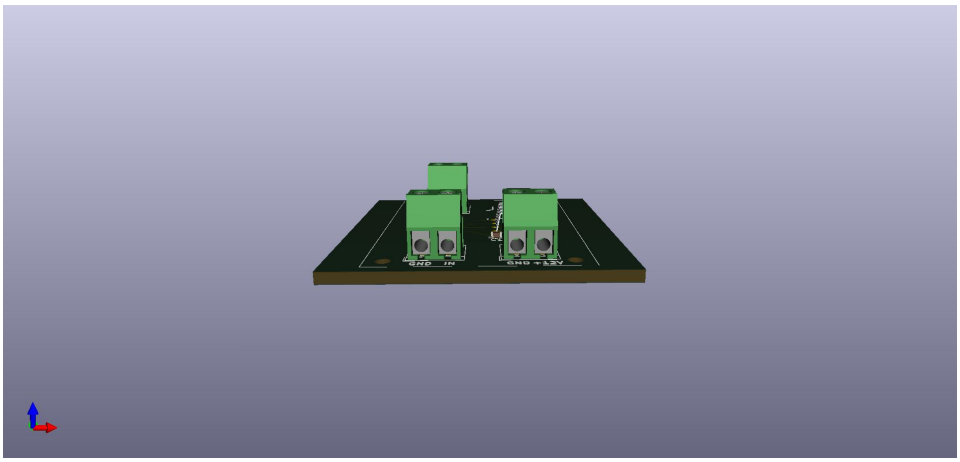
Figur 8.55.: Forsiden av kortet horisontalt.



Figur 8.56.: Baksiden av kortet horisontalt.



Figur 8.57.: Kortet sett fra siden.



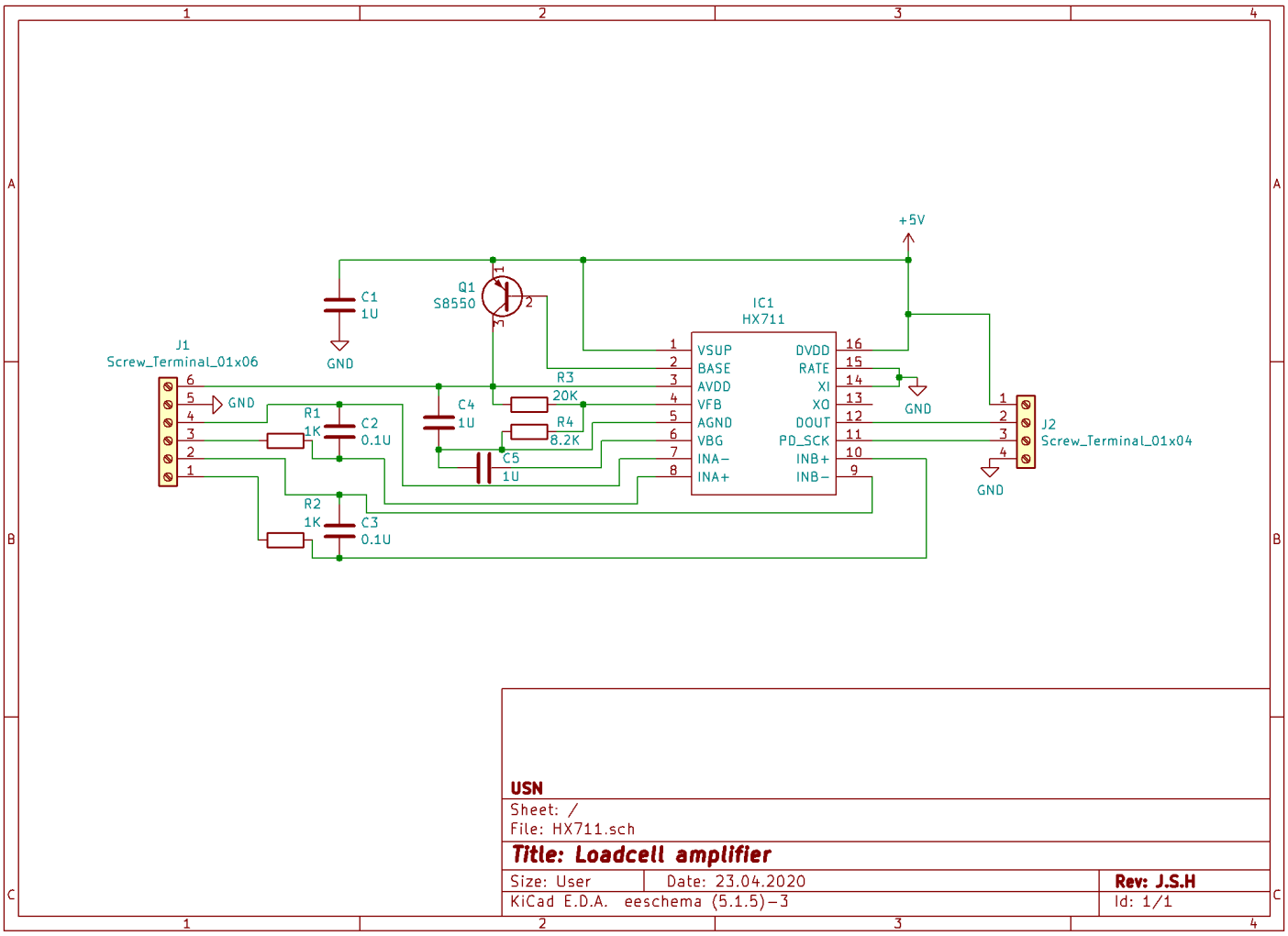
Figur 8.58.: Kortet sett fra siden.

8.8. Design av lastcelle-forsterker kretskortet

Den originale planen var å bestille et allerede ferdig kort for lastcellene, men på grunn av Covid-19 utbruddet ble prosjektet vårt endret fra praktisk til teoretisk prosjekt med enda mer fokus på system design og simulering. Vi velger derfor å lage dette kortet fra bunnen av. Kortet vi designer i denne seksjonen har identisk funksjonalitet som det HX711 lastcelleforsterker kortet vi hadde planer om å bestille som nevnt tidligere i prosjektrapporten. Denne komponenten er knyttet til krav K.M 08 og er en del av delsystemet for måling av krefter på testmodellen i vindtunnelen.

8.8.1. Skjemategning for HX711 kortet

Figur 8.59 viser kretsskjema for HX711 lastcelleforsterker kortet.



Figur 8.59.: Skjemategning for HX711 kortet.

Beskrivelse av skjemategning

J1 har 6 tilkoblingspunkter; port 1-2 er for utgangssignal for den ene lastcelle-forsterkeren og port 3-4 er utgangssignal for den andre lastcelle-forsterkeren. Port 5-6 er spenningsforsyning for lastcelle-forsterkerne. På J2 er port 1 og 4 strømforsyning til HX711 IC-en, port 2 er digital data ut til mikrokontrolleren og port 3 er for klokkesignalet inn som sendes fra mikrokontrolleren.

8.8.2. Tildeling av fotavtrykk til HX711 kortet

For kondensatorer og motstander bruker vi her også 0603 størrelsen. For tilkoblingsportene bruker vi 2.54mm avstand mellom pinsene. Figur 8.60 viser liste av fotavtrykkene til alle komponentene til kortet.

Symbol : Footprint Assignments

1	C1	1U	Capacitor_SMD:C_0603_1608Metric
2	C2	0.1U	Capacitor_SMD:C_0603_1608Metric
3	C3	0.1U	Capacitor_SMD:C_0603_1608Metric
4	C4	1U	Capacitor_SMD:C_0603_1608Metric
5	C5	1U	Capacitor_SMD:C_0603_1608Metric
6	IC1	HX711	HX711:SOIC127P600X160-16N
7	J1	Screw_Terminal_01x06	Connector_PinHeader_2.54mm:PinHeader_1x06_P2.54mm_Vertical
8	J2	Screw_Terminal_01x04	Connector_PinHeader_2.54mm:PinHeader_1x04_P2.54mm_Vertical
9	Q1	S8550	Package_TO_SOT_THT:TO-92_Inline
10	R1	1K	Resistor_SMD:R_0603_1608Metric
11	R2	1K	Resistor_SMD:R_0603_1608Metric
12	R3	20K	Resistor_SMD:R_0603_1608Metric
13	R4	8.2K	Resistor_SMD:R_0603_1608Metric

Figur 8.60.: Fotavtrykk til komponentene til HX711 kortet.

8.8.3. Materialliste for HX711 kortet

Tabell 8.4 viser materiallisten for alle komponentene til HX711 kortet med leverandørnavn og priser i NOK. Komponenter med kvantitet større enn 1 som har lik totalpris som pris er pågrunn av at de kommer i en pakke.

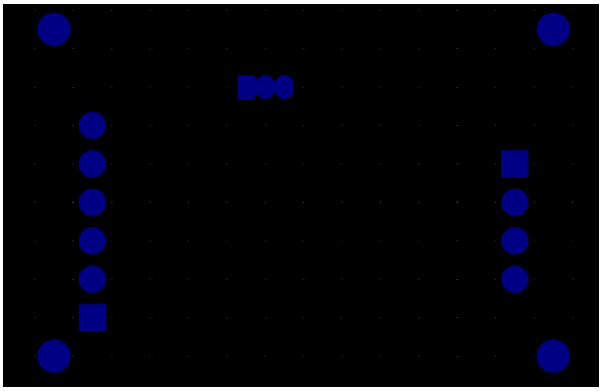
Referanse	**Kvantitet**	**Verdi**	**Leverandør**	**Pris**	**Totalpris**
C1, C4, C5	3	1u	Elfa	0.23	0.69
C2, C3	2	0.1u	Elfa	3.24	3.24
IC1	1	HX711	Alibaba	1.04	1.04
J1	1	01x06 pin	Elfa	1.02	1.02
J2	1	01x04 pin	Elfa	0.64	0.64
Q1	1	s8550	Alibaba	0.26	0.26
R1, R2	2	1k	Elfa	1.12	2.24
R3	1	20k	Elfa	1.28	1.28
R4	1	8.2k	Elfa	0.32	0.32

Tabell 8.4.: Materialliste med komponenter til HX711 kortet.

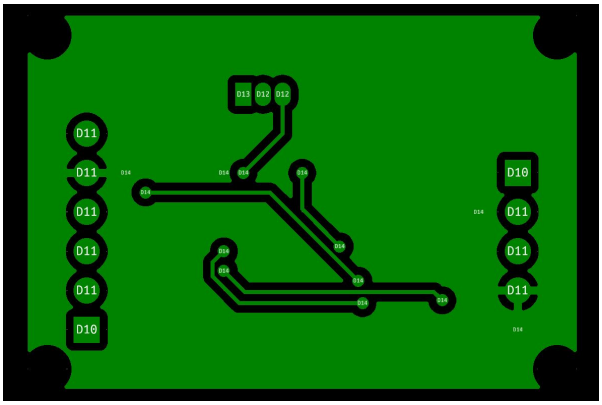
8.8.4. HX711 PCB utlegg

Vi gjør samme prosedyre som på forrige kretskort, eneste forskjellen er at på dette kortet lager vi et jordplan på baksiden av kortet slik at de fleste komponenter som må kobles til jord kobles på en “via” som er et hull drillet gjennom kortet med en kobberleder tilkoblet jordplanet. Via-ene har en diameter på 0.8 mm. Vi bruker også via-er til å rute på baksiden til de tilkoblingene vi ikke får plass til på forsiden av kortet. Vi har også lagt ved fire monteringshull, ett i hvert hjørne, disse har en diameter på 2.1 mm. Dimensjonene til hele kortet er 35.56 x 24.13 mm.

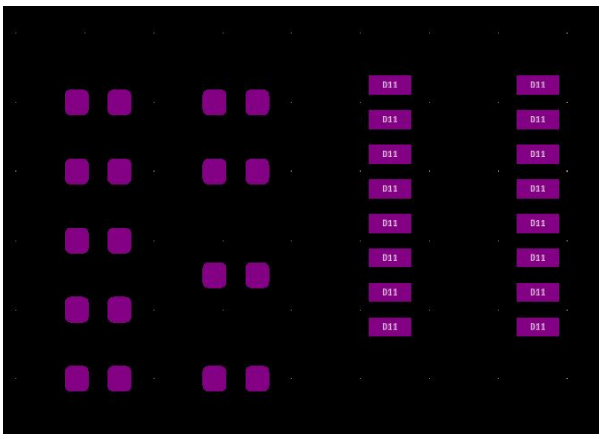
Gerber-filer



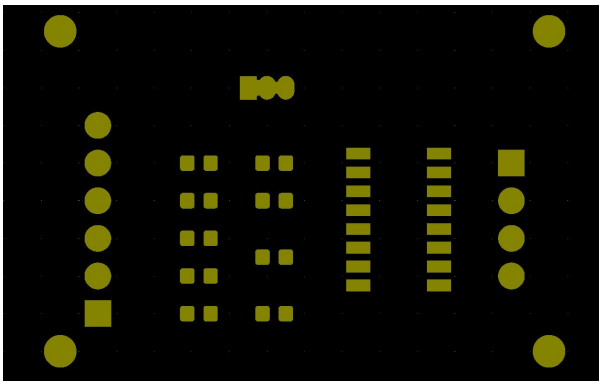
Figur 8.61.: Mask-laget på baksiden av kortet for komponenter som skal loddes.



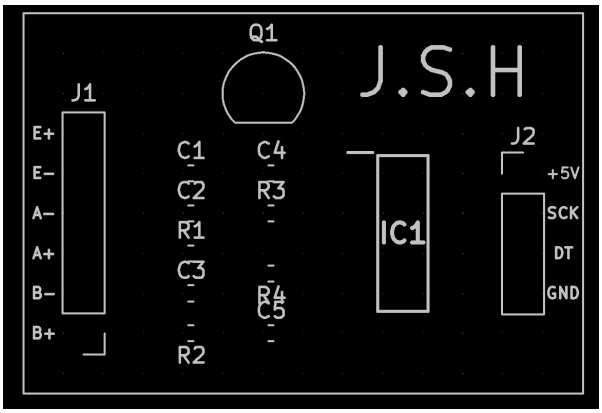
Figur 8.62.: Kobberspor-laget på baksiden av kortet.



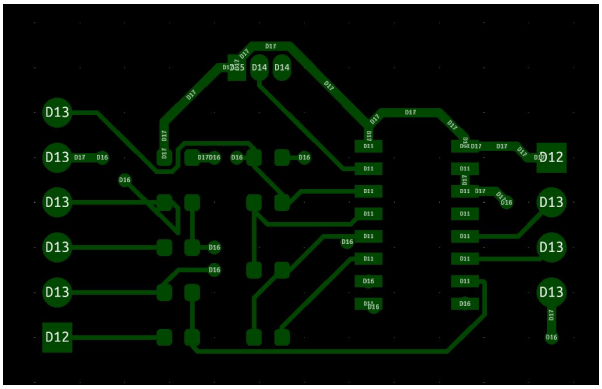
Figur 8.63.: Paste-laget på forsiden av kortet for overflatemonterte komponenter.



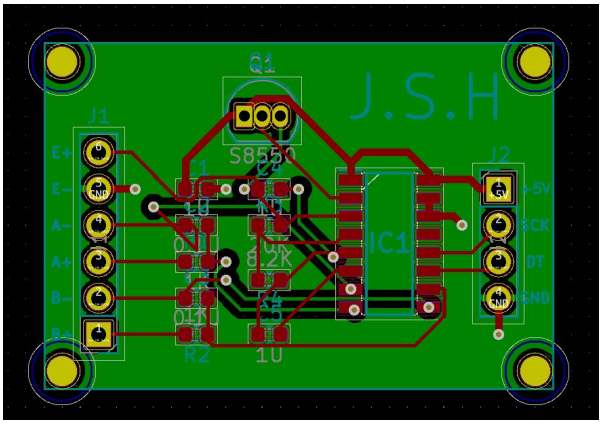
Figur 8.64.: Mask-laget på forsiden for komponenter som skal loddes på kortet.



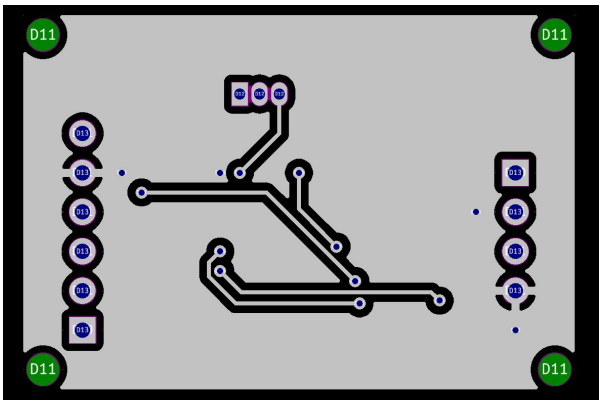
Figur 8.65.: Silkscreen-laget på forsiden av kortet for tekst og fotavtrykk.



Figur 8.66.: Kobberspor-laget på forsiden av kortet.

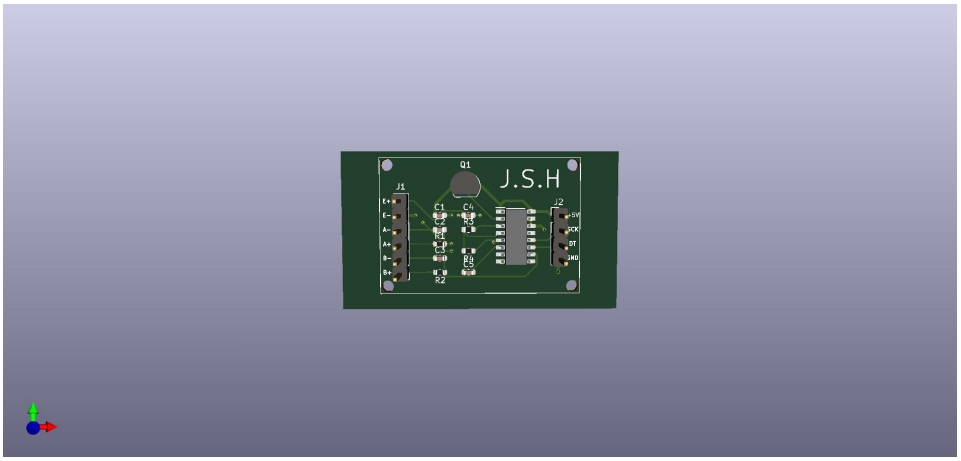


Figur 8.67.: Hele kortet med alle lagene sett fra forsiden.

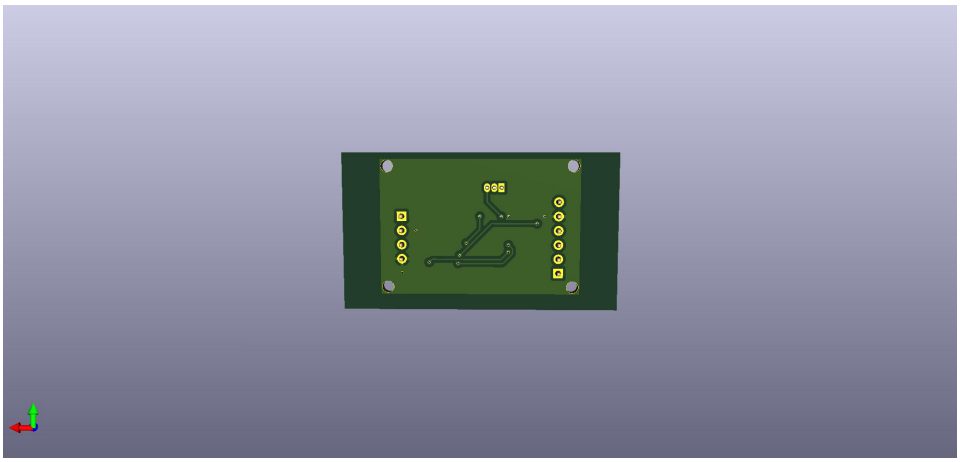


Figur 8.68.: Hele kortet med alle lagene sett fra baksiden.

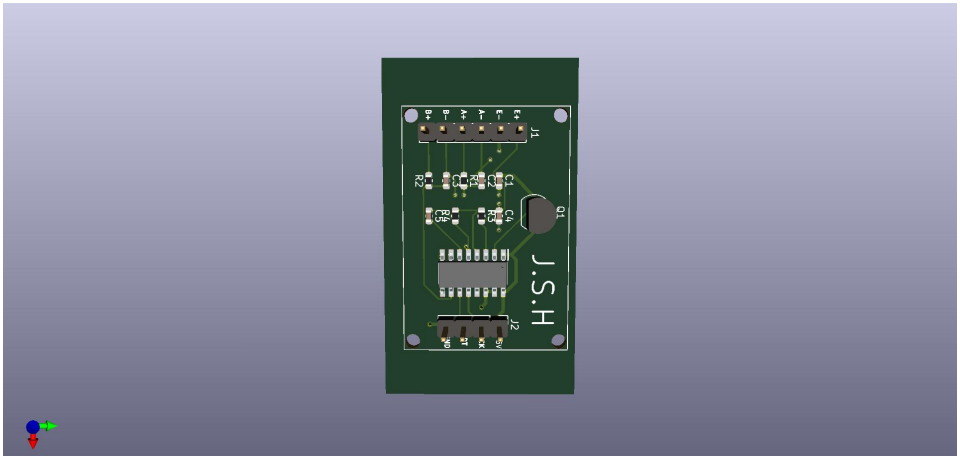
3D modell



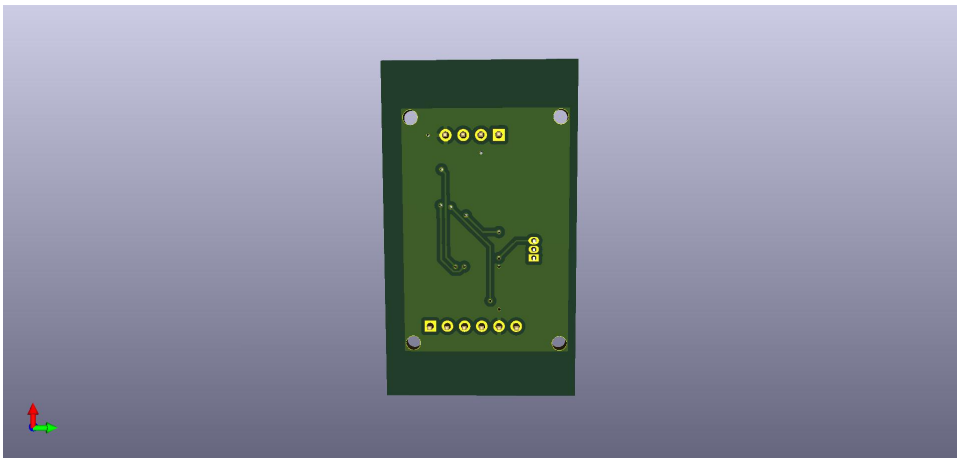
Figur 8.69.: Forsiden av kortet horisontalt.



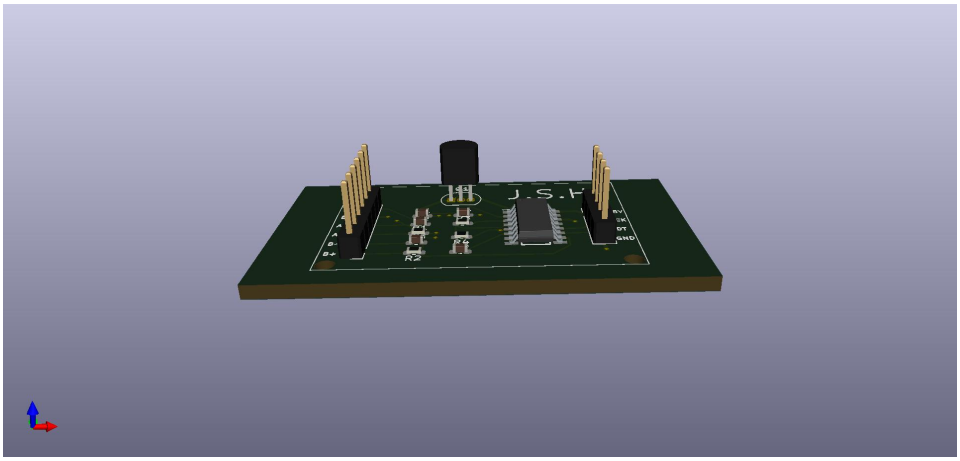
Figur 8.70.: Baksiden av kortet horisontalt.



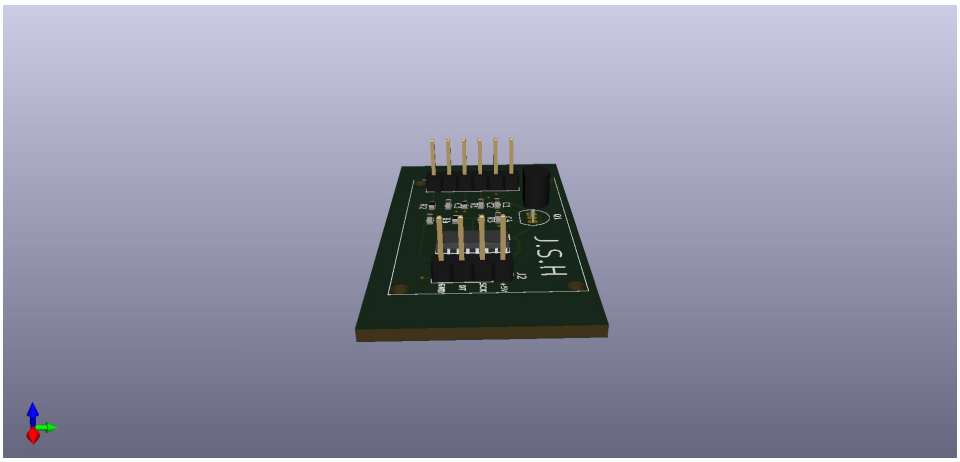
Figur 8.71.: Forsiden av kortet vertikalt.



Figur 8.72.: Baksiden av kortet vertikalt.



Figur 8.73.: Kortet sett fra siden horisontalt.



Figur 8.74.: Kortet sett fra siden vertikalt.

9. Maskinteknisk design

9.1. Grunnleggende designtankegang vindtunnel

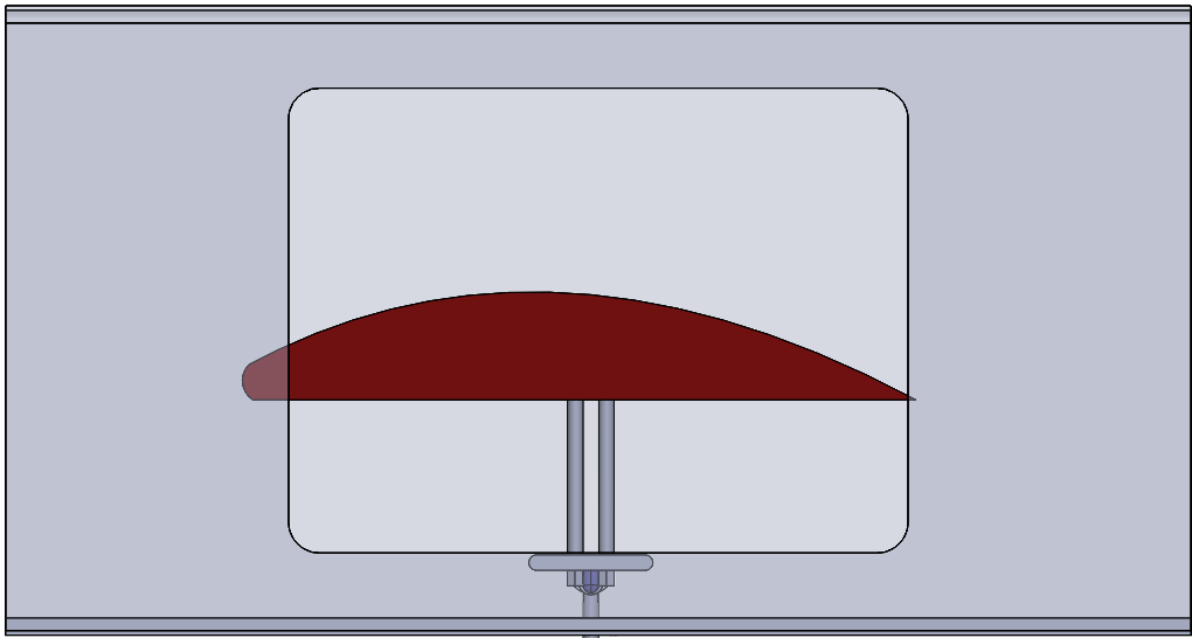
9.1.1. Ordinært design

Vanligvis når man skal designe en vindtunnel begynner man med å kartlegge behovet som i [1]. I vårt tilfelle var behovet en vindtunnel som kan brukes i undervisning, og noe forskning. Dette er allerede toneangivende for hvordan man bør designe. Men vi fikk ikke noen ønsket vindhastighet, eller krav for størrelse på modell.

Vanligvis ville man begynt med å designe testkammeret først slik at man kan få dimensjonert det for ønsket lufthastighet og størrelse på testmodell. Deretter har man da endearealer som kan brukes til å planlegge videre systemarkitektur etter. Her ville det også ha vært smart å bruke beregninger i fluidmekanikk for å verifisere at man fortsatt kan få den samme volumstrømmen til enhver tid.

Det er viktig at man har et testkammer som er stort nok for det som skal testes i det, om det er f.eks en bil eller en skalamodell av et fly. Størrelsen er avgjørende siden man trenger at modellen ikke er for stor slik at den hindrer luftflyt gjennom testkammeret. Det må også avgjøres om testkammeret bør være rundt eller ha en annen form med tanke på testinstrumenter og hvor lett det er for luft som fluid å trekkes gjennom kammeret.

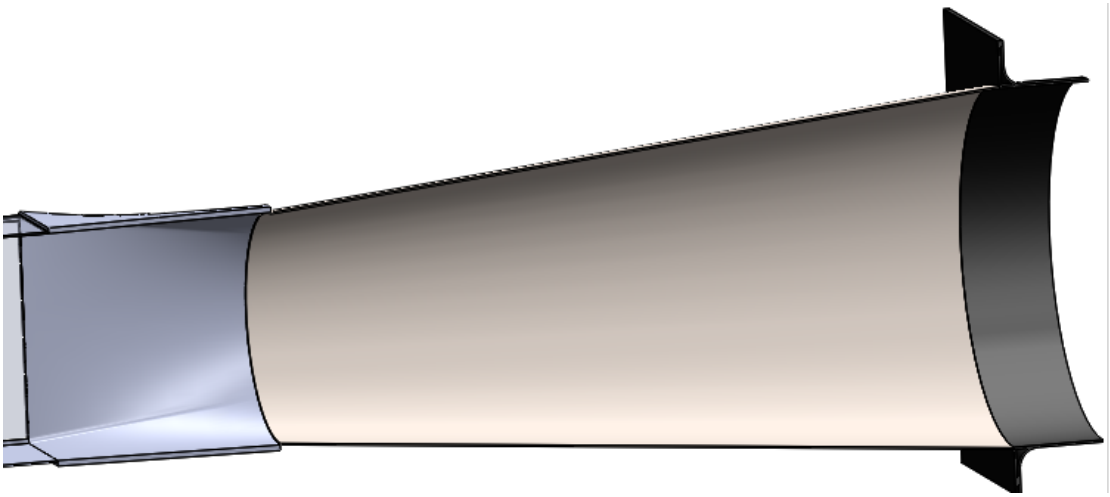
Etter at man har designet testkammeret har man en oversikt over hva som er de nødvendige parameterene å designe videre etter. Man har også da endene på testkammeret slik at man vet



Figur 9.1.: Testkammer med tenkt modell.

hvilke størrelser dysen og diffuseren må ha som sin minste profil. For dysen vil dette si uttaket for luft, og for diffuseren er det da inntaket. Deretter ville det vært smart å designet diffuseren siden den skal lede luften til viften. Viften igjen må også dimensjoneres i forhold til testkammeret slik at man får et godt forhold mellom tverrsnittsarealet til viften og testkammeret, på denne måten får man ifølge [1] god luftflyt gjennom vindtunnelen. Når man da vet hvor stor vifte man skal bruke er det også mye lettere å designe dysen slik at den forbinder dette endestykket på testkammeret og viften slik at luften får sakket ned og man får gjenopprettet trykket der slik at man får en jevnest mulig strømning gjennom vindtunnelen.

Samtidig som man kjenner tverrsnittsarealet til testkammeret kan man også designe dysen. Det er denne delen som skal forsøke å komprimere luften som trekkes inn i vindtunnelen slik at man kan få størst mulig vindhastighet inni vindtunnelen. Men for å oppnå dette må man også bestemme to parametere. Man må først bestemme seg for hvor stort tverrsnittsareal som skal brukes for å hente inn luft fra omgivelsene. Det optimale forholdet ligger mellom 6 og 10 ifølge [4]. Etter å ha funnet det beste forholdstallet ifølge krav så må man få til en god kurve for design

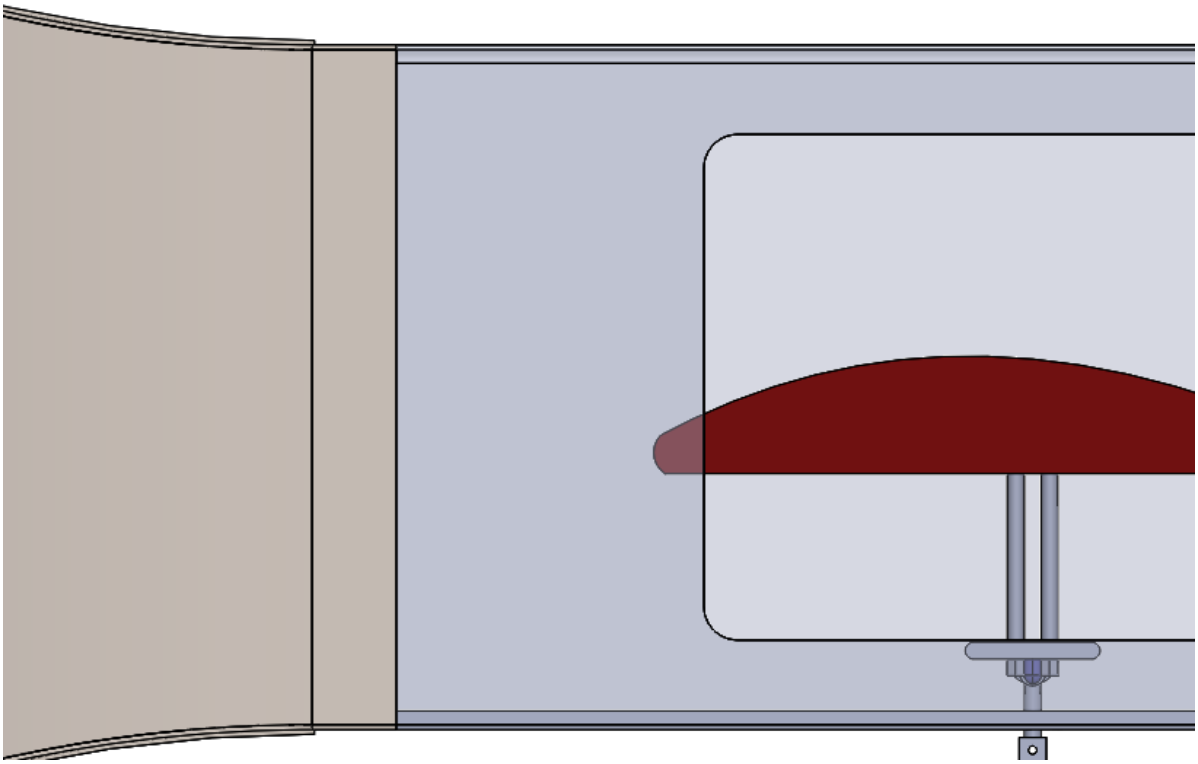


Figur 9.2.: Veien for luften ut av testkammer og ut der viften står i den sorte rammen.

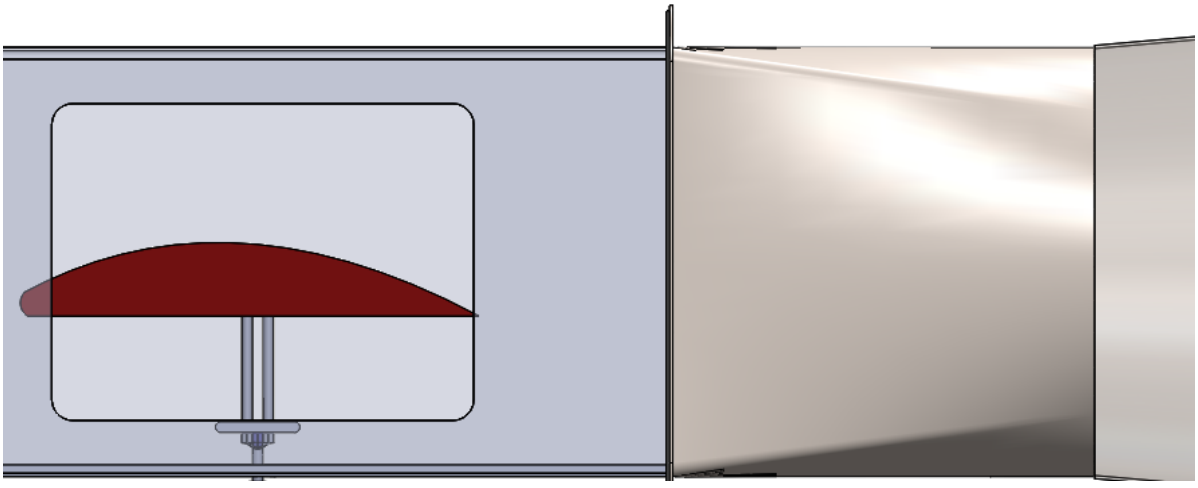
av dysen sånn at den kan presse sammen luften sånn at man får best mulig luftkvalitet. Dette kan være krevende med tanke på at man kan trenge å designe denne sammentrekningskurven (kontraksjonskurven) med en femtegrads likning.

Når man har oversikt over ønsket lufthastighet og hvor stort tverrsnittsareal som skal være på dysen så kan man dimensjonere og designe manipuleringskammeret. Manipuleringskammeret sin funksjon er å rette luftstrømmen som trekkes inn i vindtunnelen slik at man får en mest mulig laminær strøm som ikke beveger seg turbulent men jevnt gjennom hele tverrsnittet til testkammeret.

Når man da har fått naglet designet som trengs for å få ønsket luftkvalitet må man også designe for ønsket mobilitet. I noen tilfeller er ikke det en stor prioritet, men det er noe som kan være ønskelig, spesielt om man lager vindtunneler for kunder og ikke bare til eget bruk. Dette innebærer at man må ha kartlagt på forhånd hvor store de forskjellige delene av vindtunnelen kan være, og fortsatt kan fraktes. I noen tilfeller må også individuelle komponenter av en vindtunnel kunne deles opp slik at de kan flyttes.



Figur 9.3.: Testkammer med overgang til dyse.



Figur 9.4.: Testkammer med overgang til diffuser.

9.1.2. Hva vi har designet etter

Vårt design har et annerledes utgangspunkt. Vi har tatt utgangspunkt i en eksisterende vindtunnel med et design som er mer rettet mot å demonstrere for barn hvordan å justere flappen på vingeprofilen vil heve og senke vingeprofilen i luftstrømmen i testkammeret. Denne vindtunnelen skal også bruke et testkammer med forholdsvis like dimensjoner som det eksisterende, og viften som følger med. Dette betyr også at mye av den elektroniske innmaten som er med fra før også skal brukes. Vindtunnelen må også kunne transporteres innendørs på campus Kongsberg. Dette innebærer at tunnelen må kunne fraktes i ett stykke eller delt opp slik at man kan frakte den inn og ut av måleteknisk lab, og at den må kunne passe i heisene som er i bygget. Den må også ikke veie for mye for heisene.

Dermed måtte man ta utgangspunkt i viften sitt tverrsnittsareal i forhold til testkammeret, og deretter designe utifra det. Dermed vet man også hvor stort det minste tverrsnittsarealet til diffuser og dyse må være for å oppfylle sin funksjon. En ekstra ting som også må designes er en overgang fra testkammeret til diffuseren slik at man får en gradvis overgang fra en firkantet profil til en rund profil som passer utifra ønske om jevn kvalitet på luftstrømmen som skal brukes.

Dysen og manipuleringskammeret må fortsatt designes som tidligere nevnt i Ordinært design og er ellers uforandret.

9.2. Testkammer

For vindtunnelen som vi har sett for oss kan det være best å ha et firkantet testkammer. Dette fordi det da også er enklere å skulle lage dyse for det, og det er lettere å ha et firkantet testkammer med tanke på innfestingen som modellen skal festes i. Rammen som vi har opprinnelig for vindtunnelen er også laget for å bruke et firkantet testkammer. For å bruke et

sirkulært testkammer ville det ha krevd mye ombygging av rammen for å gi det tilstrekkelig støtte.

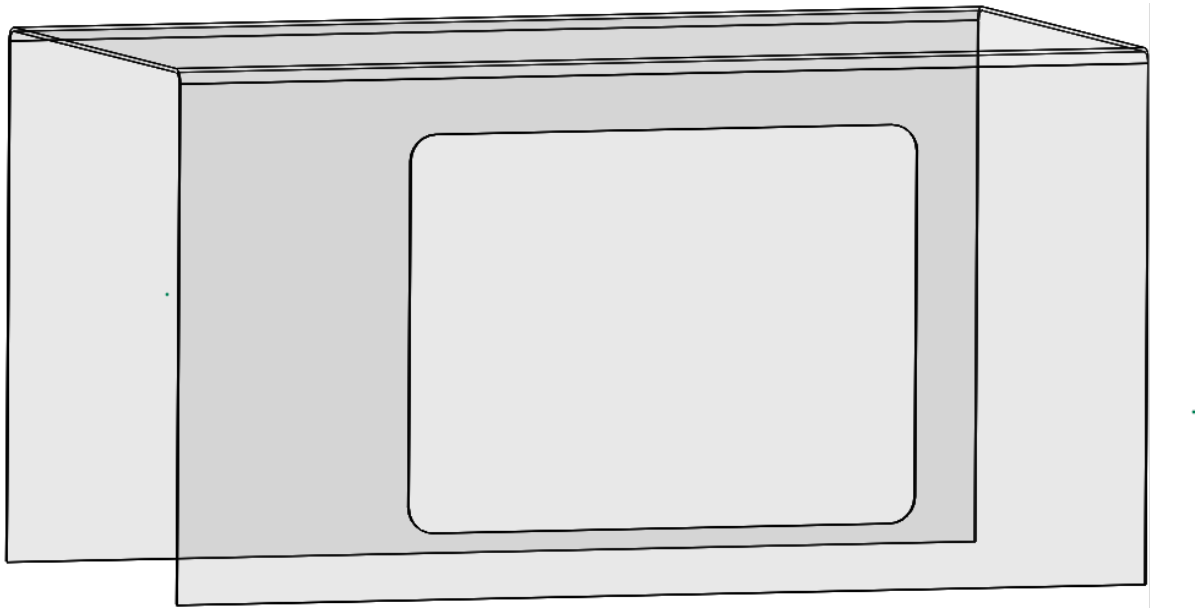
Dette er til tross for at et rundt testkammer vanligvis ([20]) ville gitt oss bedre egenskaper for luftstrøm, men vi har også planer om å ha sensorer, luke og en innfesting som det er lettere å få jobbet med om kammeret er firkantet.

I utgangspunktet ønsket ikke oppdragsgiver et større testkammer slik at dette blir toneangivende for resten av det videre designet.

Kammerets ytre dimensjoner i dag er 0.415 m høyde, 0.415 m bredde og 0.66 m lengde, per krav K.VT 05. Dette gir et tverrsnittsareal på 0.1725 m^2. Beregnet hydraulisk diameter i følge formlen under er 0.415 m, etter formelen fra [34].Den hydrauliske diameteren til kammeret er en tilnærming i fluidmekanikk for å skulle tilsvare hvor stort rørstrømningen går gjennom når man har en åpning som ikke er rund. Dette er viktig fordi mesteparten av teorien i fluidmekanikken antar at man har et fluid, i vårt tilfelle luft, som skal gjennom et rundt rør.

$$D_h = \frac{2 \cdot a \cdot b}{a + b} = \frac{2 \cdot 0.415 \cdot 0.415}{0.415 + 0.415} = 0.415\,\text{m} \tag{9.1}$$

Det indre målet som vi har gått videre med i design er på 0.40 m og er bredden og høyden på testkammeret. Ifølge dokumentasjon [20] bør et testkammer sin lengde være minst 0.5 ganger og maksimalt 3 ganger den hydrauliske diameteren. Dette er respektivt minst 0.2075 m og maksimalt 1.245 m, og oppfylles allerede i dag med nåværende lengde på testkammeret på 0.66 m. Dette er fordi luften som kommer ut av dysen trenger en lengde tilsvarende 0.5 ganger hydraulisk diameter for å få en uniform gjennomstrømning. Og hvis lengden på testkammeret er lenger enn 3 ganger hydraulisk diameter kan man få et økt grensesjikt for luften ved utgangen av testkammeret. Den lengden som har blitt valgt for testkammeret med overgangen fra



Figur 9.5.: Testkammer sett fra høyre.

dysen er 0.815 m siden man da ender opp med en lengde som er litt over 2 ganger hydraulisk diameter.

Grensesjiktet for luften er overgangen fra null hastighet ved veggen og en gradvis overgang til lufthastigheten som den frie luftstrømmen har. Hvis testkammeret er for kort vil dette kunne forstyrre den ønskede uniforme flyten gjennom testkammeret og bidra til ujevne vindhastigheter. Dette igjen gir ujevne testresultater under gjennomføringen.

Testkammeret i dag har litt avrunding i hjørnene hvor det er pleksiglass. Men den mangler tilsvarende avrunding ved overgangen til stålbunnen testkammeret er festet nedi. Avrundingen er nødvendig for å hjelpe kammeret med å tilnærme seg et sirkulert areal for å unngå øking av grensesjiktet [20] og turbulens som gjerne oppstår som følge av skarpe hjørner. Dette tilsier også at vi burde ha en pleksiglass bunn som er som resten av testkammeret for mer uniform flyt. Samtidig burde vi ha gått for avrunding av hjørnene med 45 graders innvendige kantavslipninger for å minske friksjon og turbulens. Dette kan forbedre luftkvaliteten som følge av designet i forhold til hvordan testkammeret var opprinnelig. Plexiglassbunn har ikke kommet

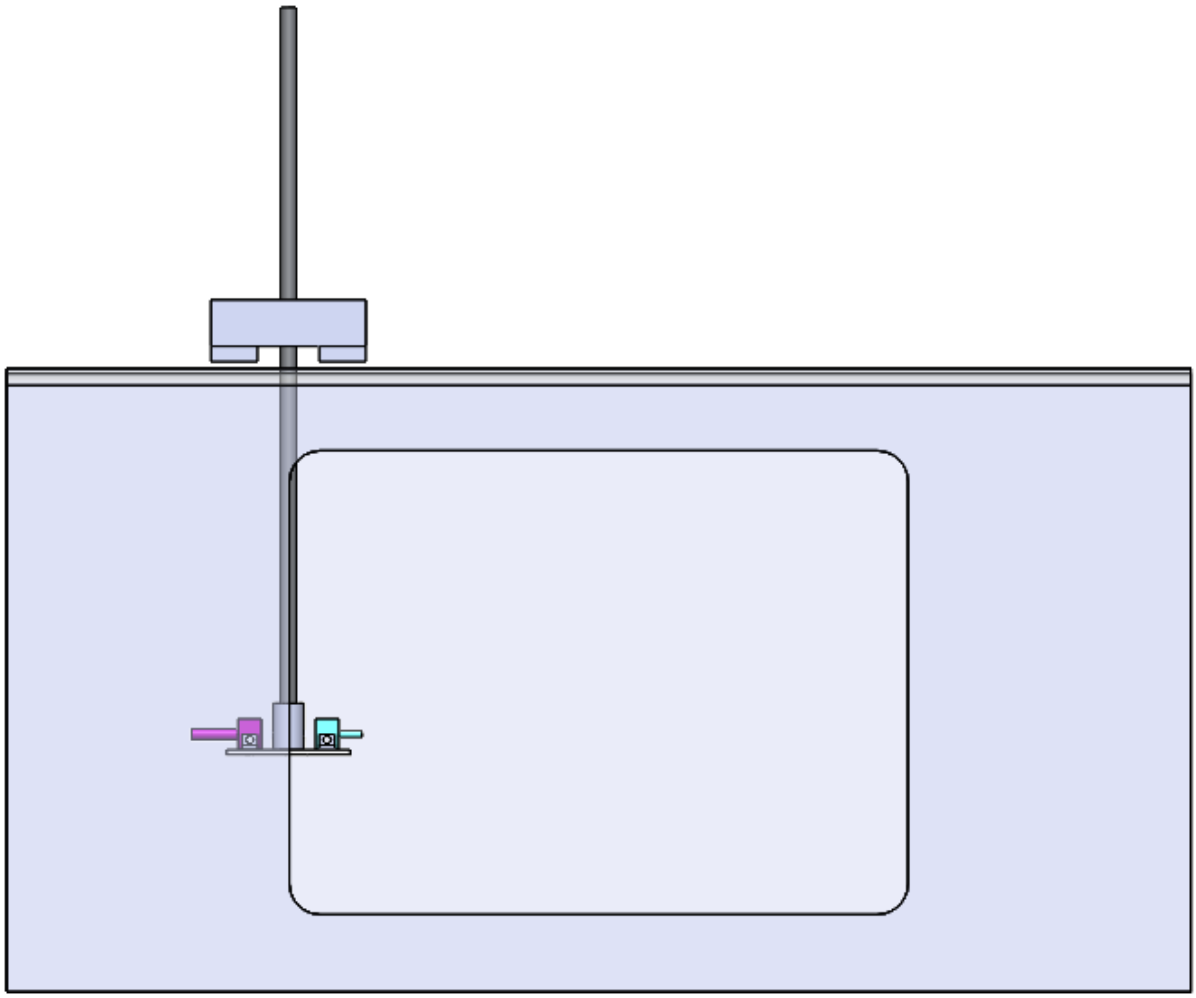
med i vårt endelige design.

I bunnen av testkammeret i dag er det også et hull til den gamle innfestingen som har ca. 22 ganger større areal enn stangen som skulle inn der opprinnelig. Dette bidrar til at det er vanskelig å få en uniform luftflyt gjennom tunnelen siden mye luft kan bli dratt inn der uten å ha gått gjennom dyse eller manipuleringskammer. Dette tilfører luft som ikke blir rettet ut i forhold til mesteparten av luften på vei gjennom vindtunnelen. Et mer ideelt design av testkammeret burde være mer uniformt og ikke ha et så overdimensjonert hull til innfesting av modellen i bunnen. På utsiden av hullet bør det være en pakning mellom innfestingen og bunnen som slipper minimalt med luft inn. En mindre permanent løsning for å tette igjen er å bruke gaffateip for å hindre mye av den uønskede luften å komme inn i tunnelen på den måten.

9.2.1. Fartsmåling foran testmodell

Vi diskuterte å ha en fartsmåler foran åpning foran tiltenkte målesnitt slik at man kan skanne målesnittet foran modellen for å kartlegge lufthastigheten i forhold til kravene K.KM 05 og K.KM 01.01. Denne senkingen innebar at man senket en stang med en platform ned i kammeret. Men for å gjøre det ville de ha vært fordelaktig å ha mest mulig mekanikk og bevegelige deler utenfor testkammeret. Dermed trengs det en åpning slik at man kan føre inn sensorene som skal brukes til å måle, og samtidig må denne åpningen holdes tettest mulig slik at man forstyrrer luftflyten minst mulig.

Det er veldig vanskelig å skulle gå for en slik løsning siden denne kan føre til at man får mye falsk luft inn i testkammeret på grunn av luften som da ville blitt trukket inn gjennom en annen åpning enn der man har forsøkt å få gjennomstrømningen til å bli laminær. Dette ville påvirket kvaliteten på luftstrømmen gjennom. Denne ideen har blitt forkastet etter presentasjon 2 og kravet kan dermed ikke oppfylles i denne oppgaven. Dette må eventuelt jobbes med av en fremtidig gruppe.



Figur 9.6.: Mobil fartsmåler foran tenkt plassering for modell på platform.

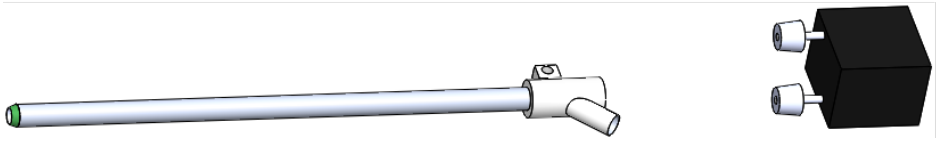
9.2.2. Stasjonær fartsmåling for tunnelen

Det trengs også en fartsmåler for vindtunnelen slik at man har en anelse om hva vindhastigheten gjennom tunnelen er, uavhengig om det er luften rett foran modellen eller ikke. Det er denne måleren som trengs for å regulere vindhastigheten på en god måte slik at man kan regulere vindhastigheten.

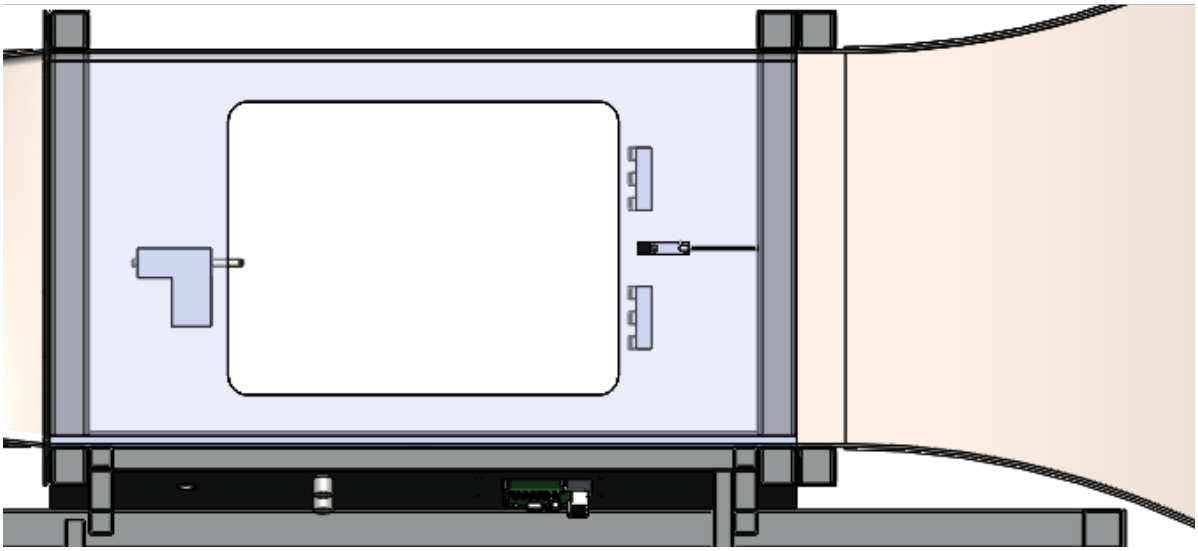
Denne fartsmåleren er i form av en enkel trykkmåler med pitotrør som konfigureres slik at den kan regne om trykkforskjeller mellom statisk og dynamisk trykk slik at man kan avlese vindhastigheten.

9.3. Valg av innfesting

For å kunne gjennomføre eksperimenter med en modell i en vindtunnel må man kunne holde fast modellen, (noe som nesten nevnes i krav K.TM 03). Ellers ligger bare modellen på bunnen av testkammeret. Den kan da om den er lett nok beveges av vinden og man kan ikke få resultater av eventuelle målinger av krefter på modellen. Innfestingen til modellen må også være solid nok til at den tåler kreftene fra vindhastigheten og ikke begynne å bøye seg sånn at man får unøyaktige målinger per krav K.TM 03. Hvis innfestingen beveger seg under drift



Figur 9.7.: Stasjonær fartsmåler med pitotrør.

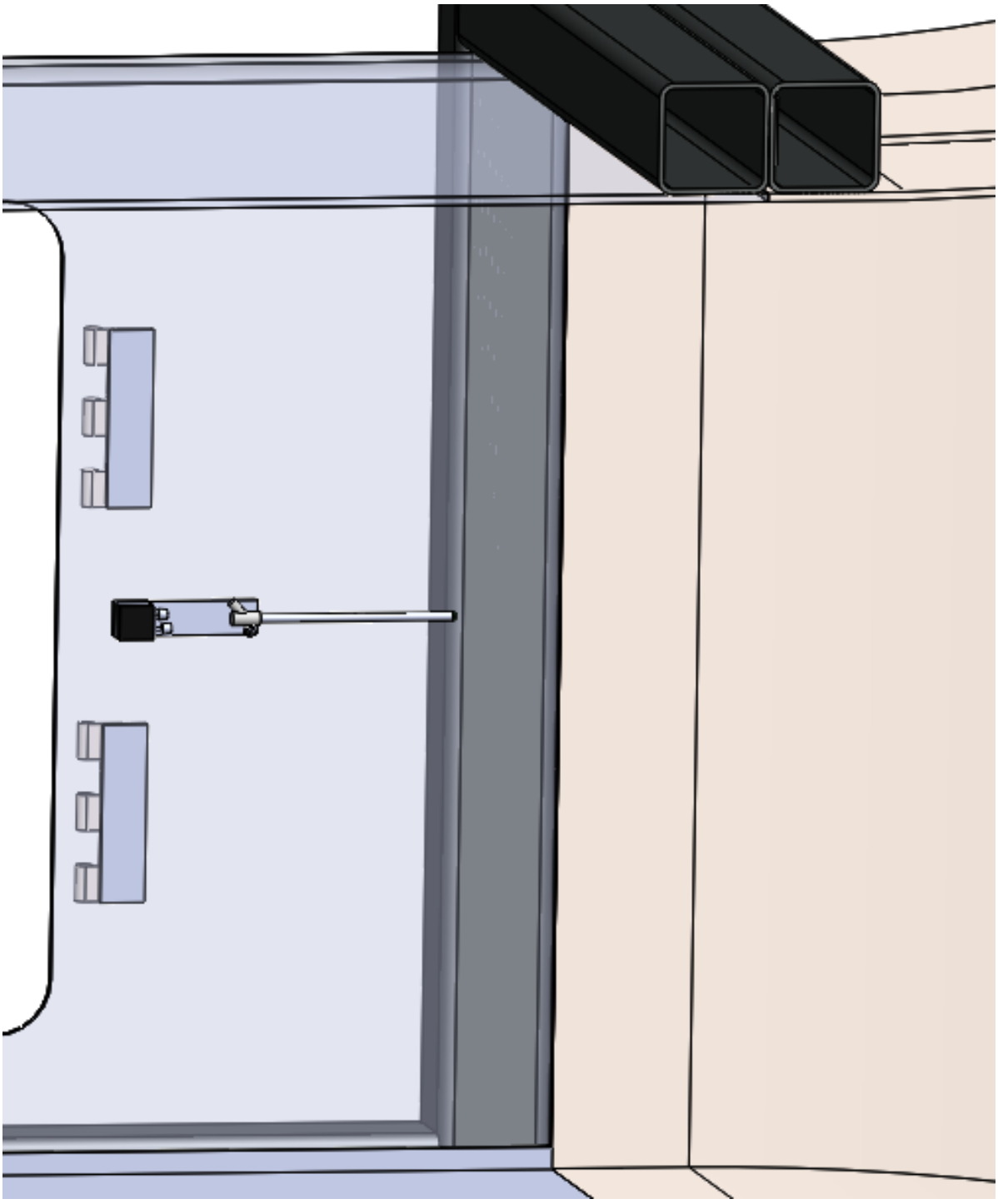


Figur 9.8.: Stasjonær fartsmåler plassert i testkammer.

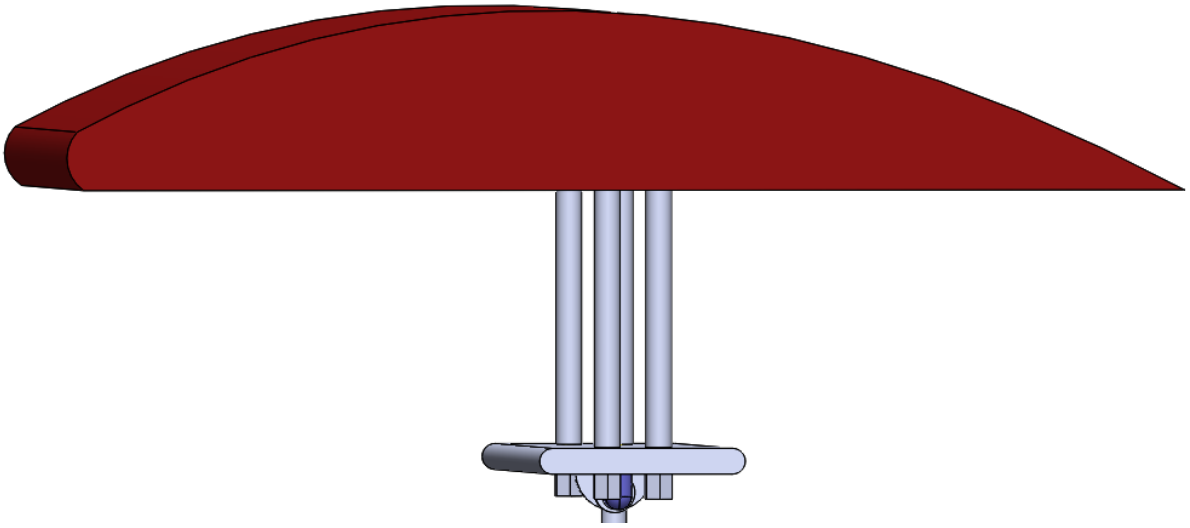
vil man kunne få ujevne og sannsynligvis ukorrekte målinger i sensorene som er koblet til innfestingen.

Innfestingen har to kjernefunksjoner som må nevnes. Per krav K.KM 08 skal den holde fast testmodellen og muliggjøre måling av kreftene som modellen opplever i form av luftmotstand på modellen (drag) og hvor mye modellen eventuelt løftes opp (lift). Dette er de to nøkkelfunksjonene som er hensikten med vindtunnelen. Å kunne måle kreftene modellen utsettes for ved forskjellige lufthastigheter. Ved å måle kreftene som modellen blir utsatt for kan man ved kombinasjon av kjent lufthastighet regne ut et estimat for lift- og dragkoeffesient. Disse to verdiene må finnes eksperimentelt selv om man kan få et anslag av dette med en simulering.

Innfestingen var også planlagt å ha to funksjoner til. Man burde kunne heve og senke innfestingen slik at man kan plassere modellen best mulig i vindstrømmen gjennom tunnelen per krav K.TM 04. Men på grunn av at vi har gått tom for tid med tanke på videre systemarkitektur, rekker vi ikke å jobbe mer med krav K.TM 04. Den andre funksjonen som har blitt

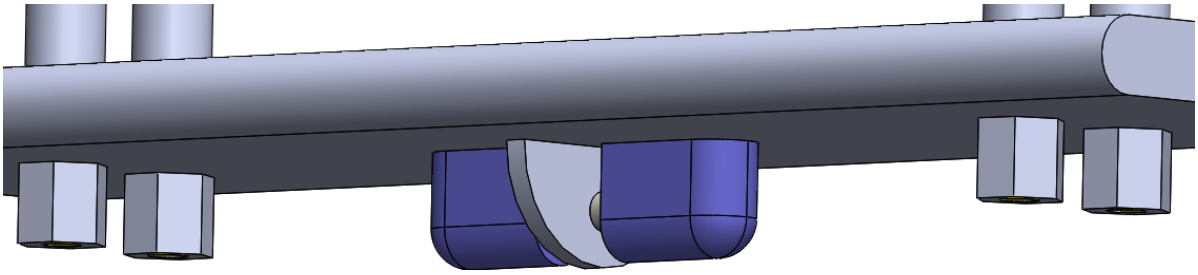


Figur 9.9.: Stasjonær fartsmåler plassert inni testkammer.



Figur 9.10.: Testmodell med innfesting.

etterspurt er at man skal kunne endre vinkelen modellen har i forhold til vinden. Dermed forandrer man også vindens angrepsvinkel som også er kjent som pitch per krav K.TM 05. Disse to funksjonene som har blitt nevnt bør kunne bli styrt fra utsiden av vindtunnelen ved hjelp av kontrollinterfacen som har blitt satt opp.



Figur 9.11.: Stepmotor under festeplate som brukes til å forandre pitch.

Pitchforandringen er tenkt at skal gjøres med en liten step-motor som har høy torque og lav hastighet slik at modellen ikke burde ha uønskede forandringer i pitch mens testmodellen utsettes for luftstrømmen gjennom tunnelen. Det at motoren har høy torque vil si at motoren må bruke mye og kan stå imot mye krefter før den beveger seg, og at det burde bli jevn forandring når man forsøker å endre pitchvinkel. Løsningen vil også kreve at vi må også kunne måle hva pitchvinkelen til modellen er både før, under og etter bruk med vind slik at man vet hva slags

angrepsvinkel modellen har i henhold til krav K.M 07.

Innfestingsmetoden som har blitt valgt i konsept 2 er bolter med et oppsett som da er tenkt at skal holde modellen på plass. Det var opprinnelig tenkt at det ville være nok med et sett festebolter som da har skruefester på seg som så skrus inn i testmodellen. Disse stengene må også ha et godt skruemønster nederst slik at man kan bruke muttere til å feste dem stramt nok under festeplaten.

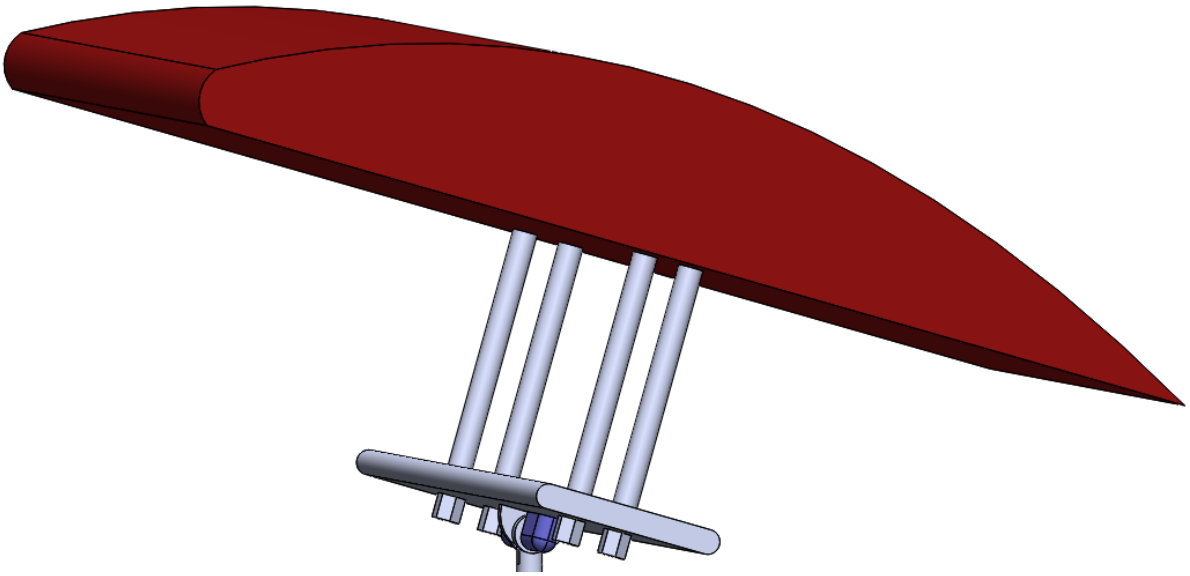
Som følge av innfestingen som er designet må nye modeller som skal testes i tunnelen tilpasses slik at de kan skrus fast i festestengene som er sammen med innfestingen.

Dette festet som vist i Figur9.12 må ikke å være mulig å åpne opp med bare hendene, og vil dermed trenge bruk av verktøy. Det tenkte verktøyet burde være en fastnøkkel som passer med mutterene. Dermed kan man få best mulig grep på skruene og jevn slitasje fremfor en skiftenøkkel som kan gi uønsket slitasje. Dette betyr at at man må ha mulighet til å komme til på alle sidene av modellen for å kunne montere og demontere festestengene. Dermed er det tenkt at festestengene kan tas løs fra selve festeplaten slik at de og modellen kan tas ut av kammeret festet sammen.

9.4. Kraftrigg

Etter modellen har blitt festet til innfestingen må man også få avlest kreftene modellen utsettes for. For å gjøre dette trenger man en rigg hvor man kan få overført kreftene til sensorene som skal lift og drag. Denne riggen har også blitt festet rammen under testkammeret og tenkt sveiset fast i vårt design.

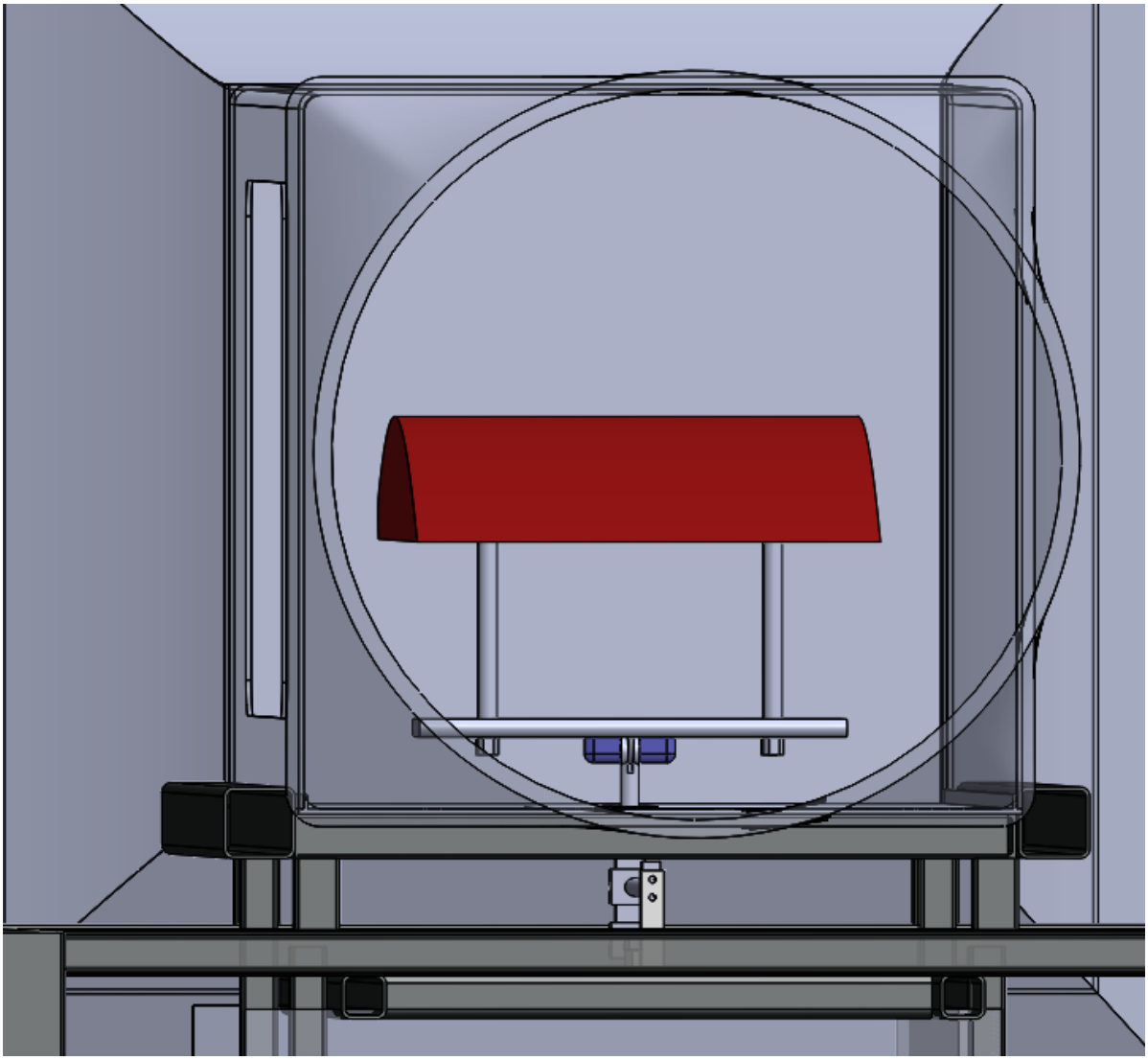
Sensorene er festet til en ramme slik at de skal kunne gi maksimalt utslag og være mest mulig følsomme. Hvordan sensorene fungerer kan du leser mer om i 8.1 i delkapittelet om sensorer. Tanken er at man skal feste kraftriggen til rammen under testkammeret på en slik måte at det



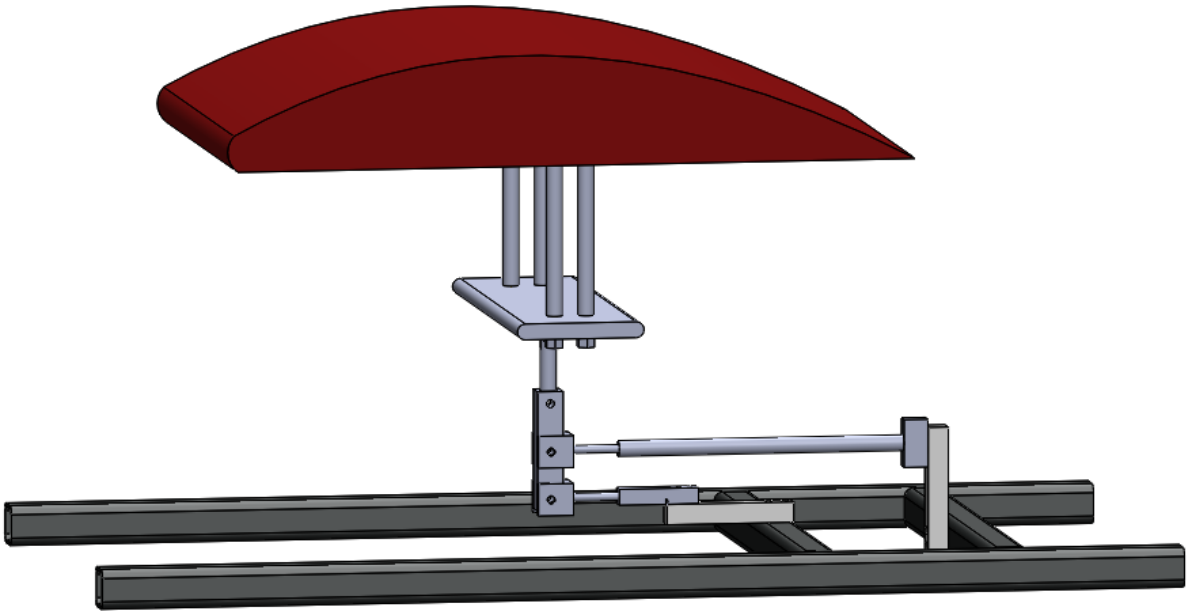
Figur 9.12.: Testmodell med forandret pitch

holder seg mest mulig i ro og ikke kan bidra til falske utslag for sensorene. Selve rammen er enkelt konstruert med stålprofiler som er satt opp for å holde på plass kraftsensorene og dermed også innfestingen som sensorene er festet fast i. Sensorene har et spesielt trekk ved seg som også må designes for. På den enden der man skal belaste sensorene har de skruehull for 5mm diameter skruer, mens på den enden der de skal festes har de skruehull med 4mm diameter. Dette må man være obs på under designing av kraftrigg og andre stenger involvert med sensorene.

Mellom innfestingen og sensorene er det først en stang som leder til en firkantet stålprofil som overfører belastninger fra drag og lift via stålbraketter til sensorene. Kraftoverførins-stengene fra brakettene er grovt designet og ikke designet som deler som er lett å produse-re.



Figur 9.13.: Plassering av kraftrigg under testkammer sett fra diffuseren sitt synspunkt.



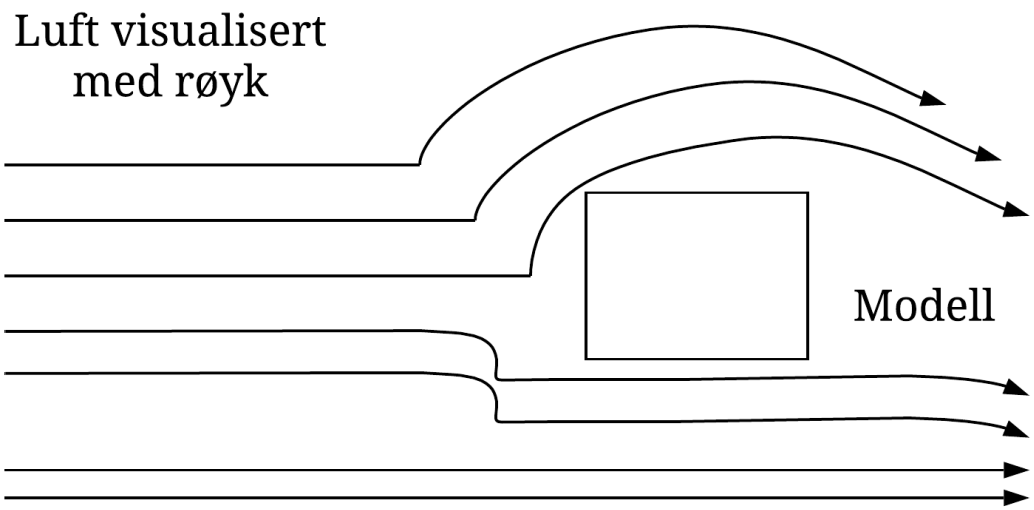
Figur 9.14.: Testmodell med kraftrigg og sensorer.

9.5. Visualisering

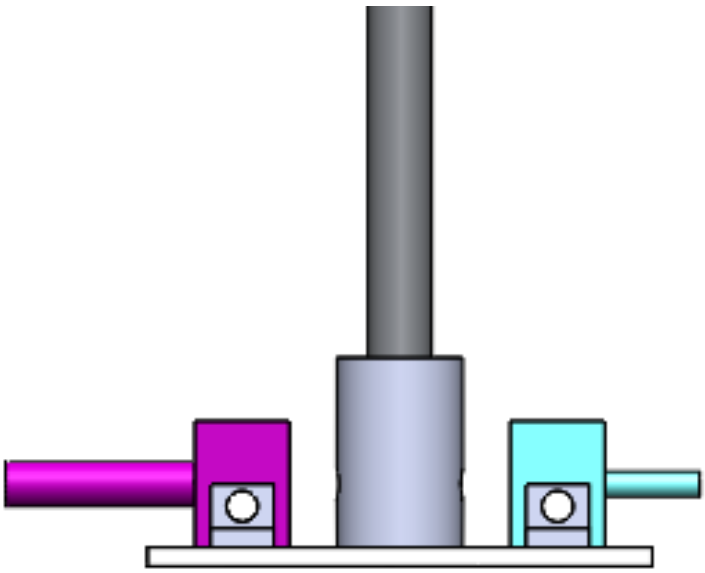
En annen men veldig viktig funksjon med tanke på vindtunnelen som skal brukes i opplæring er visualisering av luftstrømmen, per K.KT 10. Det som menes med dette er at man har en metode for å kunne gjøre luftstrømmen synlig, og da er man spesielt interessert i luftstrømmen som går rundt testmodellen.

Vi har kommet fram til at vi skal visualisere luftstrømmen med en form for røykgenerator. Denne må også sannsynligvis kobles til en røykprobe slik at vi kan fordele røyk i et målesnitt foran modellen i tråd med K.VT 10.03 på en slik måte at vi kan visualisere den delen av luftstrømmen som er interessant for modelltesting. Avhengig av hvordan røykproben og røykgeneratoren henger sammen må det nok også være noe som leder røyken til røykproben, f.eks. en slange eller et tynt rør.

Funksjonen som vi skal oppnå er å visualisere luftstrømmen gjennom vindtunnelen, da med fokus på vinden som går langs testmodellen.



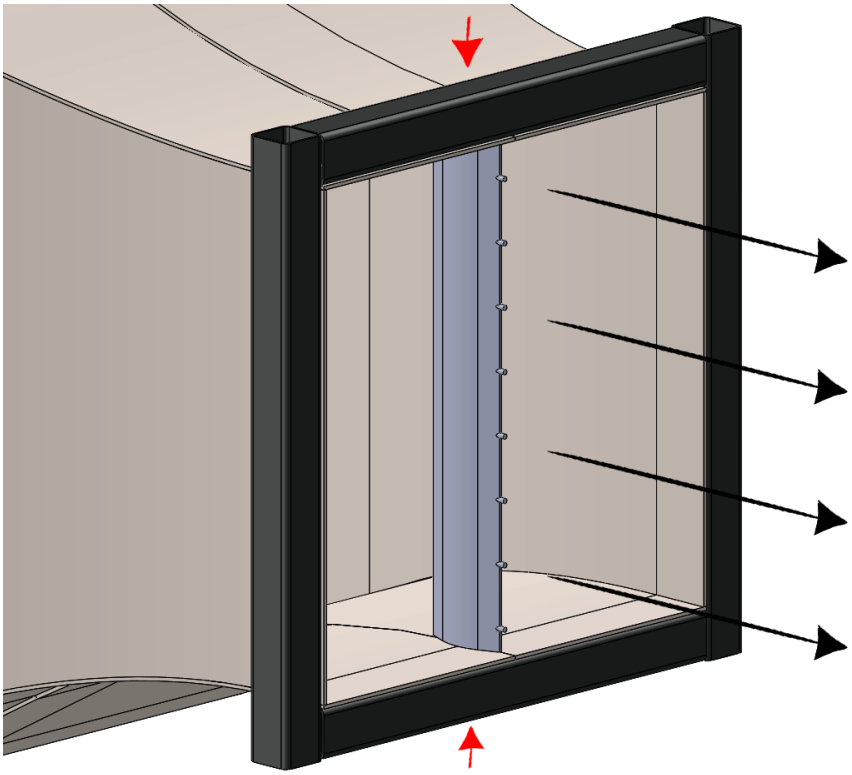
Figur 9.15.: Visualisering av en luftstrøm med røyk som passerer en modell.



Figur 9.16.: Illustrasjon av design med røykproben er til høyre på platformen (blågrønn), farts-
måler til venstre (lilla).

Den gjeldende designtanken var at man kan føre røykproben (se figur 9.16) på den samme plattformen som man har fartsmåleren på. Dermed har man god bevegelsesfrihet til å kunne styre røykfordelingen på testmodellen. En av grunnene til dette er hvordan røyk oppløses i luft og må dermed være nært nok at man kan visualisere hva det er som foregår.

En annen potensiell løsning vil være å introdusere røyken i enden av dysen, rett før testkammeret. Her er allerede luftstrømmen laminær. Det vil være et godt utgangspunkt for å bringe røyken uforstyrret frem til modellen, men krever også at selve komponenten som tres inn i strømmen ikke ødelegger luftkvaliteten. Et tenkt design på denne stasjonære komponenten er illustrert i figur 9.17.



Figur 9.17.: Tenkt komponent for introduksjon av røyk plassert ved dysens utløp. Piler i rødt viser 2 alternativer for hvor den eksterne røyken kommer inn.

Dette kan hjelpe med undervisningen i fluidmekanikk som dette er tiltenkt bruk i, siden det vanligvis er vanskelig å se luft. Hadde dette vært væske som skulle visualiseres hadde det vært

en del enklere, siden da kunne man bare ha brukt fargemiddel.

Dette er også en av funksjonene som oppdragsgiver har etterspurt siden dette skal brukes i stor grad til opplæring i faget fluidmekanikk, og det er viktig å kunne vise visuelt hva det er som foregår gjennom en luftstrøm og hva som skjer når den møter en testmodell.

9.6. Dyse

9.6.1. Dysens funksjon

Dysen er en trakteformet komponent som har som oppgave å endre hastigheten i et fluide. I vår vindtunnel vil dysen akselerere strømmen inn til testkammeret.

9.6.2. Utvikling

Ut fra kriterier og restriksjoner dimensjonerer man først et testkammer. Ønsket størrelse fungerer som et grunnlag for hvert vindtunneldesign da øvrige komponenter må designes proporsjonalt.

På lik linje med mange andre kilder har vi valgt å følge Mehta og Bell's anbefalinger [22]. Der henvises det til kriterier som skal bli møtt for at deres konklusjon er gyldig.

"**Until further data are obtained, all these conclusions should be confined to small (test section area $\leq$ 0.5 m^2), low-speed (U $\leq$ 40 m/s), contractions with area ratios of around eight, exit aspect ratios of about five and length to inlet height ratios of about one.**"

Med unntak av "exit aspect ratio" så innebefinner vårt design seg etter retningslinjene.

9.6.3. Dimensjoner og form

Dysens design starter ved testkammerets innløp hvor luften allerede er akselerert. Dette er dysens utløp og de to vil ha identisk form og størrelse. Tverrsnittet er kvadratisk med dimensjon for høyde/bredde på 400 mm. For å definere dysens innløp baserer man seg på kontraksjonsratio dvs. forhold mellom inn- og utløp.

Kontraksjonsratio mellom 6 og 10 er tilstrekkelig [4] for lavhastighets vindtunneler. Den generelle regelen er at man vil ha så stor ratio som mulig, men med tanke på plassrestriksjoner har vi lagt oss i nedre sjikt med et forhold på ca 6.

Dette gir arealet og sidelengdene på inntak

$$\begin{aligned} A_I = 6 \cdot A_U = 6 \cdot 0.160\ m^2 = 1.033\ m \\ A_U = (0.400\ m)^2 = 0.160\ m^2 \\ L_H = L_B = (\sqrt{1.033\ m^2}) = 1.02\ m \end{aligned} \tag{9.2}$$

hvor A er tverrsnittsarealet og benevnelsene "I" og "U" står for henholdsvis innløp og utløp på dyse.

9.6.4. Lengde

I forbindelse med komponentens lengde (aksiell retning) vil man at forholdet mellom lengde og høyde/bredde skal ligge rundt 1. Et forhold på under 0.667 har vist at tendensen for at strømningen separerer seg fra dysens vegger nært utløpet som gir redusert uniform strømning.

Et forhold på over 1.79 øker faren for at bredden på grenesjiktet vokser. Grensesjiktet er avstanden i hastighetsprofilet fra 0 til laminær strømning. Det vil si fra inntil veggen og en viss avstand ut.

Vårt valg har vært å sette lengden lik høyden/bredden, dvs 1.02 m.

9.6.5. Profil

I følge litteraturen ser vi at ved rektangulære dyser finner man profilet ved å plotte en femtegrads polynomlikning [22]. I figur 9.18 ser man en sammenligning av de forskjellige konturene som er vurdert. Denne viser helt klart at en femtegrads polynomligning er optimal.

Polynomligningen ser slik ut:

$$y(x) = H_i - (H_i - H_e)(6x^5 - 15x^4 + 10x^3)$$

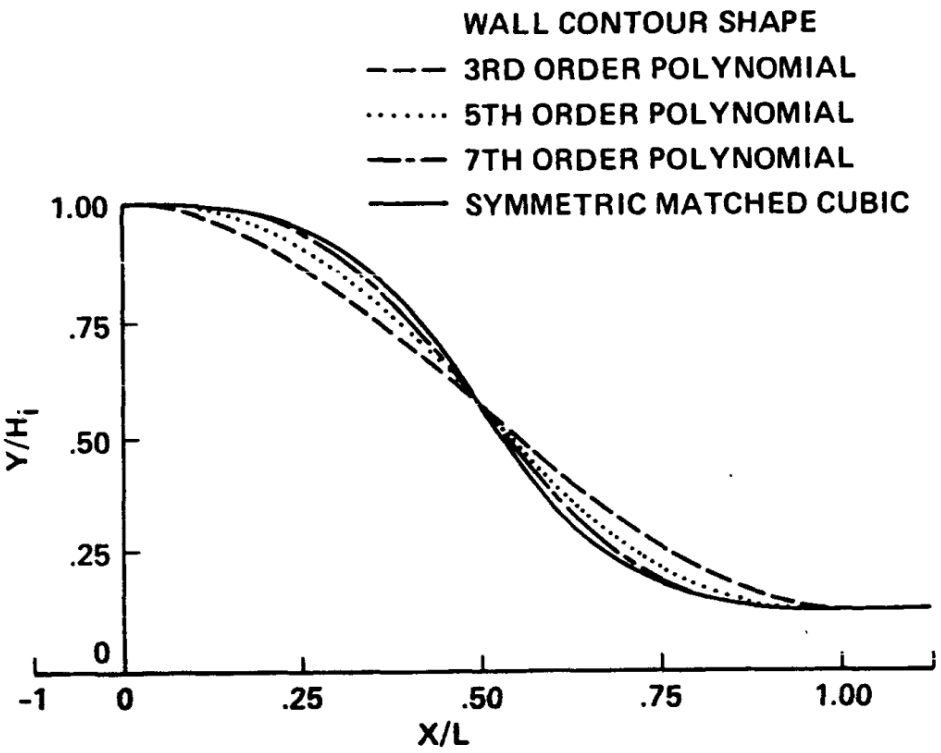
Hvor $H_I = 0.51\,\text{m}$ og $H_E = 0.200\,\text{m}$.

Ved plotting og modellering i SolidWorks får vi en kontur som vist i figur 9.19.

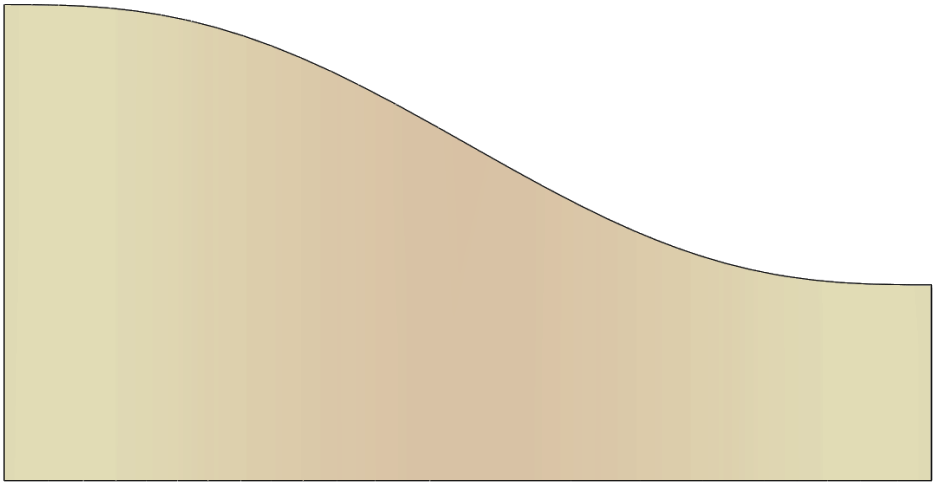
9.7. Manipuleringskammer

9.7.1. Strømretter

Med tanke på forsøk som skal utføres er det viktig å ha oversikt over kvaliteten og parameterne til luften som kommer inn i testkammeret. En strømretter er en av nøkkelkomponentene for å

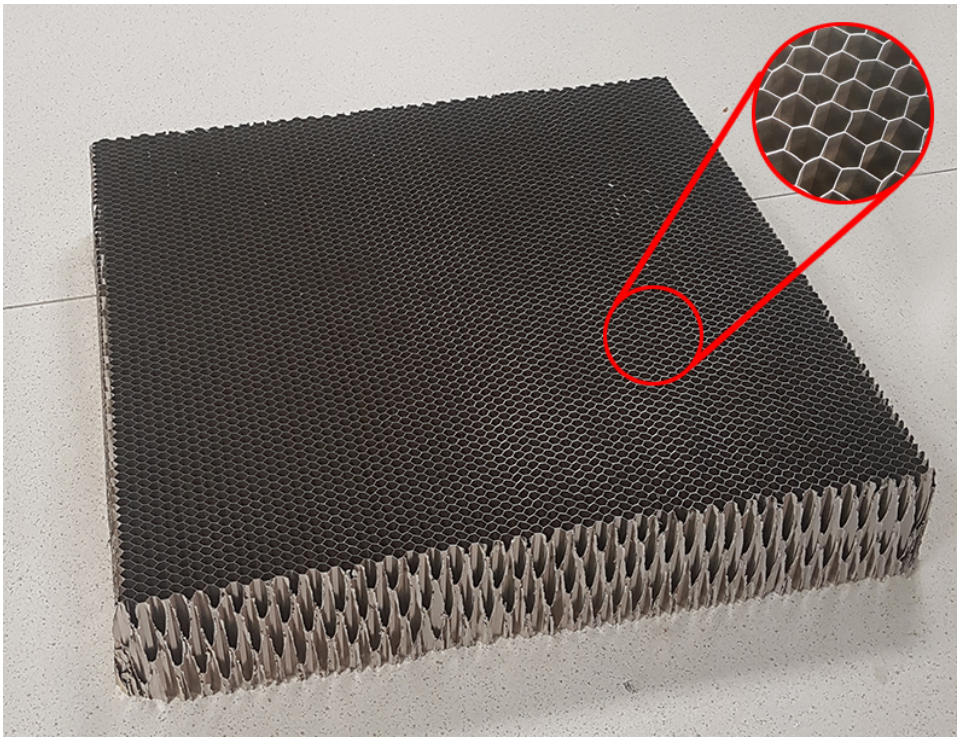


Figur 9.18.: Sammenligning av ulike konturer.



Figur 9.19.: Kontur av dyse fra SolidWorks.

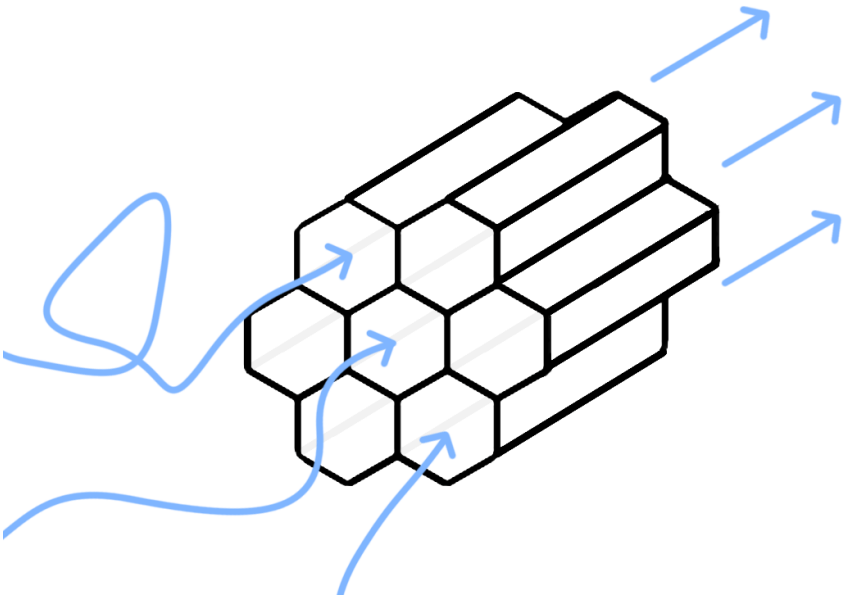
kunne påvirke kvaliteten på luftstrømmen i en vindtunnel. I relevant litteratur blir denne vanligvis referert til som en "honeycomb" pga. dens heksagonale cellestruktur, mens oversettelsen vi har valgt å bruke (strømretter) er mer intuitiv i forhold til dens funksjon.



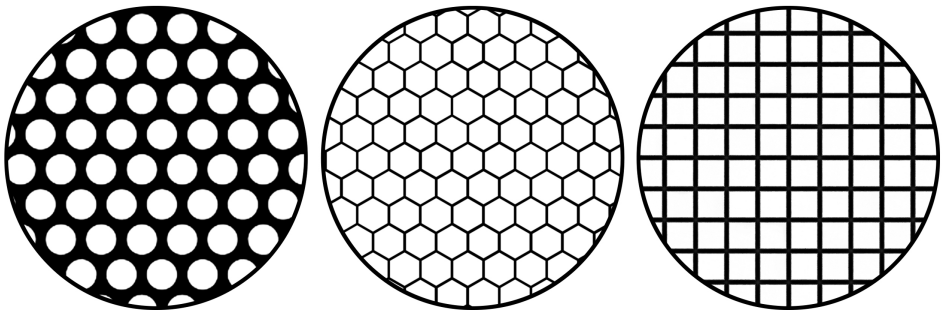
Figur 9.20.: Strømretter fra eksisterende system.

Dette inkluderer da gjerne å redusere turbulensen som presenterer seg i form av virvler. Vifta trekker luften til seg aksialt gjennom hele tunnelen men på grunn av virvlene vil store lommer i lufta også ønske å bevege seg lateralt på en kaotisk måte. Disse oppstår som et resultat av at inntaket i tunnelen trekker lufta fra omgivelsene. Denne turbulensen vil man derfor gjøre noe med før den kommer til testkammeret, og ved hjelp av cellene i en strømretter vil man kunne bryte opp og gjøre store virvler om til flere små. På denne måten vil det ta kortere tid og/eller avstand i tunnelen for virvlene å fullstendig oppløse. Se figur 9.21.

Strukturen i strømrettere kan ha flere former. Disse vises i figur 9.22. Ifølge [2] gir en heksagonal struktur et litt lavere trykkfall enn de andre variantene men viser også til at denne komponenten utgjør under 5% i en typisk vindtunnel.



Figur 9.21.: Strømretteren reduserer variasjonen i lufta laterelt.



Figur 9.22.: Noen strømrettere har også sirkulær eller kvadratisk struktur.

Strømretteren slik den så ut i begynnelsen var heksagonal. Lengden var 75 mm med sider i cellene på 3.75 mm. Et parameter for valg av størrelse av strømrettere er forholdet mellom cellelengde og cellenes hydrauliske diameter, som i vårt tilfelle gir et forhold på

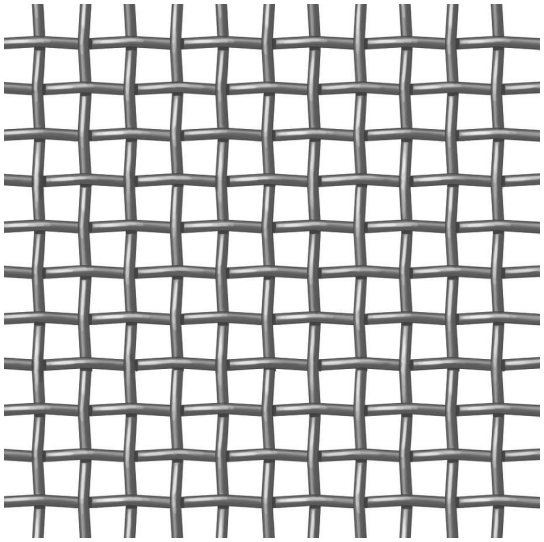
$$\frac{L}{D_h} = \frac{L}{\frac{4 \cdot Areal}{Wet\ Perimeter}} = \frac{75}{\frac{4 \cdot 36.535}{6 \cdot 3.75}} = \frac{75\,\text{mm}}{6.495\,\text{mm}} = ca.11.55 \tag{9.3}$$

Både Mehta og Bradshaw [21] og Barlow et al. [2] beskriver et typisk tall på dette som mellom 6 til 8. Kulkarni et al. [15] er av den oppfatning at basert på simuleringer bør man ikke bruke noe

lavere enn 8 men at det heller ikke er hensiktsmessig og gå noe særlig høyere.

9.7.2. Nettinger

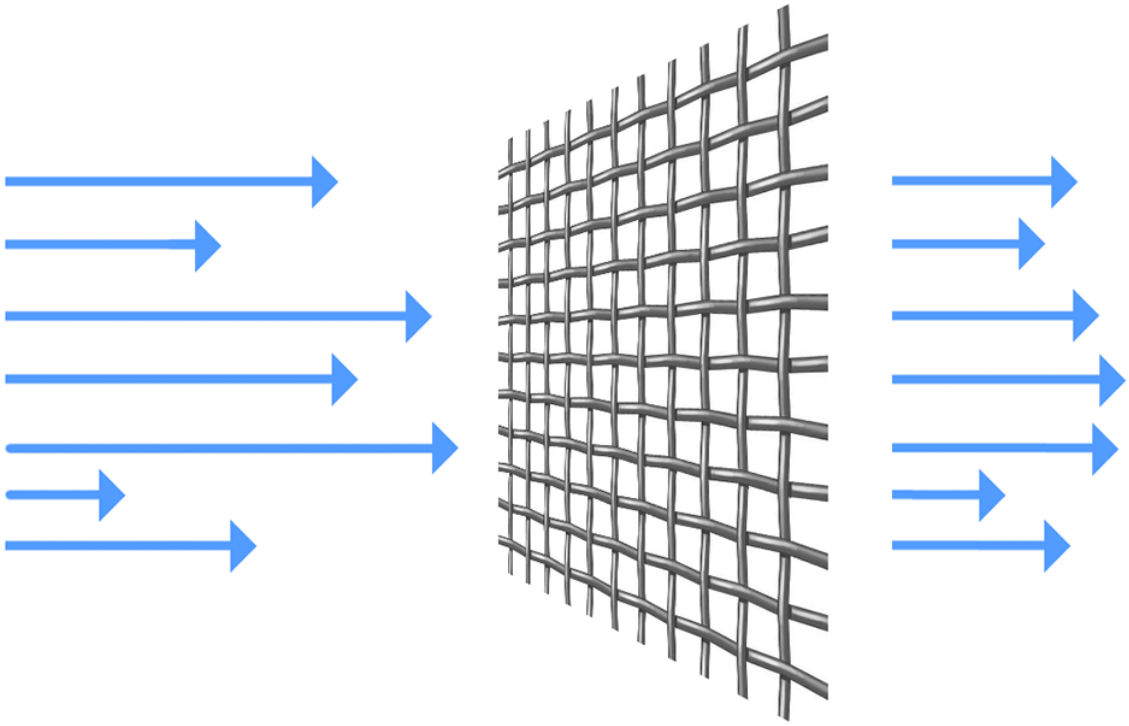
Strømretteren vil være mer effektiv for å redusere lateral turbulens enn aksiell turbulens. For å takle sistnevnte bruker man i tillegg nettinger med forskjellig porøsitet. Disse blir plassert nedstrøm for strømretteren og, om det tas i bruk flere enn én, med gradvis redusert porøsitet (finere mesh). Men for at både strømretter og nettinger skal få maksimal effekt må de plasseres med nok avstand til at reduksjonen i turbulens får tid til å skje. Scheiman [26] og Kulkarni et al. [15] hevder begge at 250 mm er nok mellomrom.



Figur 9.23.: Illustrasjon av netting.

$$F_D = \frac{1}{2} \cdot \rho \cdot u^2 \cdot C_D \cdot A \tag{9.4}$$

Dette er formelen for luftmotstand hvor p er tettheten, u er lufthastigheten, C er motstandskoeffisienten og A representerer arealet.

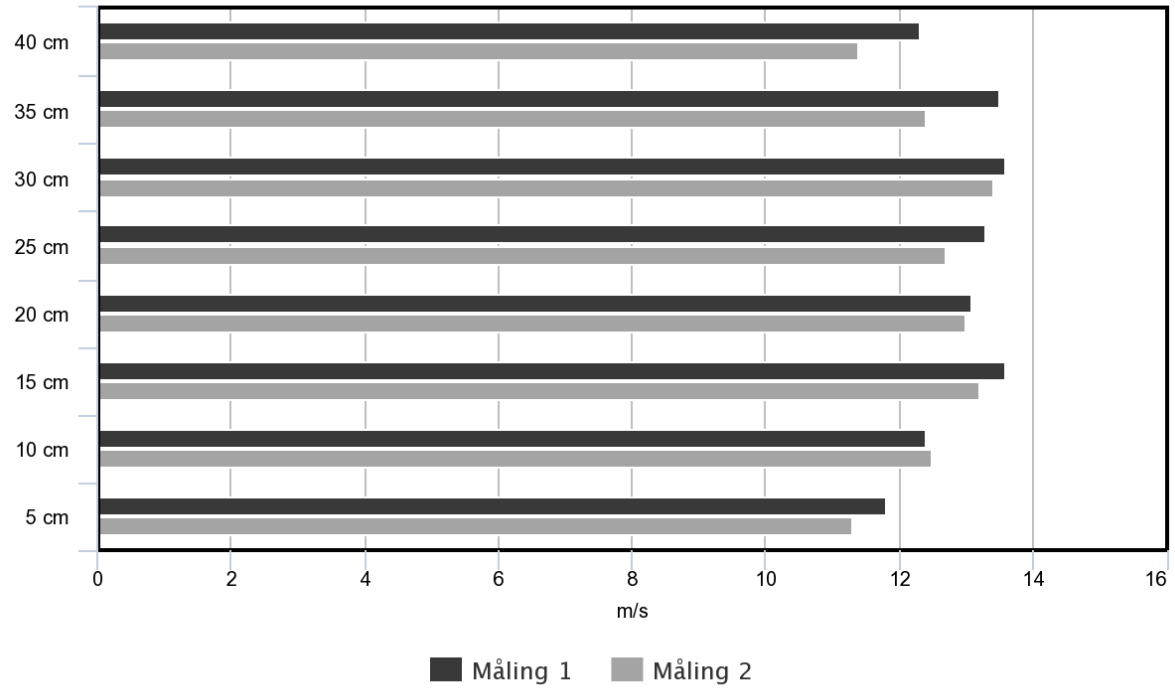


Figur 9.24.: Nettingen reduserer variasjonen i hastighetsprofilet.

Luftmotstanden fra vaierne i nettingen på innkommende luft er proporsjonal med kvadratet av lufthastigheten. Man ønsker at hastighetsprofilen til strømningen som entrer testkammeret er så uniform som mulig og nettingen vil da hjelpe med å redusere de høyere punktene i profilen mer enn de lavere punktene og på den måten redusere variasjonen. Se figur 9.24. Figur 9.25 visualiserer profilen til eksisterende system.

Med tanke på dimensjonering av nettingene mener [22] at basert på diameter av vaier bør reynoldstall ikke overstige 50. Dette er for å unngå fenomenet med "vortex shedding" [16] som essensielt introduserer mer turbulens. Samtidig burde også nettingenes porøsitet ikke være lavere enn 0.57.

I henhold til krav K.VT 02 om at "**det skal være jevn ($\pm 6\,\%$) vindhastighet over målesnittet (snittet der lufta møter testmodellen) i testkammeret**", har det blitt gjort målinger. Bereg-



Figur 9.25.: Hastigheter i senter av tunnel parallelt med strømning, målt fra bunn av testkammer.

ninger basert på disse viser at eksisterende system ikke tilfredstiller kravet og det er derfor nødvendig å gjøre forbedringer.

9.7.3. Beregning for netting

Reynoldstallet i en strømningsituasjon beskriver forholdet mellom treghets- og friksjonskrefter i fluidet. Et eksempel er strømning i et rør; et tall under 2000 fastslår laminær strømning men om Reynoldstallet overstiger 2000 kan man konkludere med at treghetskreftene dominerer og

forholdene vil gradvis bli mer turbulente.

$$Re = \frac{U \cdot l}{\nu} \tag{9.5}$$

Dette er formelen for reynoldstallet hvor U er lufthastighet, l er en karakteristisk lengde og v er luftens kinematiske viskositet ved 22 °.

$$l = \frac{Re \cdot \nu}{U} \tag{9.6}$$

Gjennomsnittshastigheten i det opprinnelige testkammeret er 12.5 m/s. Hastighetene vil skalere i forhold til tverrsnitt så ved bruk av formelen for volumstrøm kan vi regne ut at hastigheten ved nettingenes plassering.

$$U_{MK} \cdot A_{MK} = U_{TK} \cdot A_{TK} \rightarrow U_{MK} = \frac{U_{TK} \cdot A_{TK}}{A_{MK}} \tag{9.7}$$

A er tverrsnittsarealet og benevnelsene "*MK*" og "*TK*" står for henholdsvis manipulerings- og testkammer.

$$U_{MK} = \frac{12.5\ m/s \cdot 0.416^2\ mm^2}{1.020^2\ mm^2} = 2.079\ m/s \tag{9.8}$$

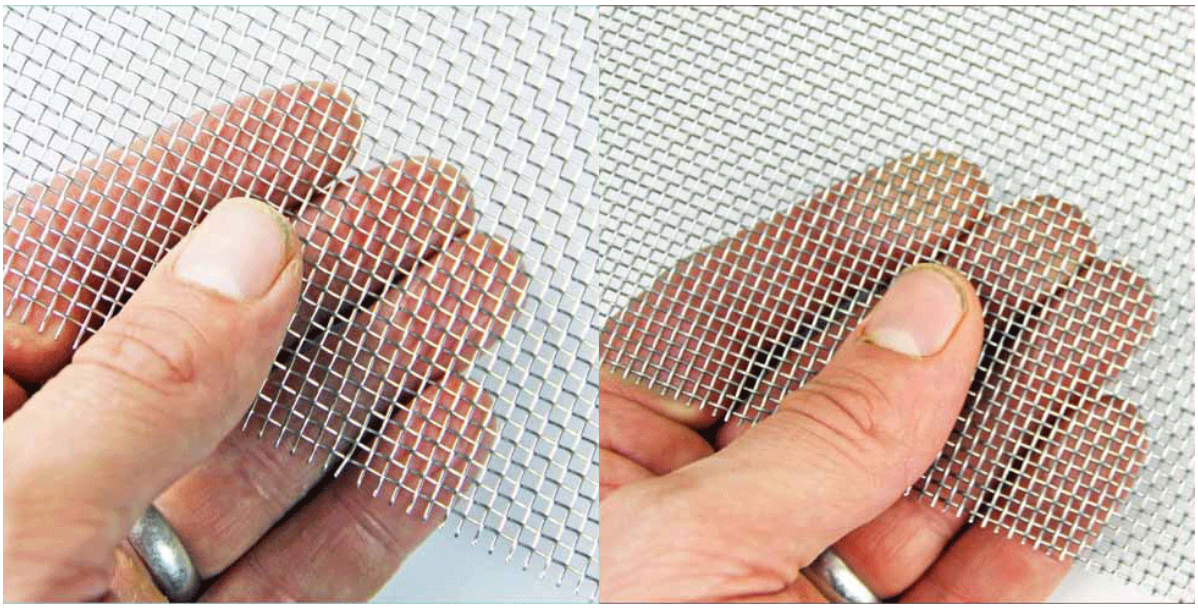
Som nevnt tidligere er vi interessert i et Re-tall på maksimum 50. Hvis vi løser ligningen for den karakteristiske lengden, som i vårt tilfelle er diameteren på nettingens individuelle tråder, får vi

$$l = \frac{Re \cdot \nu}{U} = \frac{50 \cdot 1.516 \cdot 10^{-5}}{2.079\ m/s} = 0.36\ mm \tag{9.9}$$

Basert på denne diameteren, anbefalt porøsitet (over 0.57) og ønsket størrelse (min 1020x1020 mm) begynte vi å undersøke hvor slike nettinger blir solgt. "**The Mesh Company**" har et bredt utvalg men på nåværende tidspunkt er nettingene beskrevet i Tabell 9.1 det beste valget. Alternativet hadde vært å gått betydelig ned i størrelse og fått finere mesh. Konsekvensen av dette er mer oppsamling av skitt og støv i nettingene som krever mer rengjøring.

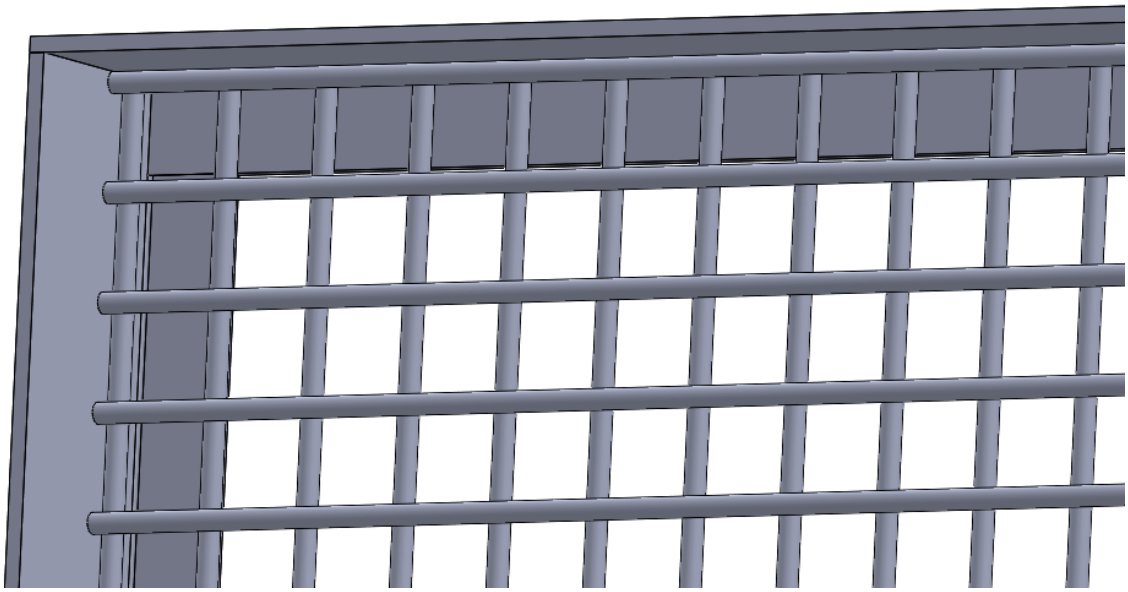
	Diameter	**Hullstørrelse**	**Porositet**	**Materiale**
Netting 1	0.56 mm	1.98 mm	0.61	SS430
Netting 2	0.45 mm	1.67 mm	0.62	SS304

Tabell 9.1.: Karakteristikker for valgte nettinger.



Figur 9.26.: Sammenligning av størrelse på nettinger.

Nettingene må ha langt finere oppløsning enn forventet. Dette gjør det rimelig å anta at for å ikke kollapse/deformeres trengs det rammer som holder nettene stramme. Disse bør være tynne for å ikke påvirke luften som passerer.



Figur 9.27.: Ramme som nettinger festes i. Rammen skrus så fast i kammerveggene.

9.8. Diffuser

En diffuser er en innretning formet for å senke hastighet og å gjenopprette det statiske trykket til et fluid i strømning. Dette bidrar til en jevnere strømning som går mindre støtvis gjennom vindtunnelen som følge av hvor mange ganger viften roterer i sekundet. For å designe en passende diffuser trenger vi først å finne innløpets tverrsnittsareal. Dette er det samme som testkammerets profil, opprinnelig i designet var dette $0.172\,\mathrm{m}^2$ med sidekantene $0.415\,\mathrm{m}$ i høyde og bredde. Etter gjennomgang på hvilket areal som passer med den opprinnelige viften har dette blitt forandret til $0.161\,\mathrm{m}^2$ og med sidekanter med lengde på ca $0.400\,\mathrm{m}$. Utløpet for diffuseren er det enkelt å finne areal for da regelen er at utløpet skal være tilpasset viftens tverrsnittsareal så tett som mulig. På denne måten får man det man trenger størrelsesmessig til diffuseren.

I forbindelse med den eksisterende vindtunnelen er forholdet mellom viftearealet og viften noe for lite, pluss at det er en grop rundt viften slik at luften ikke kommer rett ut ved endekantene men må tvinges over en nesten flat endekant for å komme inn i viften.

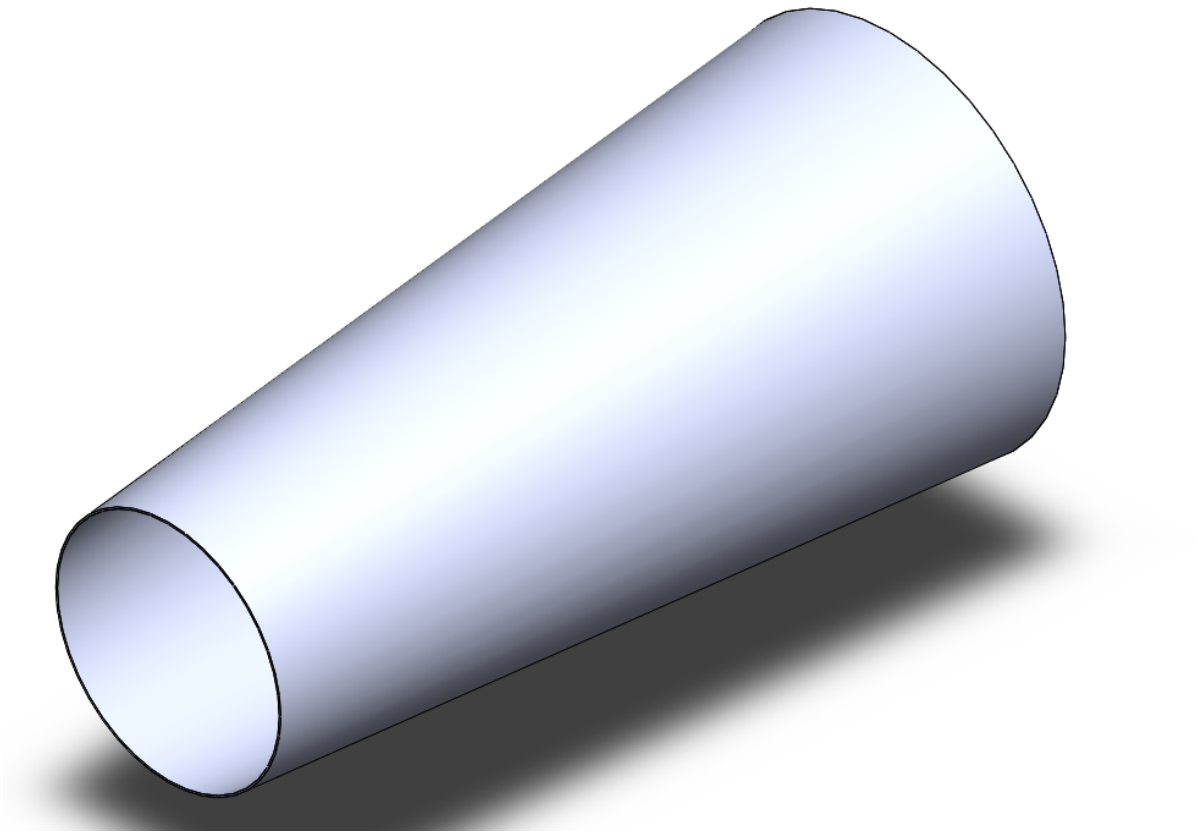
$$A_{\text{inn}} = 0.172\,\mathrm{m}^2$$

$$A_{\text{vifte}} = 0.322\,\mathrm{m}^2$$

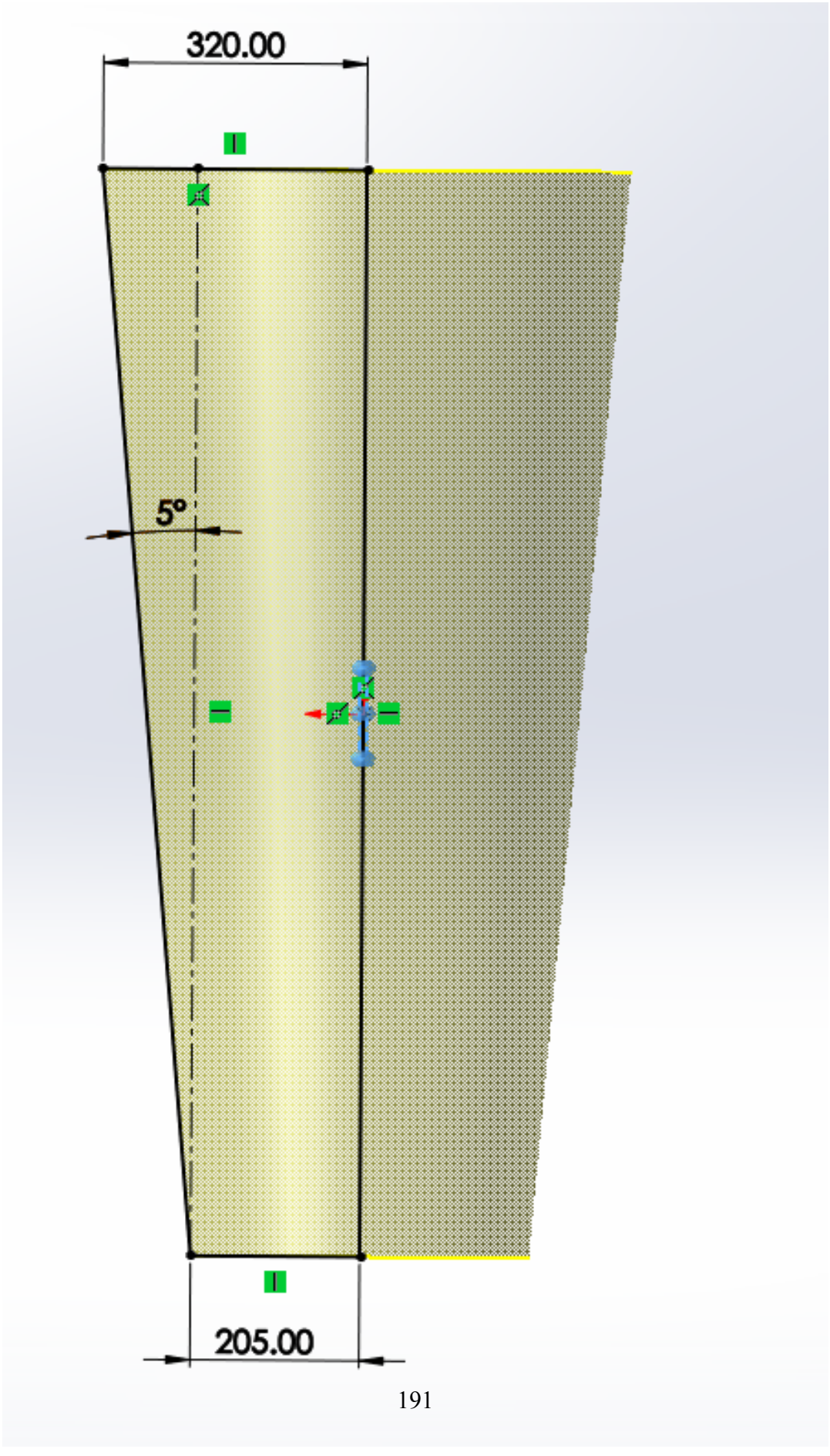
$$\text{Ratio} = \frac{A_{\text{vifte}}}{A_{\text{inn}}} = 1.867$$

9.8.1. Vinkel

Siste designparameter man må tenke på i design av diffuser er den konisk vinkel som igjen er med på å bestemme lengden på diffuseren. Denne vinkelen bør etter litteraturen [25] være maks 5-6 grader.



Figur 9.28.: Diffuser med isometrisk perspektiv.



Figur 9.29.: Diffuser, 5 grader.

9.9. Luke

9.9.1. Kriterier

Bruk av vindtunnelen krever også at man har tilgang til testkammeret. En god løsning til dette kravet betyr at flere viktige kriterier blir møtt.

Først og fremst ønsker man at tunnelen ikke trekker inn luft fra andre steder enn inntaket i front, da dette vil påvirke luftkvaliteten. Dette må også et lukedesign ta hensyn til og det er derfor tenkt at det må brukes gummipakninger.

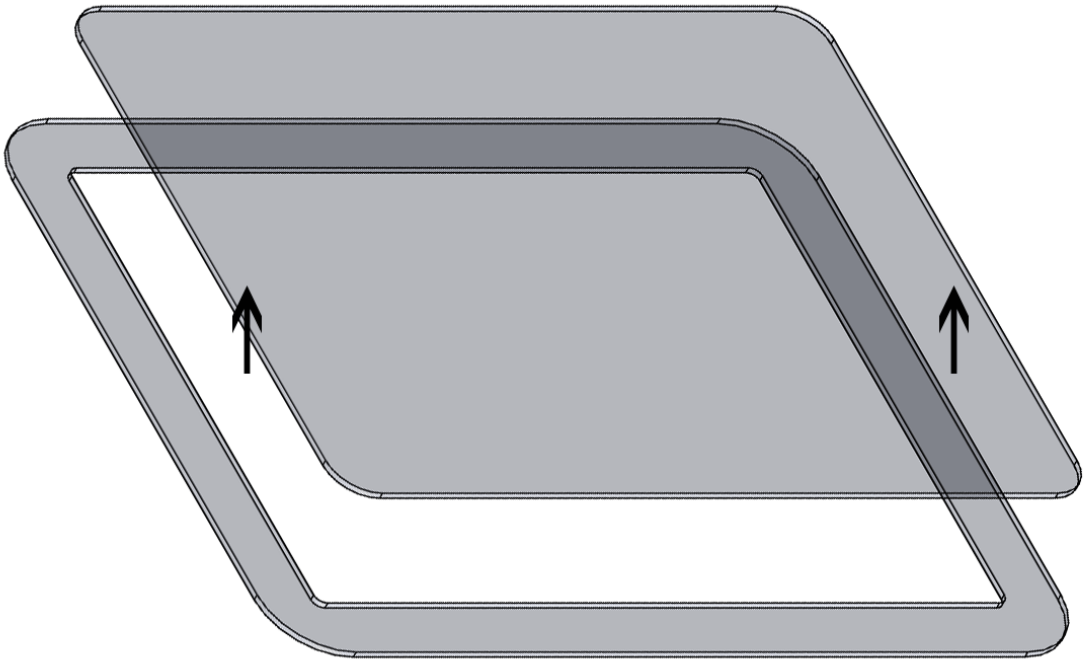
Størrelsen på åpningen og luka må skaleres i forhold til størrelsen på forventede modeller slik at disse kan tres inn, samt at åpningen gir god plass til å jobbe inne i testkammeret når modellene skal festes. Det er også greit at selve luka ikke blokkerer for innsyn i testkammeret i forbindelse med visualisering på modellene. Det er forøvrig tenkt at gjennomsiktig plexiglass er et fornuftig materiale for testkammerets vegger og tak, på lik linje som testkammeret står i dag. Derfor er det også fornuftig at luka består av samme materiale.

Når man skal ha tilgang til testkammeret er det brukervennlig at luka kan åpnes slik at den hverken blir til bry eller trenger hjelp med å forbli åpen. Det er også sikkerhetsfunksjonalitet knyttet til løsningen i form av at vifta kun skal kunne kjøres når kammeret er lukket. Til dette er det tenkt at en sensor skal plasseres ved lukas lås.

9.9.2. Løsning

For å ikke bryte opp symmetrien i testkammeret, og derav få usymmetriske strømninger, ønskes det at innsiden av luka går i ett med resten av veggen i testkammeret. Selve testkammeret vil bestå av en hel plate i plexiglass med to 90°hjørner. Tanken er derfor at før platen blir bøyd

skjærer man ut ønsket åpning med laserkutter. Denne profilen limes så til et annet laserkuttet profil med dimensjoner som overlapper åpningen både innover og utover (Figur 9.30). Dette gjør at innsiden vil gå i ett med resten av kammerveggen.

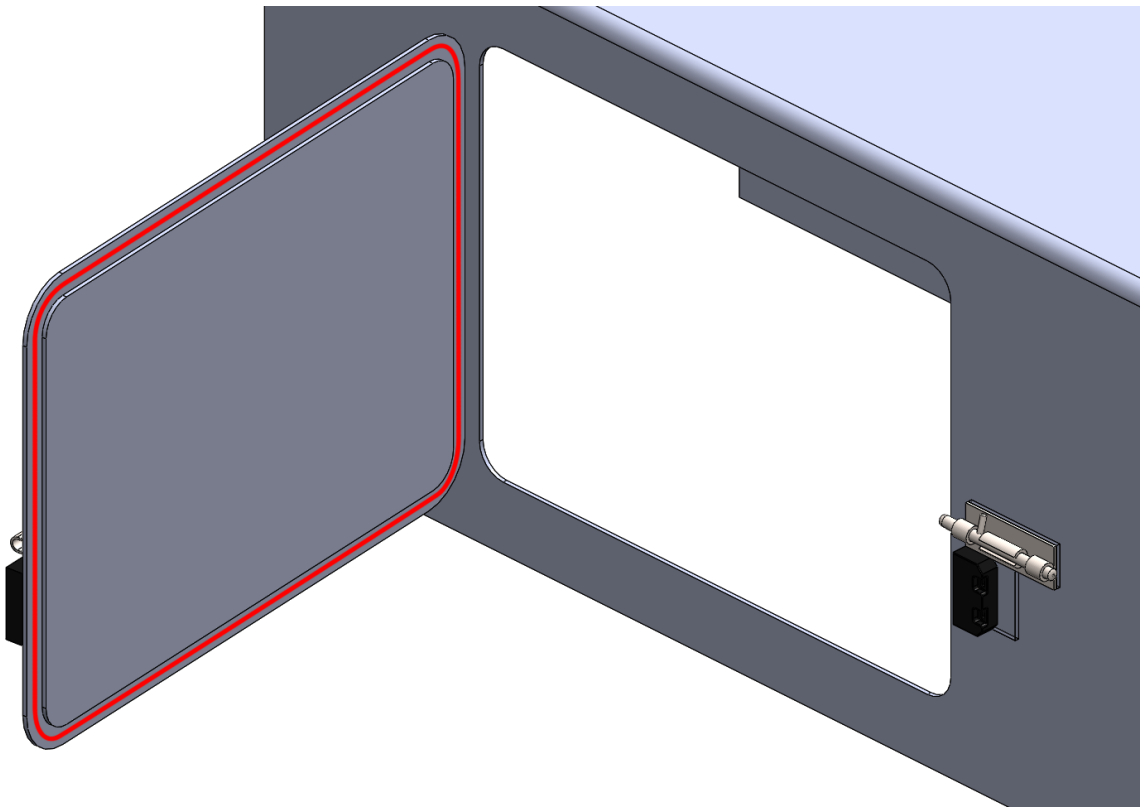


Figur 9.30.: Laserkuttede profiler limes sammen.

På innsiden av denne rammen (profil nr 2) kan man feste en pakning rundt hele åpningen som vil gjøre at når luka klemmes igjen av låsen slippes det ingen luft igjennom. Se Figur 9.31.

Hengsler blir plassert på venstre side av luka. Høyre side av hengslene blir skrudd fast i rammen på luka og venstre side festes i kammerveggen. Under sistnevnte må det også brukes en foring med tykkelse mindre enn rammen slik at pakningen rundt luka har mulighet til å deformeres. Luken vil da kunne åpnes horisontalt mot venstre. Se figur 9.32.

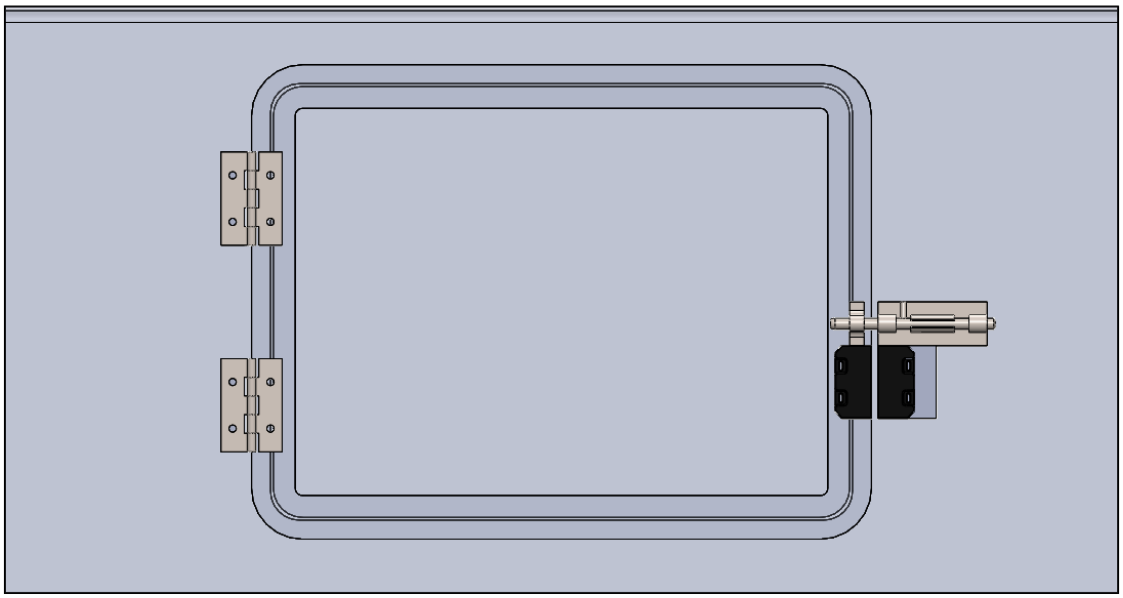
Om luken skulle kunne åpnes oppover hadde man trengt noe for å holde den åpen. I tillegg ville både plassering av hengsler og lås begrenset høyden på åpningen. Om luken skulle ha blitt



Figur 9.31.: Åpen luke med plassering av pakning markert i rødt.

åpnet nedover ville den ha kunne hvilt på kontrollpanelet i front men hatt samme begrensning som nevnt over.

Låsen plasseres naturligvis på motsatt side av hengsler. Ved siden av plasseres en sikkerhets-sensor som beskrevet i kriteriene. Begge disse trenger foringer på lik linje som hengslene av samme grunn.



Figur 9.32.: Testkammer med lukket luke.

9.10. Mobilitet

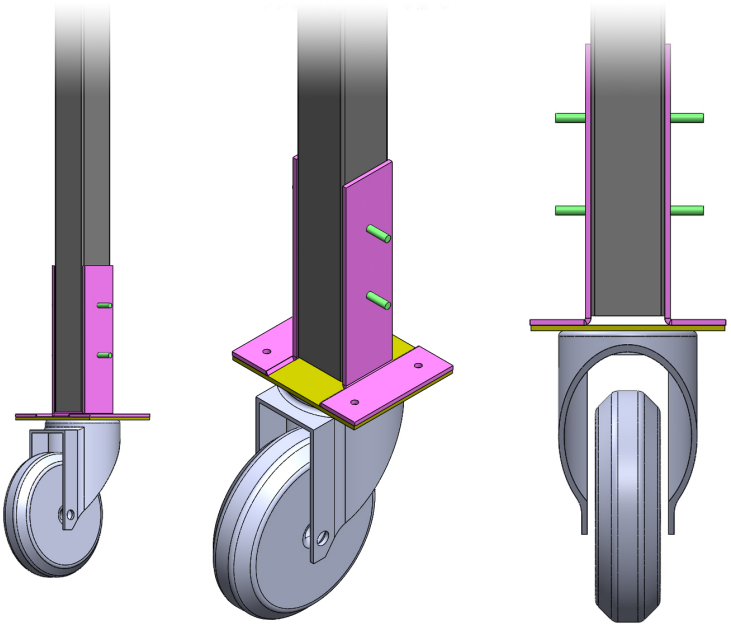
9.10.1. Hjul

I forbindelse med vår overtagelse av tunnelen ble det brukt en jekketralle og treklosser for å flytte den. Tatt i betraktning de relativt store dimensjonene (LxHxB) på tunnelen i tillegg til at vår veiing ga en totalvekt på 127 kg er det tungvint å skulle flytte tunnelen på egenhånd. Det var klart fra begynnelsen av at vindtunnelen trengte en ny måte å transporteres på. Vi startet derfor tidlig med å lage løsningforslag til dette problemet. Alle 3 av disse baserer seg på bruk av hjul. Forskjellen ligger i hvordan de er festet til bena på det eksisterende systemet.

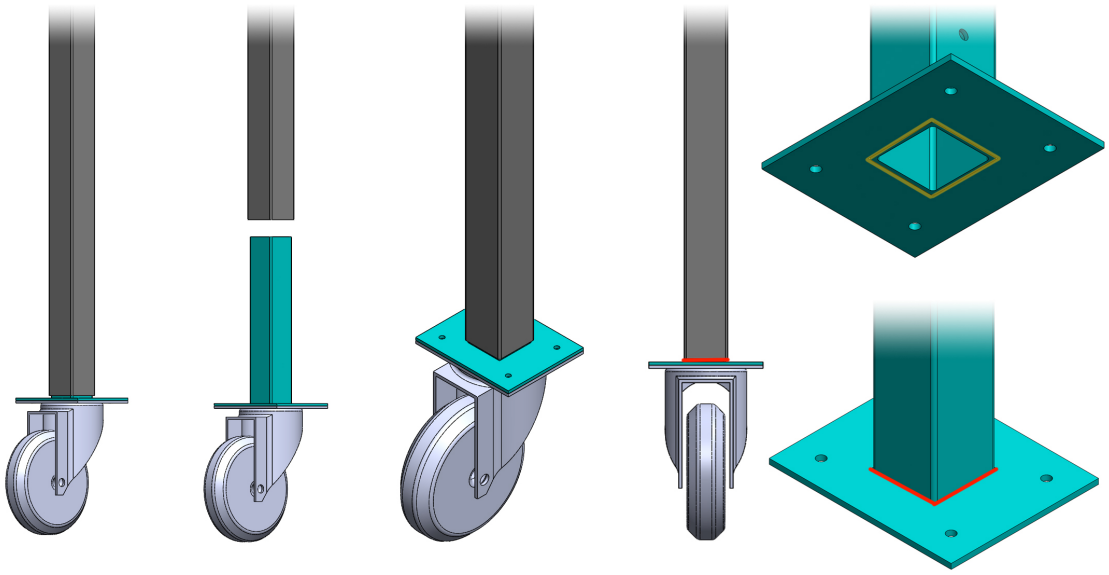
For å komme over dørkarmer etc ønsket vi store hjul. Vi gikk tidlig til innkjøp av 120 mm hjul som vi baserte de 2 første designene på. Først var tanken å lage braketter som koblet hjul og bein sammen. En DAK-tegning av dette kan sees i figur 9.33. Dette ville gi mye stødighet men innebar en god del maskinering i form av skjæring, bøying og boring. De metallplatene vi hadde ville brukt til dette (3 mm stål) måtte også ha blitt bestilt.

Løsning 2 var noe mer elegant i form av et kvadratisk rør som festes til topplaten av hjulet, for så å bli tredd opp på innsiden av beina. Dette ville krevd sveising da festebraketten i utgangspunktet måtte ha bestått av 2 komponenter (kvadratisk rør og plate). På grunn av manglende erfaring var sveising noe vi helst ville unngå. Løsningen ville hatt fordelen ved å kunne plassere vibrasjonsdempende materiale mellom bein og brakett om vibrasjoner skulle vise seg å bli et problem. Størrelsen på røret vi ville brukt her (36 mm) ville ha måtte blitt bestilt.

Løsning 3, og den løsningen vi så langt har gått for, er en modifisert løsning 2. På grunn av bredden på topplaten på hjulene vi baserte oss på var vi av den oppfatning at disse ville øke systemets totale bredde unødvendig. Vi fant derfor nye hjul (100mm) med gjenger som kan



Figur 9.33.: Forslag 1. Festebrakett i rosa, bolter for festing i grønt og topplate tilhørende hjul indikert i gult.



Figur 9.34.: Forslag 2 med sveisestreng i gult og plassering for vibrasjonsdempende materiale i rødt.

både låses normalt så de ikke ruller men også slik at de ikke roterer horisontalt. Disse er avbildet i figur 9.35.

Tanken her er da at man kutter til en firkantskive i stål (36x36 mm) som akkurat får plass på innsiden av av beina. Denne skiven borres hull i så hjultappen kan tres igjennom. Skiven blir også sveiset til et 150 mm langt rør med 35 mm diameter. En M12 mutter blir så skrudd fast igjennom røret slik at sammenstillingen er festet godt.

Som nevnt tidligere har vi lite erfaring med sveising så noe tid har blitt brukt på å lære litt om dette. Røret vi bruker i løsning 3 er av såkalt galvanisert stål som vil si at stålet har et tynt belegg av sink som beskytter mot korrosjon. Problemet med å sveise dette er at sinkets kokepunkt er lavere enn stålets smeltepunkt. Dette gir avgasser som absolutt ikke er bra for kroppen og man må derfor ta forholdsregler (avtrekk, slipe vekk belegg etc).

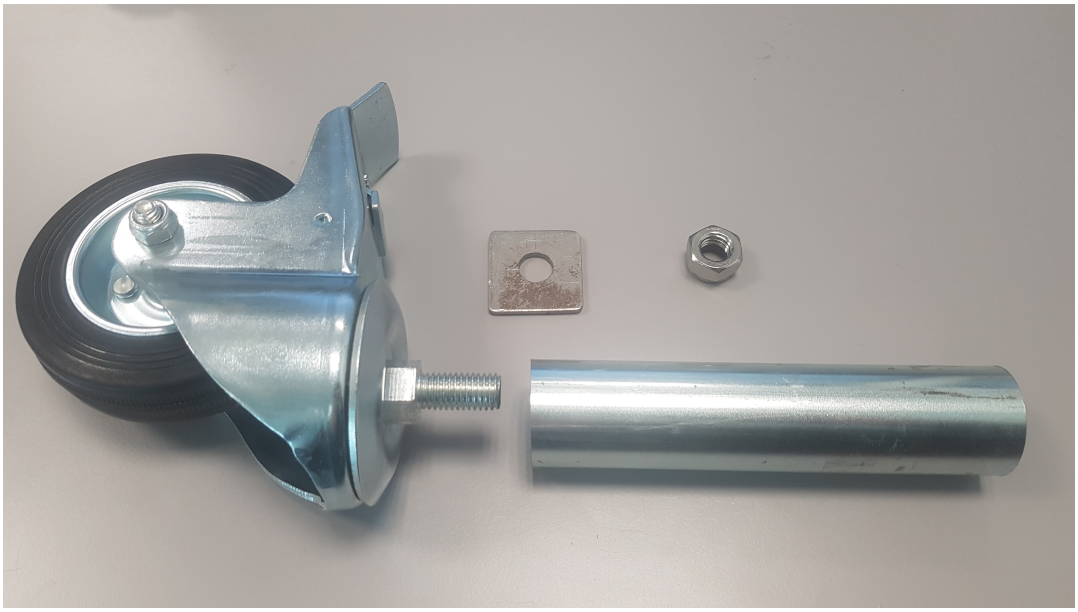
9.10.2. Demontering

Modul 3

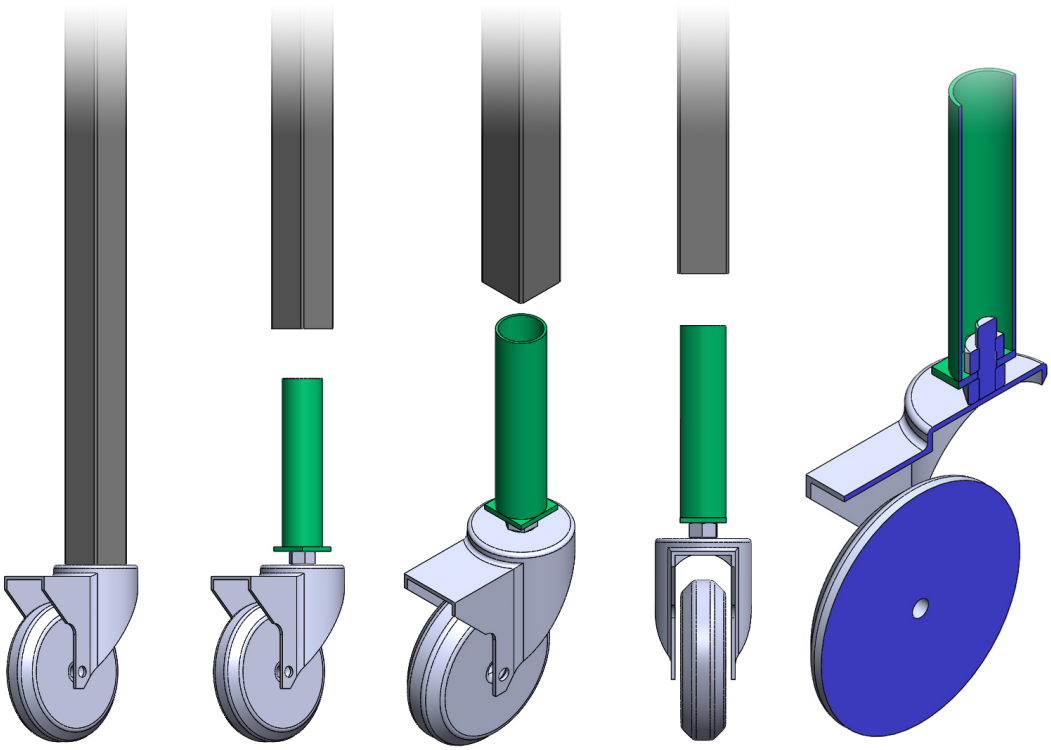
Vi ble informert tidlig om at det var vanskelig å få flyttet den opprinnelige tunnelen dit den står i dag (måleteknisk lab, USN Kongsberg).

I konsept 2 forårsaker diffuserens lengde og dysens bredde at henholdsvis dørkarmen og korridoren utenfor måleteknisk lab setter en stopper for transport i tilfellet tunnelen besto av kun en modul.

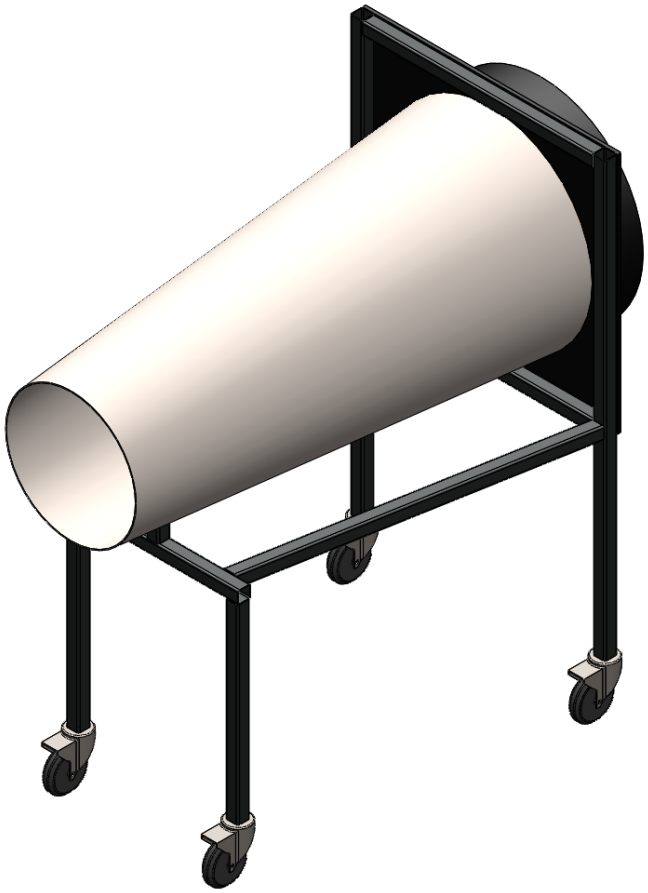
Kravene som omhandler tunnelens evne til å bli flyttet til et annet sted på universitet (**K.VT 08** og **K.VT 09.03**) dikterte derfor at tunnelen i sin helhet måtte deles opp i enkeltstående moduler.



Figur 9.35.: Nytt valg av hjul med gjenger samt øvrige komponenter.



Figur 9.36.: Løsning 3 med sveiset skive og sylinder i grønt og tverrsnitt i blått.



Figur 9.37.: Modul 3: Diffuser med tilhørende ramme.

Modul 3 består av diffuseren, vifte, viftefeste, gitter, ramme og hjul. Panelet som viften er festet til på den opprinnelige tunnelen brukes på nytt og diffuseren festes også i dette panelet. Sikkerhetsgitteret som hindrer tilgang til viften blir også gjenbrukt. Komponentene festes til en ny ramme som også støtter oppunder diffuseren i front. Se figur 9.38. Her vil også rammen være smalere i møte med modul 2 for å kunne føres inn der den opprinnelige viften stod.

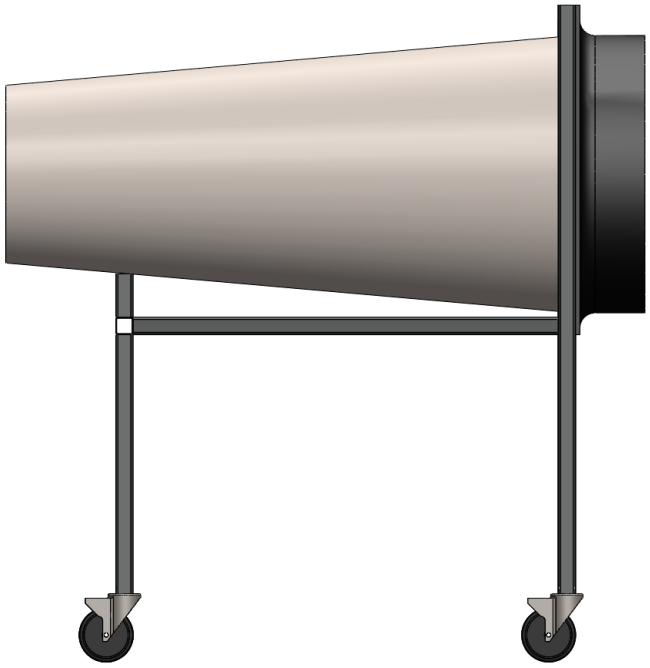
Modulen må enten kobles fra tunnelen ved overgangen til diffuser eller ved testkammer. Måten alle modulene kobles til hverandre forklares nærmere i 12.

Modul 1

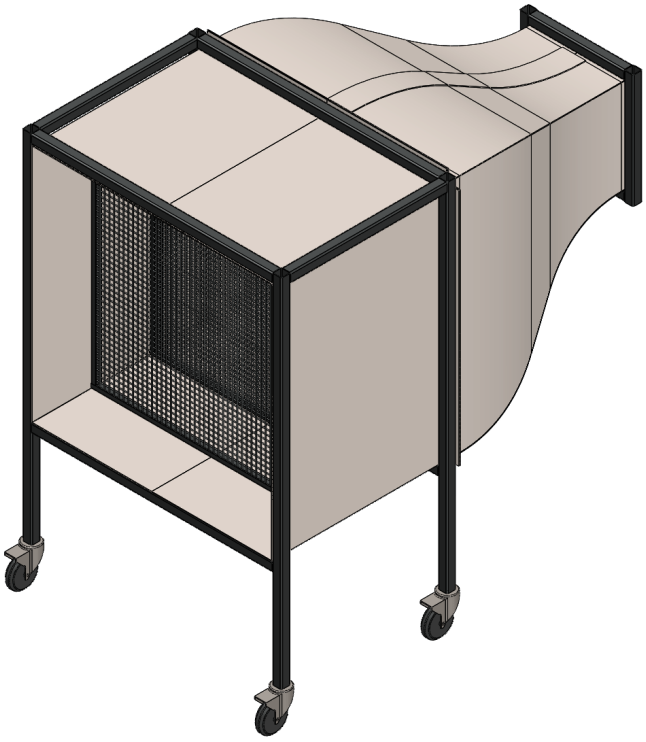
Modul 1 består av manipuleringskammeret med nettinger og strømretter, dyse, ramme og hjul (figur 9.39). Om man ved transport først demonterer dysen fra modulen vil lengden på manipuleringskammeret (750mm) alene kunne imøtekomme kravene. Derfor er sistnevnte festet til rammen og krever ikke demontering ved transport. Dysen og manipuleringskammeret er begge designet med en flens som brukes til å koble disse sammen. Her er det også tenkt at det skal brukes en pakning imellom for å unngå luftlekkasje.

På figur 9.40 ser man modulens tyngdepunkt. Det er plassert noe til venstre for rammens to bakre hjul som vil bety at modulen ikke vil tippe over når den er stasjonær. Strømretteren lar seg ikke modellere i sin helhet så denne er ikke representert i illustrasjonene. Det betyr at i realiteten vil vekten av strømretteren flytte tyngdepunktet ytterligere til venstre og føre til høyere stabilitet enn det som gis inntrykk av her.

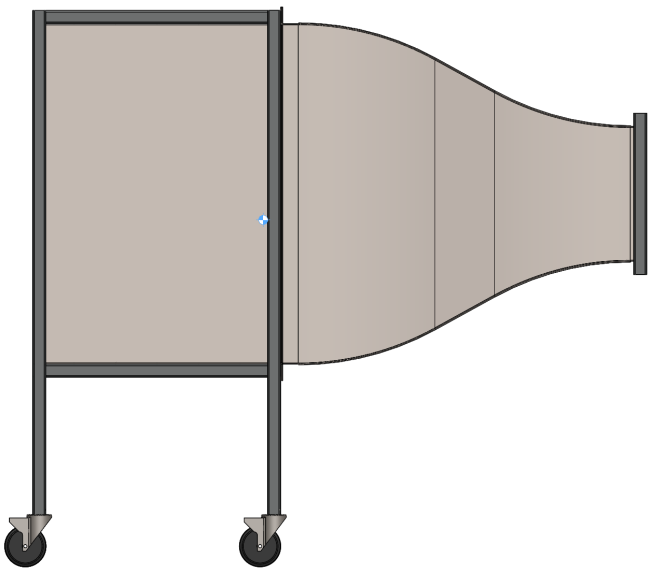
Dysen må som nevnt kobles fra. Denne er like bred som den er høy (1026 mm) og vil ikke kunne flyttes igjennom en dør uansett hvordan den roteres. Derfor må den også være todelt. Rekkefølgen for demontering er illustrert i figur 9.41. For å holde enden samlet og unngå luftlekkasje brukes det en rammme av firkantrør. Dette forklares nærmere i 12.



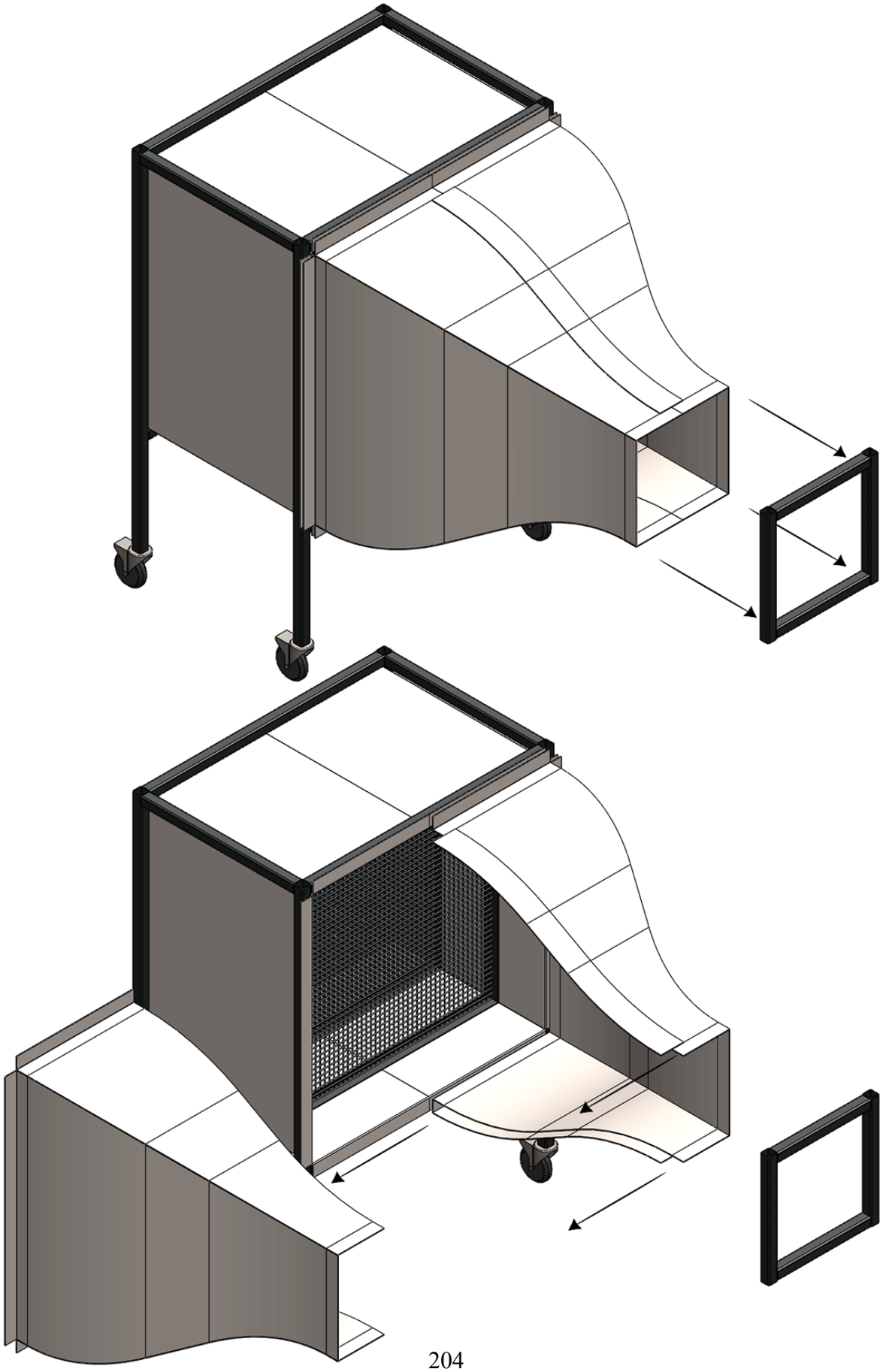
Figur 9.38.: Modul 3: Diffuser med tilhørende ramme sett fra siden.



Figur 9.39.: Modul 1 med tilhørende komponenter.



Figur 9.40.: Modul 1 med tilhørende komponenter. Legg merke til plassering av tyngdepunkt.



Figur 9.41.: Rekkefølge for demontering av dysen før transport.

10. Datateknisk design

10.1. Innebygd Datamaskin (IDM)

Fra krav K.DB 01 skal brukeren av vindtunnelen kunne lagre og ta med seg måledataene fra et eksperiment, for å kunne bearbeide og analysere dem videre. For å oppfylle dette kravet må vindtunnelen utstyres med en datamaskin. Denne kaller vi vindtunnelens **Innebygde Datamaskin**, eller IDM, som en fornorskning av uttrykket "embedded computer". Måledata skal mellomlagres på datamaskinen og kunne bli eksportert i et regnearkformat.

Den innebygde datamaskinen skal også gjennomføre arbeidet med lukket sløyfe regulering av vindhastigheten i vindtunnelen. Dette skal la brukeren sette en vindhastighet som holdes, istedenfor å måtte justere vifteturtallet manuelt.

10.1.1. Teknisk beskrivelse

IDM-en består av tre maskinvarekomponenter og tre softwarekomponenter:

- Hardware
    1. Arduino mikrokontroller
        - Foretar avlesing av sensorer og lukket sløyfe regulering av vindhastigheten i tunnelen.
    2. Raspberry Pi single-board computer (SBC)
        - Kjører brukergrensesnittet og utfører logging og eksport av måledata.

3. Touchskjerm
    - Display og kontrollpanel som lar brukeren observere måledata og styre vindtunnelfunksjonene.

- Software
    1. Kontrollprogram, som kjører på mikrokontrolleren.
    2. Brukergrensesnitt, som kjører på Raspberryen.
    3. Webserver, som også kjører på Raspberryen.

Valget å bruke en Raspberry Pi ble tatt på grunnlag av

- Hardware – Raspberry Pi-kort var tilgjengelig for gruppa, utvikling kunne starte umiddelbart.
- Software – Raspberry Pi kjører Linux, et åpent og fleksibelt operativsystem.
- Erfaring – Gruppa hadde allerede erfaring med bruk av Raspberry Pi.
- Dokumentasjon – Det finnes store mengder tilgjengelig dokumentasjon til Raspberry Pi og Linux.

Ulempene med Raspberry Pi er at den ikke har innebygd analog-digital konverter (ADC), kan ikke motta sensordata fra sensorer som gir analoge signaler; og ikke optimalisert for sanntidssystemer. Det siste poenget er ikke kritisk, siden den raskeste syklusen i kontrollsystemet er 200 ms, men mangelen på ADC var vi ikke villige til å akseptere. Vi kunne ha benyttet en SBC med innebygd ADC, men vi valgte å løse begge ulempene ved å benytte en Arduino mikrokontroller. Den fungerte da som en ekstern ADC/sanntidskoprosessor.

Kommunikasjon med ekstern PC

Det var ikke praktisk å koble en PC til IDM-en via nettverkskabel eller USB. Den valgte løsningen ble å konfigurere Raspberry Pien som et trådløst aksesspunkt og gjøre data tilgjengelig

via en webserver. Prinsippet er illustrert i Figur 10.1.

Kommunikasjon mellom Arduino og Raspberry Pi

Den interne kommunikasjonen i IDM-en utfører to oppgaver: Sensordata og tid fra sanntidsklokka sendes fra Arduinoen til Raspberryen, og settpunkter for vindhastighet og testmodellvinkel sendes fra Raspberryen til Arduinoen, som vist i Figur 10.2. Kommunikasjonen gjøres ved hjelp av seriekommunikasjon over en vanlig USB-kabel, som det finnes biblioteker til på begge platformene.

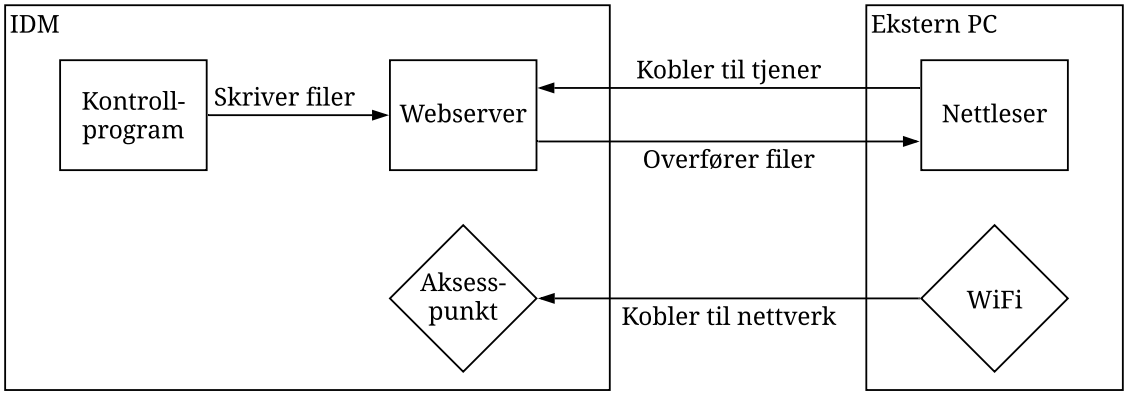
10.1.2. Lagring av måledata

IDMen lagrer måledata i en SQL-database i opptil seks måneder. For å hindre at databasefilene blir for store opprettes det en ny fil hver måned. Når det eksisterer seks filer blir den eldste filen slettet fra disken. Dette gir en god balanse mellom å ta vare på dataene i tilfelle de trengs senere og ytelse på IDM-en.

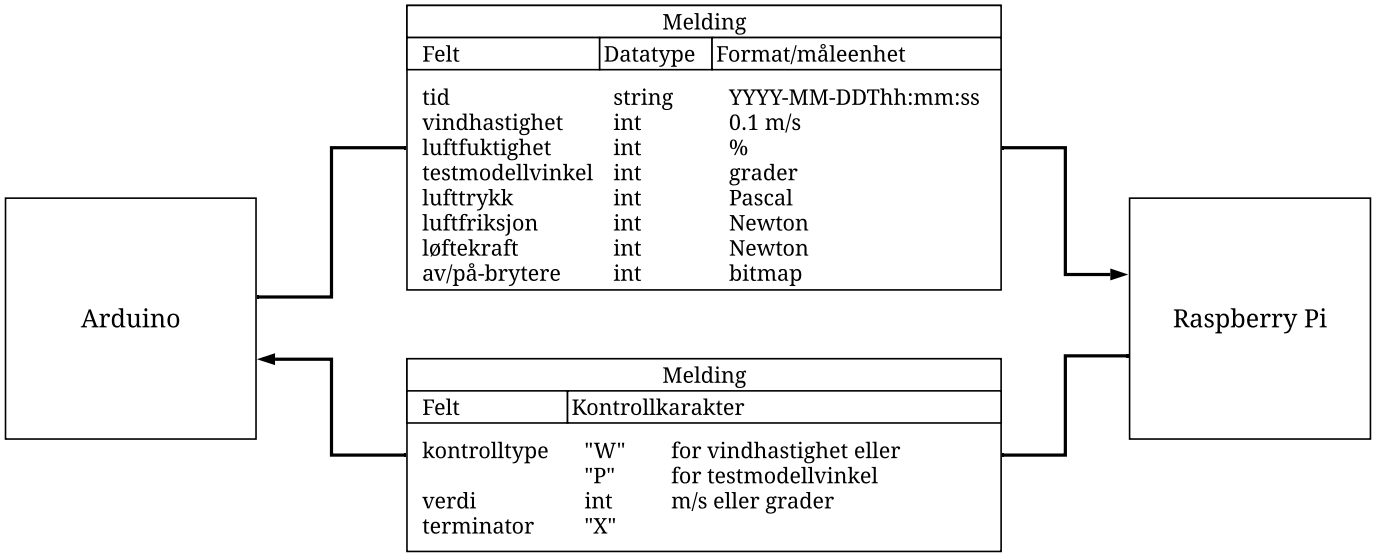
Eksport av måledata

Måledata kan eksporteres fra databasen ved hjelp av eksportsiden i brukergrensesnittet vist i avsnitt 13.3.4 på side 269.

Dataene lagres på webserveren i comma-separated values (.csv)-format, som kan importeres i Excel og OpenOffice.



Figur 10.1.: Webserver og trådløst aksesspunkt.



Figur 10.2.: Meldingsformat for meldinger mellom Arduino og Raspberry Pi

10.1.3. Brukergrensesnitt og skjerm

Vi har sett på løsninger med vindtunnelen for å bruke en touch skjerm med IDM-en. Brukeren kan styre vindtunnelen igjennom skjermen festet til vindtunnelen. Samt kunne se utvalgte data under testing, som vindhastighet og krefter påført testmodellen. I henhold til kravene vi lagde med oppdragsgiver.

Planen for skjermen på vindtunnelen, er at vi kan både styre vindtunnelen samt se dataverdier på en og samme skjerm. Dette gjør det lettere for brukeren å kontrollere systemet og lett for brukeren å se data mens eksperimentet pågår. Da trenger man ikke en ekstern datamaskin.

Raspberry Pi touch-skjerm

Med Raspberry Pi har vi muligheten til å bruke en 7-tommers touch-skjerm som har oppløsning på 800 x 480 piksler. Denne er produsert spesifikt for Raspberry Pi, og kommer ferdig med kontrollbrett og ledninger illustrert i Figur 10.3.

Brukergrensesnitt

Brukergrensesnittets oppgave er å gi brukeren en enkelt og intuitiv måte å observere og styre vindtunnelen. Det må være forståelig og lett å bruke, og med tanke på at det er en touch-skjerm, må det designes med knapper det er lett å trykke på.

Brukergrensesnittet inneholder funksjoner for å:

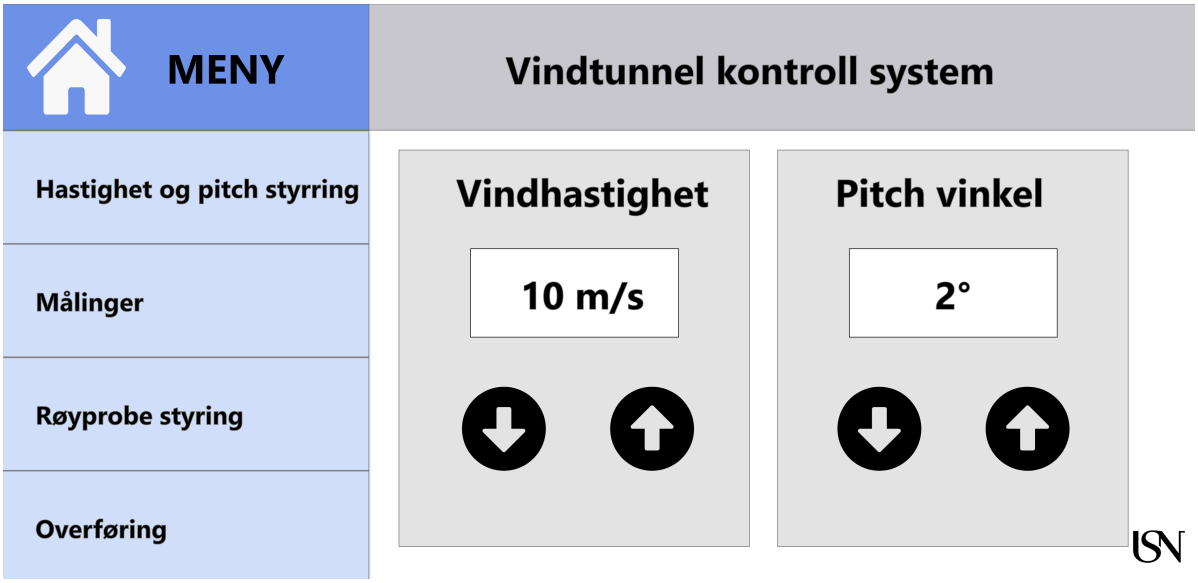
- Styre lufthastigheten i vindtunnelen.
- Styre pitchvinkelen til testmodellen.

Figur 10.3.: Raspberry Pi Touch-skjerm 7”.

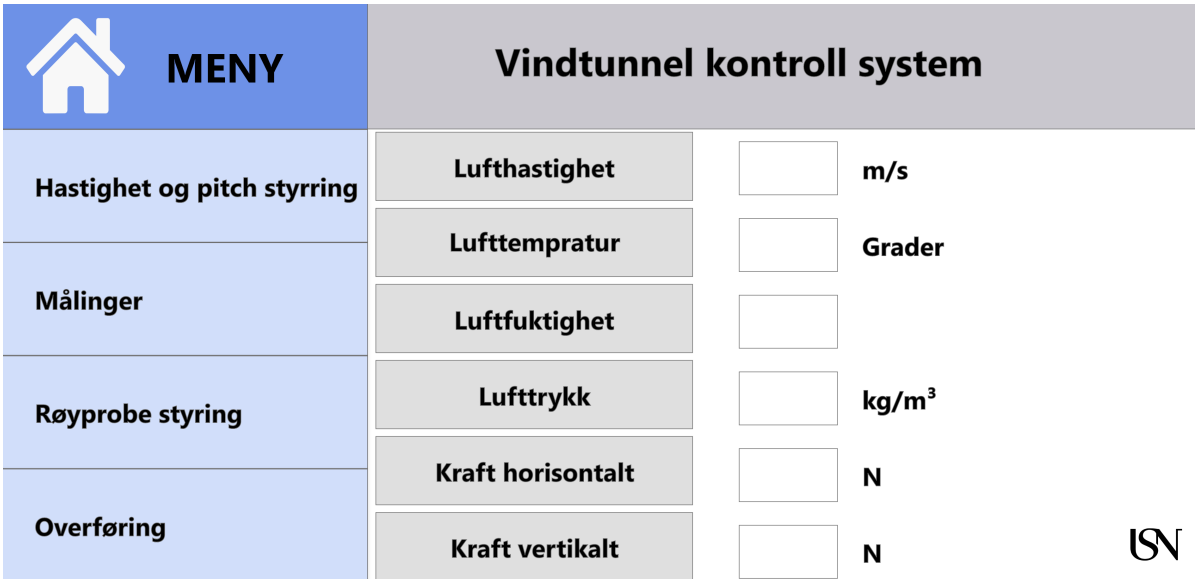
- Vise lufthastighet, lufttrykk, lufttemperatur, og luftfuktighet i vindtunnelen.
- Vise vertikal og horisontal kraft på testmodellen.
- Styre posisjonen til røykproben.
- Eksportere data slik at den kan brukes i videre analyse.

De opprinnelige modellene for grensesnittet kan sees i Figurene 10.4 til 10.9, mens det ferdige brukergrensesnittet diskuteres i avsnitt 13.3 på side 264.

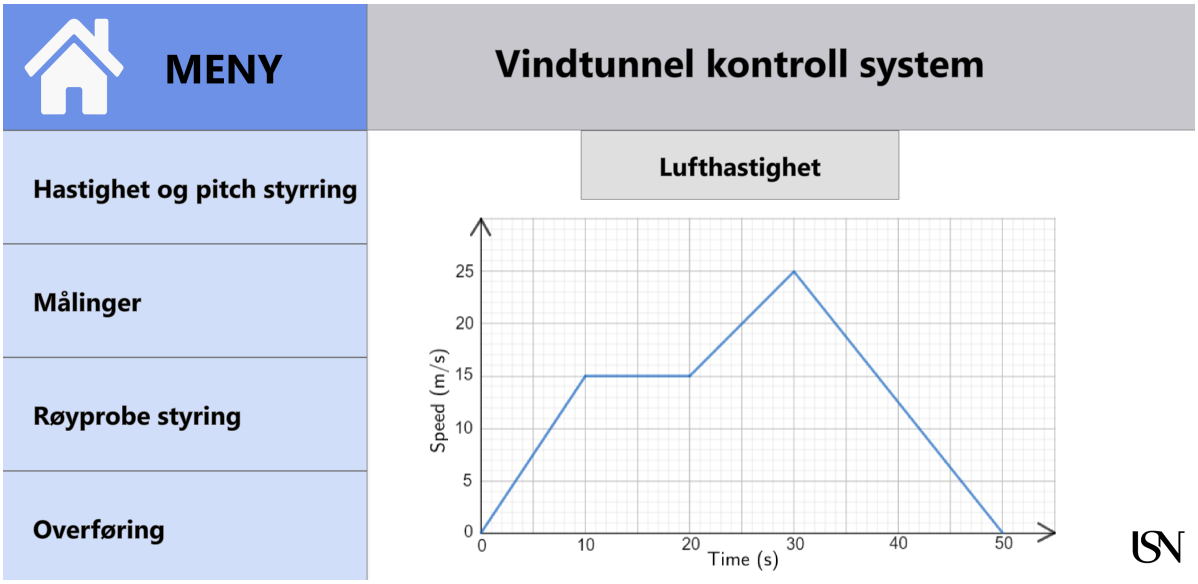
Figur 10.4.: Brukergrensesnittets startside.



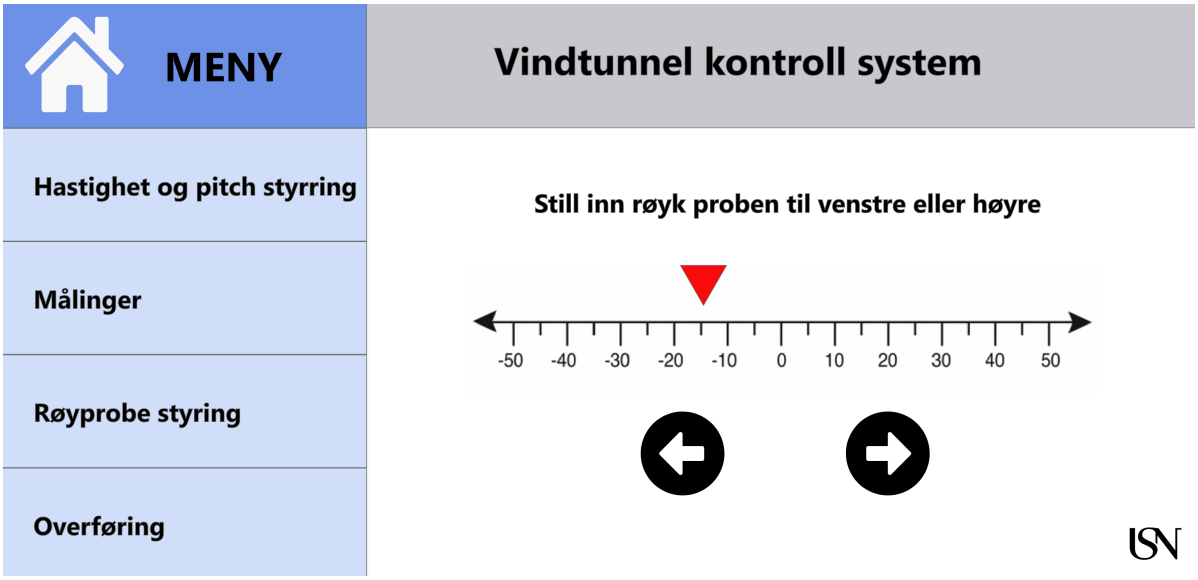
Figur 10.5.: Side for styring av lufthastighet og pitchvinkel.



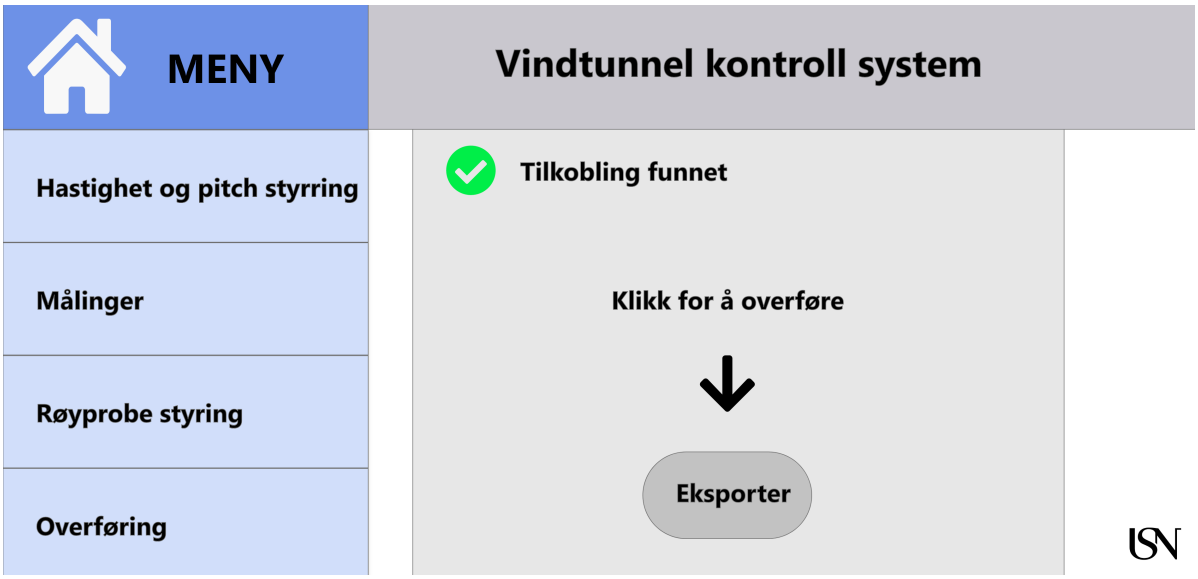
Figur 10.6.: Side for fremvisning av måledata.



Figur 10.7.: Side for måledatagraf.



Figur 10.8.: Side for stilling av røykprobe.



Figur 10.9.: Side for eksportering av data.

Del III.

Gjennomføring og resultat

11. Elektroteknisk implementasjon

11.1. Implementasjon av styringssystemet

Denne seksjonen fokuserer kun på lavnivådelen av styringssystemet. Med lavnivå så mener vi implementasjon av systemdesign som er nede på komponentnivå, dvs hvordan enkelstående hardware komponenter kobles sammen itillegg til programvaren som styrer disse komponentene.

Siden prosjektet vårt ble omgjort til et teoretisk prosjekt grunnet Corona-viruset, så får vi ikke fullført den fysiske implementasjonen og testingen av styringssystemet. Så denne seksjonen vil ta for seg klargjøringen av den fysiske implementasjonen, det inkluderer en veiledning for kobling av hardware og programmering av vindhastighetsensoren og lukket sløyfe PID-kontrolleren.

Denne seksjonen vil ta for seg alt som neste prosjektgruppe trenger å vite for å være i stand til å fysisk implementere lavnivådelen av styringssystemet, inkludert vindhastighetssensoren som er knyttet til K.M 01 og K.VT 04.

Når lavnivå delen er testet og verifisert at det fungerer som det skal, så skal det interfaces med høynivå delen som involverer kommunikasjon med en Raspberry PI og brukergrensesnittet i form av et touch-display.

11.1.1. Lavnivå hardware implementasjon av styringssystemet

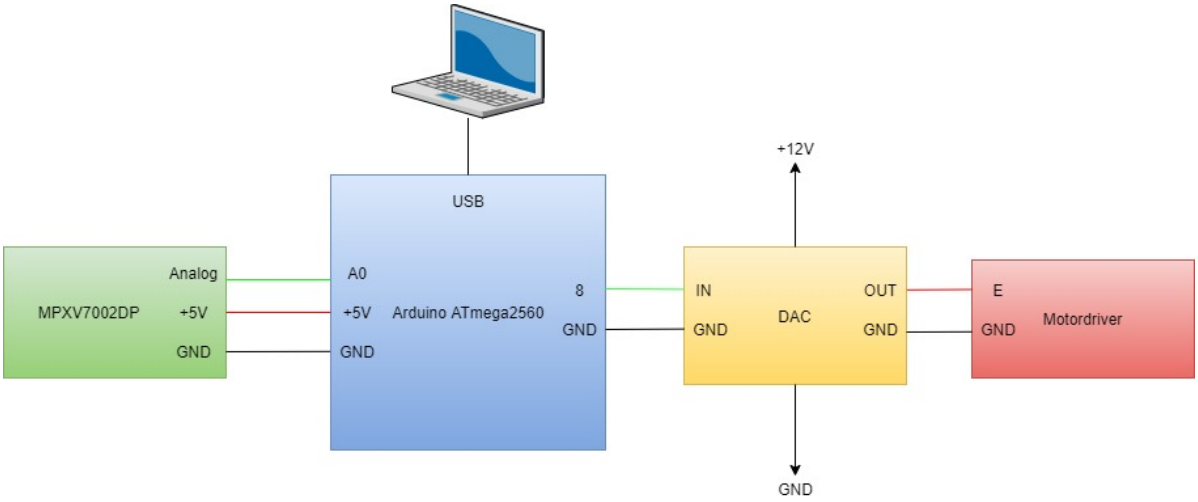
All nødvendig lavnivå hardware som kreves for å implementere styringssystemet med vindhastighetsensoren er følgende:

1. Pitotrør (1 stk.)
2. MLXV7002DP differensial trykksensor (1 stk).
3. Arduino ATmega2560 mikrokontroller (1 stk).
4. Digital til Analog konverter (1 stk).
5. 12 V strømforsyning.
6. Pakke med jumper ledninger.

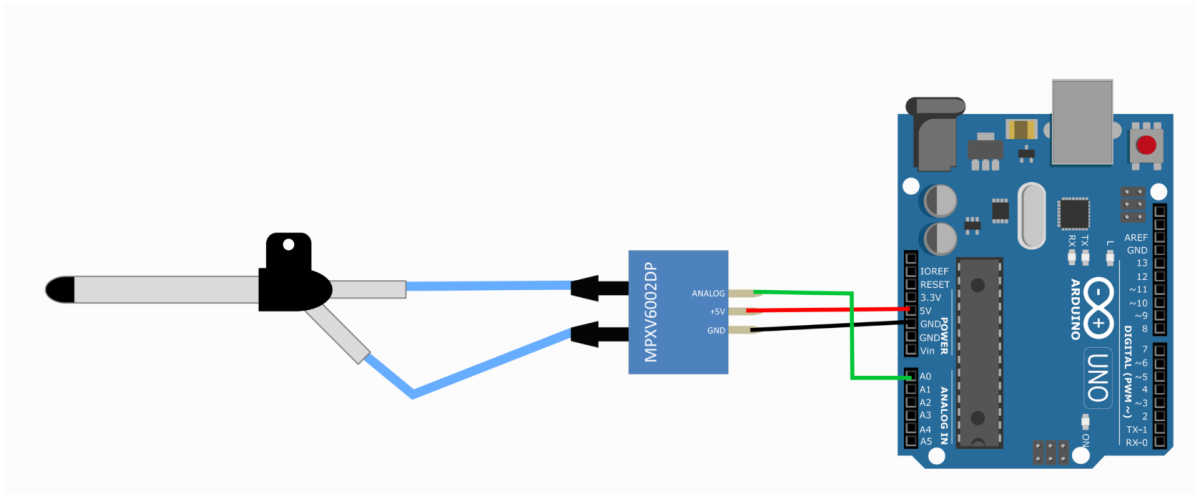
Dette er det nødvendige utstyret som neste prosjektgruppe må få tak i selv enten via skolen, bestille selv eller lage selv. Vi har allerede ett design tilgjengelig for Digital til Analog konverter kortet hvis neste prosjektgruppe ønsker å produsere kortet selv. Pitotrøret og differensial trykksensoren kan man få bestilt i en og samme pakke.

Koblingsskjema for lavnivå hardware

Figur 11.1 viser hvordan lavnivå delen av styringssystemet kobles opp. DAC kortet krever en ekstern 12 V strømforsyning for å kunne operere. PC kobles til mikrokontrolleren med USB port for å kunne laste opp programvaren til styringssystemet. Alle komponentene er jordet. Figur 11.2 viser detaljert koblingsskjema for vindhastighetssensoren med pitotrør.



Figur 11.1.: Koblingsskjema for styringssystemet.



Figur 11.2.: Detaljert koblingsskjema for differensial trykksensor med pitotrør.

11.1.2. Lavnivå software implementasjon av styringssystemet

PID-kontrolleren implementerer vi i form av software, dataprogrammet skrives i C/Arduino og lastes opp til ATmega2560 kortet som tar seg av alle sensormålingene og styringssystemet til vindtunnelen. For at dataprogrammet skal fungere som en PID-kontroller så trenger den følgende struktur:

1. Funksjon for å måle vindhastigheten i m/s.
2. Funksjon for PID-kontrolleren for beregning av pådraget.
3. Funksjon som sender pådraget i form av PWM signal ut til motordriveren.

Funksjonene for å måle vindhastighet og sende pådraget som PWM signal skrives i hovedprogrammet. Funksjonen for måling av vindhastighet er basert på koden i [11], funksjonen for selve PID-kontrolleren henter vi fra ett eget PID software bibliotek [3] som vi linker til hovedprogrammet. Alle tre funksjonene kjører i hovedprogrammets main loop. For PID-kontroller funksjonen må vi definere syv variabler:

1. 'Input' defineres som den målte vindhastigheten fra sensoren.
2. 'Output' blir kalkulert av PID-kontrolleren og sendt som ett PWM signal til motordriveren.
3. 'Setpoint' defineres som referansehastigheten inntastet av brukeren.
4. 'SampleTime' defineres som samplingstiden mellom hver gang PID-kontrolleren tar imot måledata fra sensoren. Den står som standard på 100 ms i kildefilen til PID biblioteket.
5. Kp defineres som den proporsjonale forsterkningsfaktoren.

6. Ki defineres som den integrerte forsterkningsfaktoren.

7. Kd defineres som den deriverte forsterkningsfaktoren.

Kp, Ki og Kd er de PID parameterne vi fant fra simuleringen i Simulink. Denne simuleringen var en generell simulering for å demonstrere funksjonaliteten til styringssystemet, og ikke en nøyaktig simulering for den spesifikke motoren i vindtunnelen. Så neste gruppe som tar over denne oppgaven må bare prøve seg fram med forskjellige verdier av PID parameterne, med mindre de klarer å lage en mer nøyaktig simulering.

Dataprogrammet for lukket-sløyfe-styringssystemet med PID-kontroller deles opp i tre filer: Hovedprogrammet som vi har for det meste skrevet selv og lagret i en egen fil med navn 'PID-controller' og PID software biblioteket som er delt opp i en header-fil og en kildefil.

Kodeutdrag 1 viser utdrag fra hovedprogrammet til styringssystemet med en funksjon for måling av vindhastighet og en funksjon for å sende PWM-signal til motordriver. Koden for måling av vindhastighet er hentet fra [11]. Kildekode for PID biblioteket er lagt som vedlegg i denne rapporten der Vedlegg B.7 er header-fila og Vedlegg B.8 er kildefila. Det trengs ikke gjøre noen endringer i PID bibliotekfilene med mindre man ønsker å endre samplingraten for hvor ofte PID-regulatoren skal ta imot måledata for vindhastigheten. Kodeutdrag 2 viser et utdrag fra kildefilen til PID biblioteket hvor samplingraten kan endres.

```
// Funksjon for å måle vindhastighet
void measureSpeed(){
  float adc_avg = 0; float veloc = 0.0;
  for (int ii=0;ii<veloc_mean_size;ii++){
    adc_avg+= analogRead(A0)-offset;
  }
  adc_avg/=veloc_mean_size;

  // make sure if the ADC reads below 512, then we equate it to a negative
  ↪  velocity
  if (adc_avg>512-zero_span and adc_avg<512+zero_span){
  } else{
    if (adc_avg<512){
      veloc = -sqrt((-10000.0*((adc_avg/1023.0)-0.5))/rho);
    } else{
      veloc = sqrt((10000.0*((adc_avg/1023.0)-0.5))/rho);
    }
  }
  Serial.println(veloc); // print velocity
  delay(10); // delay for stability
  }

void PWMout(){ // PWM funksjon
  analogWrite(DAC, Output); // Sender PWM signal til AC motordriver
  }
```

Kodeutdrag 1: Kodeutdrag fra hovedprogrammet til styringssystemet.

```
PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int POn, int ControllerDirection)
{
    myOutput = Output;
    myInput = Input;
    mySetpoint = Setpoint;
    inAuto = false;

    PID::SetOutputLimits(0, 255);  //default output limit corresponds to the
    ↪  arduino pwm limits

    SampleTime = 100;^^I^^I^^I^^I^^I^^I^^I//default Controller Sample Time is
    ↪  0.1 seconds

    PID::SetControllerDirection(ControllerDirection);
    PID::SetTunings(Kp, Ki, Kd, POn);

    lastTime = millis()-SampleTime;
}
```

Kodeutdrag 2: Kodeutdrag fra kildefilen til PID biblioteket.

11.2. Implementasjon av lastceller

Denne seksjonen vil ta for seg hvordan lastcellene skal kobles og konfigureres slik at de er istand til å måle lift og drag på testmodellen i vindtunnelen basert på [35]. Hensikten med denne seksjonen er å være en veiledning for hvordan neste prosjektgruppe enkelt kan fysisk implementere dette i vindtunnelen for å validere og fullføre krav K.M 08.

11.2.1. Lavnivå hardware implementasjon av lastcellene

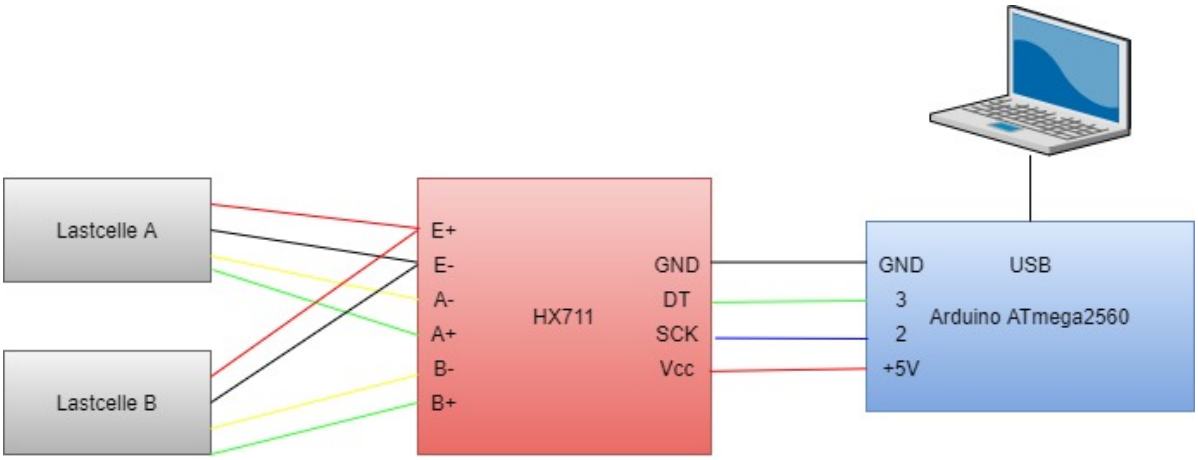
All nødvendig lavnivå hardware som kreves for implementasjonen av lastcellene er som følger:

1. Lastceller (2 stk).
2. HX711 lastcelleforsterker kort (1 stk).
3. Arduino ATmega2560 mikrokontroller (1 stk).
4. Pakke med jumper ledninger.

Dette er utstyr som prosjektgruppen må skaffe selv enten via skolen, kjøpe selv eller lage selv. Lastceller kan man bestille i mange varianter, men det anbefales å bruke den typen vi har anbefalt i denne rapporten. HX711 kortet kan bestilles som ferdig løsning eller dere kan finne en produsent og sende dem gerber-filene til HX711 kortet som vi designet så lager dem kortet for dere. Dette er alle komponentene man trenger for å konfigurere og teste at lastcellene virker som dem skal før man integrerer dem med høynivå hardwaren som inkluderer Raspberry PI og brukergrensesnittet i form av touch-display.

Koblingsskjema for lavnivå hardware

Figur 11.3 viser hvordan lastcellene kobles sammen med lastcelleforsterkeren og Arduino kortet. E portene gir spenning til lastcellene. Når lastcellene blir påført krefter så vil dem produsere en analog utgangsspenning som er proporsjonalt med den påførte kraften, denne spenningen går til A og/eller B portene avhengig av om det er lift og/eller drag krefter som blir påført lastcellene. Denne utgangsspenningen blir så forsterket og konvertert om til ett digitalt signal som sendes til mikrokontrolleren. Mikrokontrolleren sender ett klokkesignal til lastcelleforsterkeren som definerer ett tidsintervall for hvor ofte ett nytt digitalt signal skal sendes til mikrokontrolleren. Man kan se på det som hvor ofte neste "bit" i køen skal sendes til mikrokontrolleren. Man kobler en PC til mikrokontrolleren ved hjelp av USB for å laste opp programvaren for avlesning av lastcelle målingene.



Figur 11.3.: Koblingsskjema for lastcellene.

11.2.2. Lavnivå software implementasjon av lastcellene

For software delen av dette delsystemet bruker vi ett eksisterende software bibliotek som består av to funksjoner, en for kalibrering av lastcellene og en for å konvertere måledataene om til vekt i kg. Det følger med to hovedprogram, ett for kalibreringen og ett for kraftmåling. Tilsammen

utgjør dette 4 kildekode filer, header-fil og kildefil for biblioteket, kildefil for kalibreringen og kildefil for kraftmåling. Lastcellene vil på denne måten fungere som en digital vekt, men siden det er Newton krefter vi er interessert i måle så må vi legge på en ekstra kodesnutt som konverterer fra kg til Newton krefter for lift og drag. Kalibrerings koden brukes for å verifisere at lastcellene gir riktige målinger. Man kjører koden, legger på ett objekt med en kjent verdi, for eksempel ett lodd på 2 kg, så ser man hva man får ut på skjermen. Gir den feil måling så øker man kalibreringsfaktoren i koden til man får riktig måling [5]. Kalibreringskoden kjører man bare en gang til man ser man får riktige målinger, deretter er det hovedprogrammet for konverteringen av måledataene for lift og drag som man laster inn på mikrokontrolleren som vil ta seg av kraftmålingene. Kodeutdrag 3 viser utdrag fra hovedprogrammet for kalibreringen av lastcellene.

Kodeutdrag 4 viser utdrag fra det generiske hovedprogrammet som konverterer måledataene fra lastcellene om til vekt målt i kilo. Det er fint å kjøre dette programmet først for å verifisere at lastcellene klarer å vise riktig vekt målt i kilo, før man øker kompliksiteten med konvertering om til krefter målt i Newton.

Siden vi ikke får testet denne koden og det nærmer seg prosjektets deadline i skrivende stund, så har vi ikke skrevet noen kodesnutt for konverteringen av kilo til Newton for måling av lift og drag. Men vi fant en rapport fra et lignende vindtunnel-bachelorprosjekt som vi legger ved i referansene [33]. Koden i denne referansen er ment for vindtunnel med flere lastcelleforsterker kort, så denne gir mulighet for måling av krefter i mer enn to-akser hvis man ved en senere anledning ønsker å gjøre vindtunnelen enda mer kompleks i framtiden. Denne koden er ihvertfall fin å ta som utgangspunkt også kan neste prosjektgruppe tilpasse den for dette prosjektet.

Vi inkluderer hovedprogrammet fra [33] i Vedlegg B.13, men det krever noen justeringer i lastcelle software-biblioteket som står forklart i [33].

```
void loop() {

  scale.set_scale(calibration_factor); //Adjust to this calibration factor
  Serial.print("Reading: ");
  Serial.print(scale.get_units(), 1);
  Serial.print(" lbs"); //Change this to kg and re-adjust the calibration
  ↪  factor if you follow SI units like a sane person
  Serial.print(" calibration_factor: ");
  Serial.print(calibration_factor);
  Serial.println();

  if(Serial.available())
  {
    char temp = Serial.read();
    if(temp == '+' || temp == 'a')
      calibration_factor += 10;
    else if(temp == '-' || temp == 'z')
      calibration_factor -= 10;
  }
```

Kodeutdrag 3: Kodeutdrag fra hovedprogrammet for kalibrering av lastcellene.

```
HX711 scale;

void setup() {
  Serial.begin(9600);
  Serial.println("HX711 scale demo");

  scale.begin(DOUT, CLK);
  scale.set_scale(calibration_factor); //This value is obtained by using the
  ↪  SparkFun_HX711_Calibration sketch
  scale.tare(); //Assuming there is no weight on the scale at start up, reset
  ↪  the scale to 0

  Serial.println("Readings:");
}

void loop() {
  Serial.print("Reading: ");
  Serial.print(scale.get_units(), 1); //scale.get_units() returns a float
  Serial.print(" kg"); //You can change this to lbs but you'll need to
  ↪  refactor the calibration_factor
  Serial.println();
}
```

Kodeutdrag 4: Kodeutdrag fra hovedprogrammet for vektmåling når lastcellene blir påført krefter.

```
void loop() {
// Serial.println("RAW DATA BELOW");
// sendRawData(); // this is for sending raw data, for where everything else
is done in processing
// Serial.println();
 Serial.println("WEIGHT DATA BELOW");
 sendWeightData();
 Serial.println();
 //calculate forces in N

F1 = weights[0]*go/1000; //[N]
F2 = weights[1]*go/1000; //[N]
lift = F1 + F2;
drag = (weights[2])*go/1000; //[N]
Mle = x01 * F1 + x02 *F2; // [Nm]
xcp = (x02 * F2 + x01 * F2) / (F1 + F2)*1000;
 //aerodynamic coefficients

Cl = (lift*2)/(rho*v*v*S);
Cd = (drag*2)/(rho*v*v*S);
Cm = (Mle*2)/(rho*v*v*S*C);
Serial.print("lift = ");
Serial.print(lift);
Serial.println(" [N]");
}
```

Kodeutdrag 5: Kodeutdrag fra hovedprogrammet for konvertering av kilo til Newton [33].

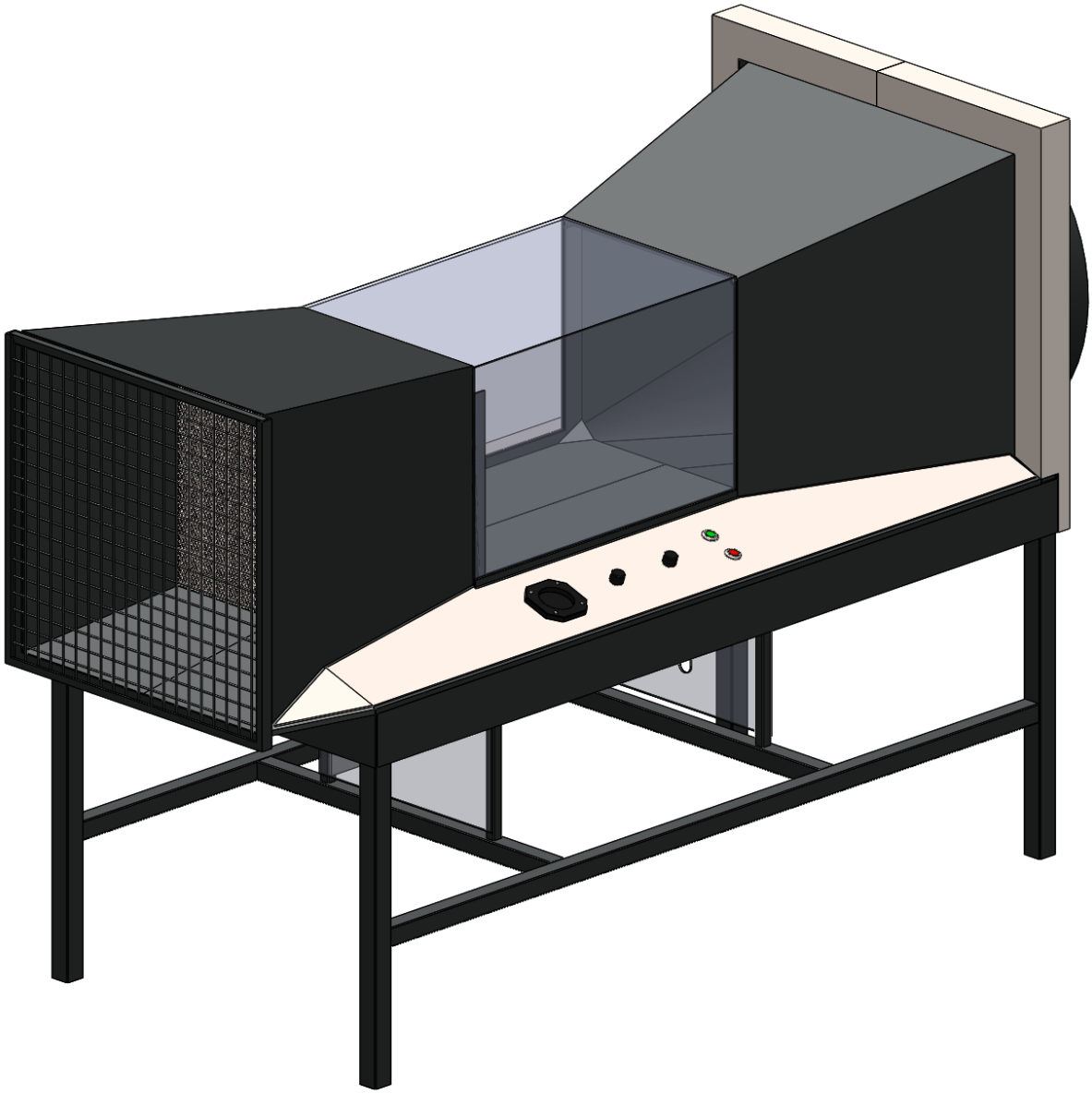
12. 3D-Modellering og simulering

12.1. DAK (Dataassistert konstruksjon)

For å både forbedre forståelsen og danne et sammenligningsgrunnlag for simulasjoner har den opprinnelige vindtunnelen blitt modellert i SolidWorks (figur 12.1).

Ettersom oppgaven ble mer teoretisk vinklet ble det klart at gruppen ikke kom til å produsere et (ferdig) fysisk produkt. Da oppdragsgiver fortsatt ønsker en vindtunnel er det sannsynlig at noen andre tar fatt i problemstillingen på et senere tidspunkt og bygger videre på vårt arbeid. For at de skal få så mye nytte som mulig av vår oppgave har det vært viktig for oss at de har noe helhetlig visuelt å forholde seg til, og det er her vår DAK-sammenstilling av konsept 2 kommer inn (figur 12.2).

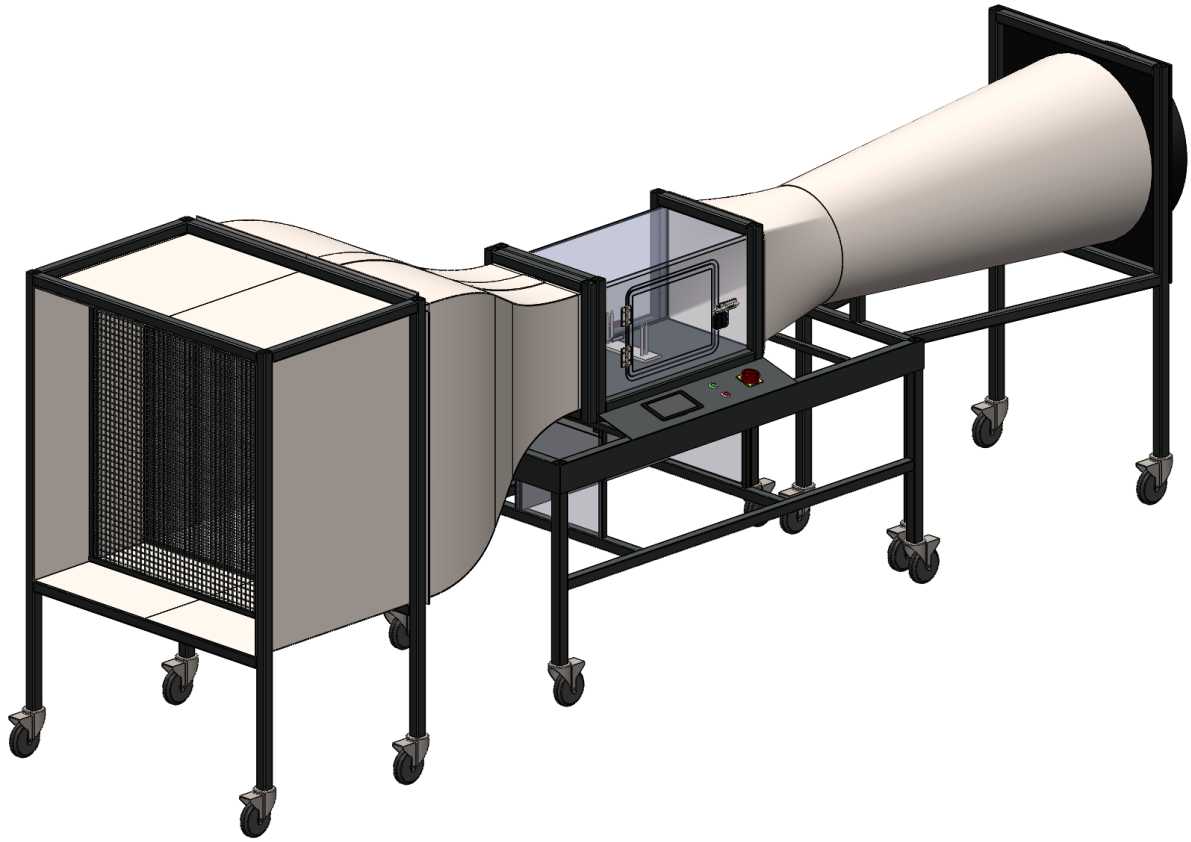
For å kunne kjøre både styrkeberegninger og flow-analyse må man ha en 3D modell, men det har blitt gitt ekstra oppmerksomhet til å modellere hovedkomponentene som skal brukes på en realistisk og lett modifiserbar måte. På den måten blir det enklere for oppgavens arvtagere å tilpasse sammenstillingen til de materialene de kommer til å ha tilgang på. For eksempel er det antatt at metallplater og firkantrør kommer fra universitets materiallager og man vil derfor måtte tilpasse dimensjoner på deler utifra dette.



Figur 12.1.: Sammenstilling av opprinnelig vindtunnel.

12.1.1. Modellering og K-faktor

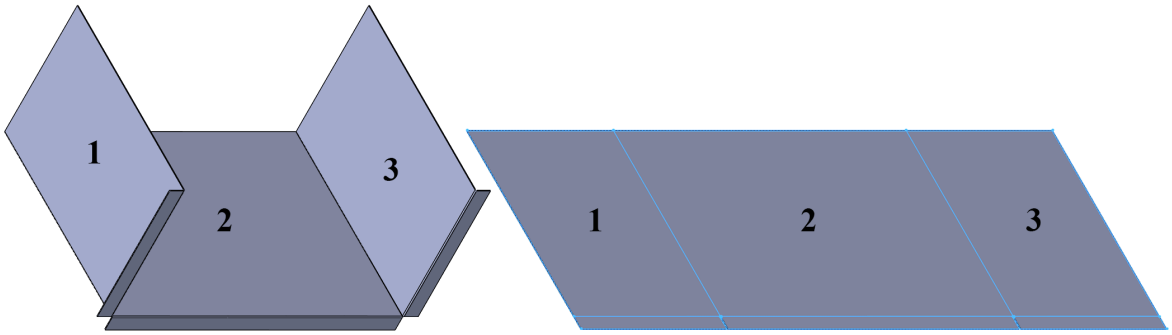
I sammenstillingen er det flere av de store komponentene som skal skal fremstilles av plater. Manipuleringskammeret, dysen, testkammeret, diffuseren og overgangen mellom testkammer og diffuser ville alle få sin form ved hjelp av bøying. I flere av 3D-modellene for disse delene er det derfor tatt i bruk en spesifikk funksjon for deler som skal bøyes, kalt “Sheet Metal”



Figur 12.2.: Sammenstilling av konsept 2.

(metallplater). Denne funksjonen vil brette ut modellen slik at man får et flatt mønster som representerer delen før tilbøying. Se figur 12.3. Fellesnevneren her er at modellen trenger en spesifisert ”K-faktor” som sier noe om hvor mye ekstra materiale som må tas i betrakning for at bøyen skal overholde de ønskede dimensjonene.

Denne K-faktoren avhenger blant annet av total lengde, innsideradius, bøyemetode, bøyevinkel og materialtykkelse. Det er derfor av hensyn til disse variablene (spesielt bøyemetode og materialtykkelse) det er unnlatt å spesifisere k-faktor i selve 3D-filene. Spesielt dysen har en avansert profil hvor dette blir viktig for å få ønsket sluttresultat.

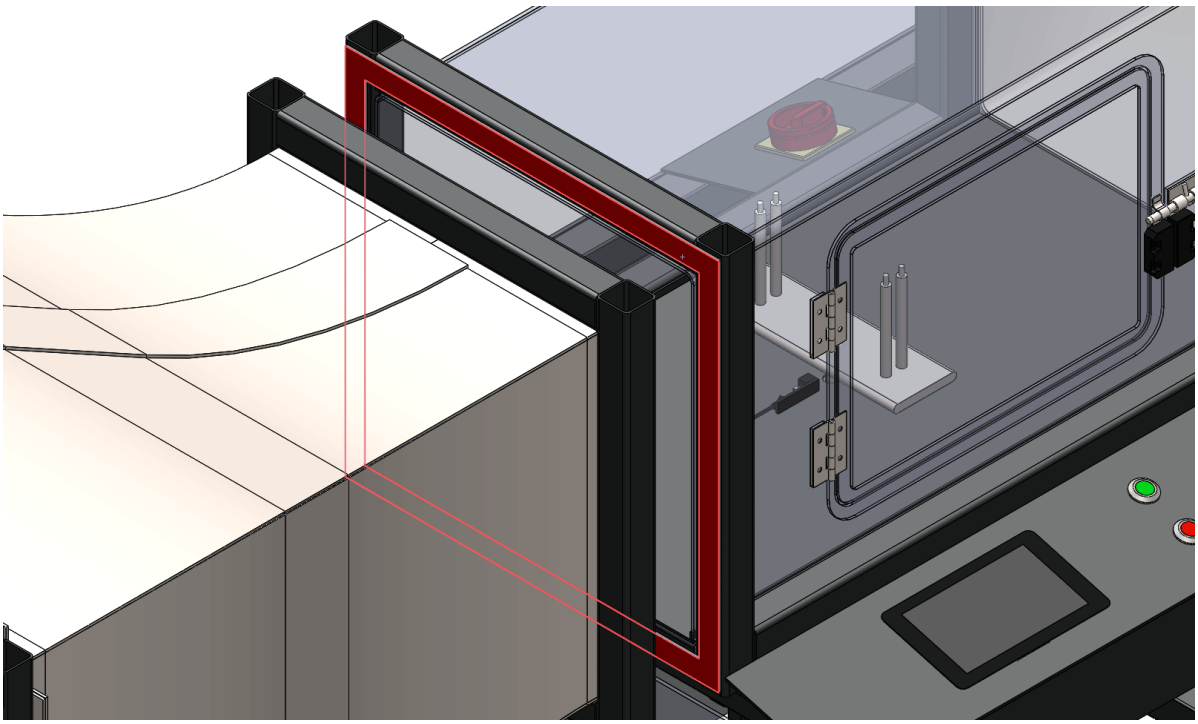


Figur 12.3.: 3D-modell for halve manipuleringskammeret. Sammenligning av original og flatpakket modell.

12.1.2. Rammer

Selve testkammerets utforming i konsept 2 ligner veldig på sånn det er i det eksisterende systemet. Rammen har derimot blitt modifisert i form av at testkammeret har blitt flyttet lengre frem mot modul 1 sånn at ikke rammen er til hinder for dysen. Også rammen rundt testkammeret er endret i form av at begge åpninger nå omsluttes av firkantrør. Årsaken til dette er todelt. Siden tunnelen nå deles opp i moduler, må de ha robuste grensesnitt og disse overgangene skal heller ikke slippe luft fra utsiden og inn.

Ved å gi dysen og testkammer disse like rammene kan man ved hjelp av gummipakninger imellom løse begge problemer. Lignende pakninger plasseres også mellom koblingspunktene og testkammerets vegger. Innføringen av testkammerets to rammer kompliserer et eventuelt bytte av testkammerets tak og vegger. Dette grunnet designet på luken som dikterer at alle komponenter relatert til lukens funksjon (hengsler, foringer, lås etc) må fjærnes før plexiglasset kan gli ut.



Figur 12.4.: Grensesnitt for modul 1 og 2. Rammer rundt åpninger og plassering av pakning markert i rødt.

12.1.3. Overgang testkammer til diffuser

Løsningen på problemet med overgang fra testkammer til diffuser, dvs overgang fra firkantet til sirkulært tverrsnitt, er ikke fullstendig definert. Tanken her er at det fremstilles en overgang lik de som brukes i ventilasjonskanaler (figur 12.5).

En slik komponent regner vi som bestillingsvare og vi har brukt "**Gol Stål AS**" sin priskalkulator for å få en estimert pris i underkant av 1300 kr. Se Vedlegg H. Det vil bli tatt i bruk pakninger her på lik linje som andre grensesnitt.

Figur 12.5.: Eksempel på komponent som endrer tverrsnitt.

12.1.4. Strømretter og nettinger

Strømretteren har som blitt nevnt tidligere ikke blitt modellert da SolidWorks sliter når det kommer til objekter med tusenvis av hull. De har derimot blitt simulert i 12 basert på parametere gjort rede for i 9.

12.2. Computational Fluid Dynamics

For å få en oversikt over hvordan strømningsforholdene kunne blitt gjennom vindtunnelen uten å kunne teste dette mens vi var hjemme brukte vi flowsimulering fra SolidWorks til å gjennomføre CFD (Computational Fluid Dynamics) analyse av vindtunnelen. I begynnelsen brukte vi en maksimalhastighet på vinden som var basert på databladet til viften (Vedlegg L) vi hadde for motoren som sa 15 m/s.

En annen ting som også var nyttig med flowsimuleringen, var at når den var satt opp riktig for tunnelen kan man hente ut resultatene fra strømningene og få de gjort om til krefter som ble påført testmodellen og dermed kraftriggen. Dette kunne da bli brukt til styrkeanalyse av komponentene i kraftriggen for å finne hvilke krefter forskjellige komponenter ble utsatt for.

Simuleringen ble gjort hjemmefra på egne datamaskiner da vi under store deler av gjennomføringen av prosjektet ikke hadde USN sine PC-er på Automasjonslabben tilgjengelig.

Etterhvert brukte vi en simulert vifte basert på 4.11 ytelseskurven som fulgte med databladet for viften. Denne simulerte viften ble da brukt både for simulering av den gamle vindtunnelen som vi allerede hadde fra før, og også til simulering av hvordan den nye vindtunnelen kunne oppføre seg. Med denne flowsimuleringen som har blitt gjennomført kan man få et estimat for hvordan hastighet og trykkfordelingen kan være gjennom tunnelen. Vi har ikke gjennomført noen beregninger som kan brukes til å verifisere om de resultatene fra flowanalysen er realistiske.Vi er klar over at det er en svakhet med prosjektet men vi har nå gått tom for tid. Men vi har også funnet at avviket mellom simulert hastighet gjennom den gamle vindtunnelen og verdiene som er funnet i 9.25 er forholdsvis små.

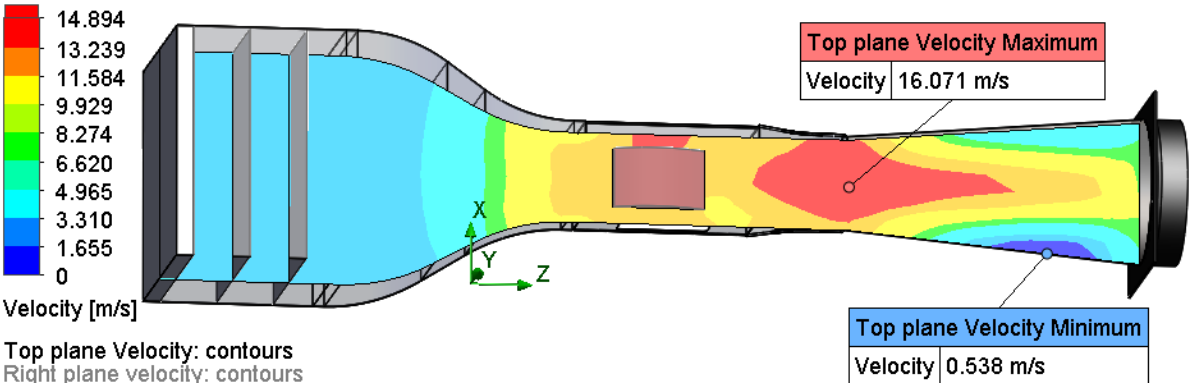
For å sette opp analysen har vi benyttet oss av SolidWorks sine standardinnstillinger på forskjellige punkter. Temperaturen brukt var 20 °C (293.2 K), og et trykk rundt tunnelen på 101 325 Pa som tilsvarer at tunnelen opererer ca ved havnivå. Disse forholdene igjen er med på å påvirke hvordan den simulerte gassen som brukes i simuleringen vil oppføre seg. Vi har valgt å bruke air (luft) som den er satt opp i standardinnstillinger i SolidWorks. Tetthetet på luft ifølge simuleringen er oppgitt til 1.2 kg/m^3 i SolidWorks Flow Simulation.

Etter andre presentasjon ble det også klart for oss at vi trengte å finne en måte å simulere strømningsrettere på. Dette ble da løst ved at det ble benyttet porøse medium i SolidWorks som bestemmer hvordan et fluid slippes gjennom. Vi har ved bruk av den opprinnelige honeycomben

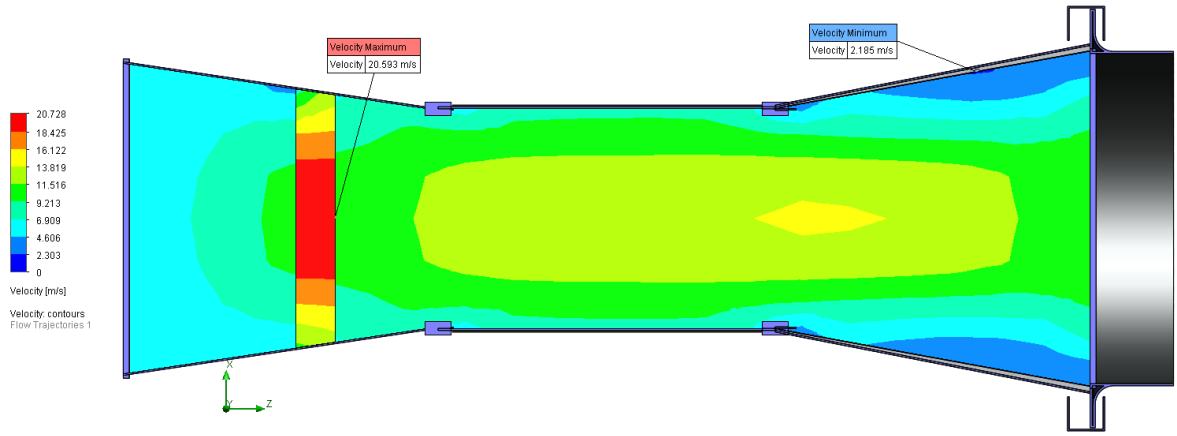
(strømretter) og noen beregninger (Avsnitt 9.7.3) om hvor fine åpninger man trenger gjennom skjermene vi har vurdert å bruke kommet fram til hvor finmaskede skjermene trenger å være. Men med skjermene valgte vi andre dimensjoner på åpningene med utgangspunkt i hva slags skjermer som kan kjøpes.

Når man da har gjennomført denne flowanalysen kan man også få en oversikt over hvordan hastigheten til vinden er fordelt gjennom tunnelen og hvordan trykket forandrer seg gjennom. Der trykket er høyt er vindhastigheten lav og motsatt. Dette bekreftes av Bernoullis prinsipp.

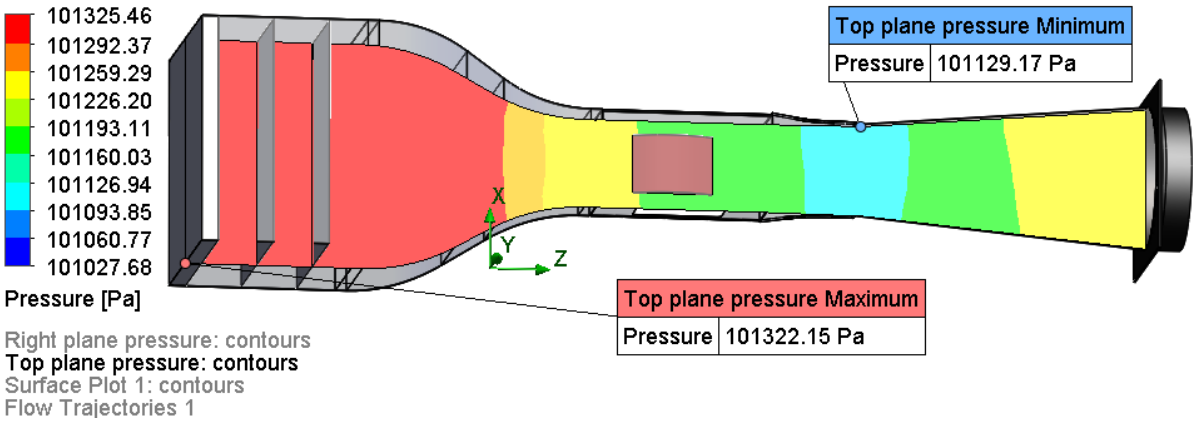
$$\frac{1}{2}\rho a V^2 a + Pa + \rho agha = \frac{1}{2}\rho b V^2 b + Pb + \rho bghb \tag{12.1}$$



Figur 12.6.: Fartsfordeling gjennom ny vindtunnel ovenfra.

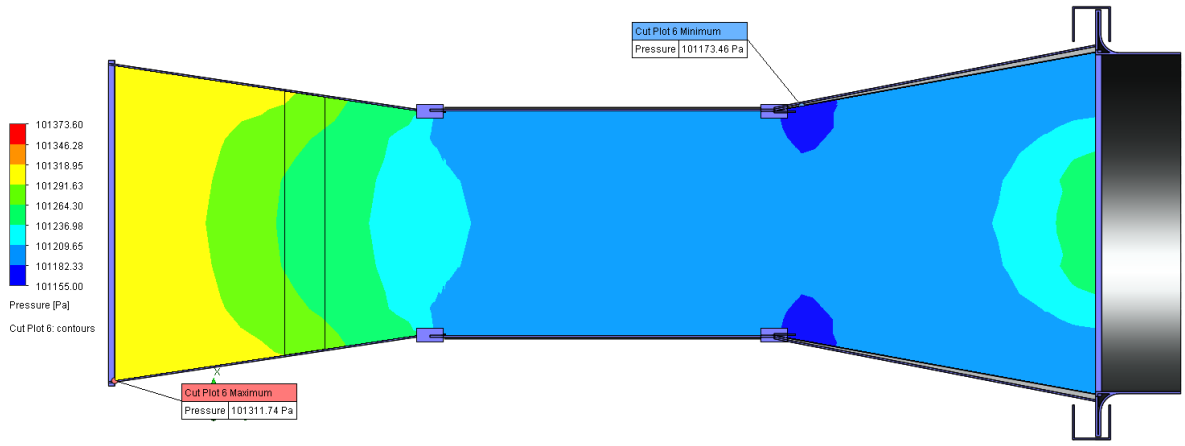


Figur 12.7.: Fartsfordeling gjennom gammel vindtunnel ovenfra.



Figur 12.8.: Trykkfordeling gjennom ny vindtunnel ovenfra.

Det har også vist seg at på grunn av rot med filer og simuleringer er det vanskelig å hente fram

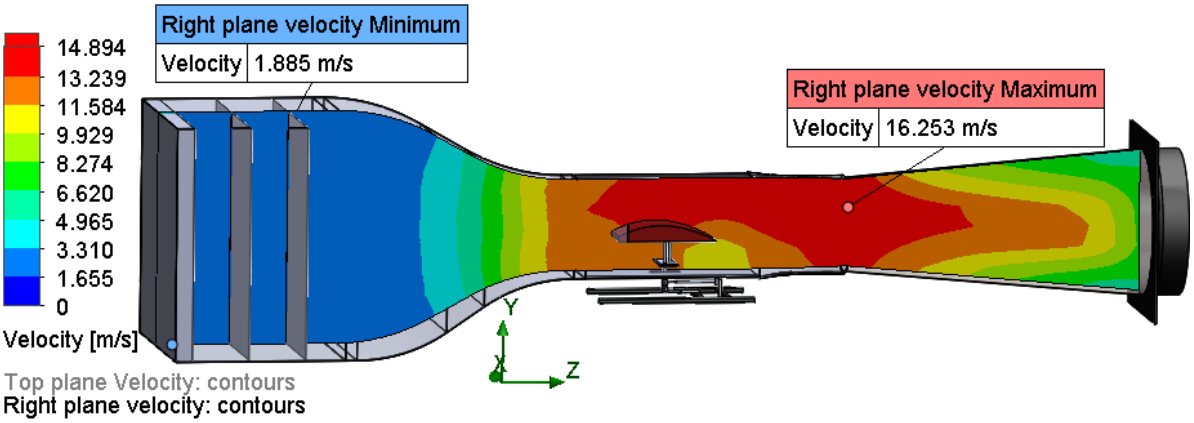


Figur 12.9.: Trykkfordeling gjennom gammel vindtunnel ovenfra.

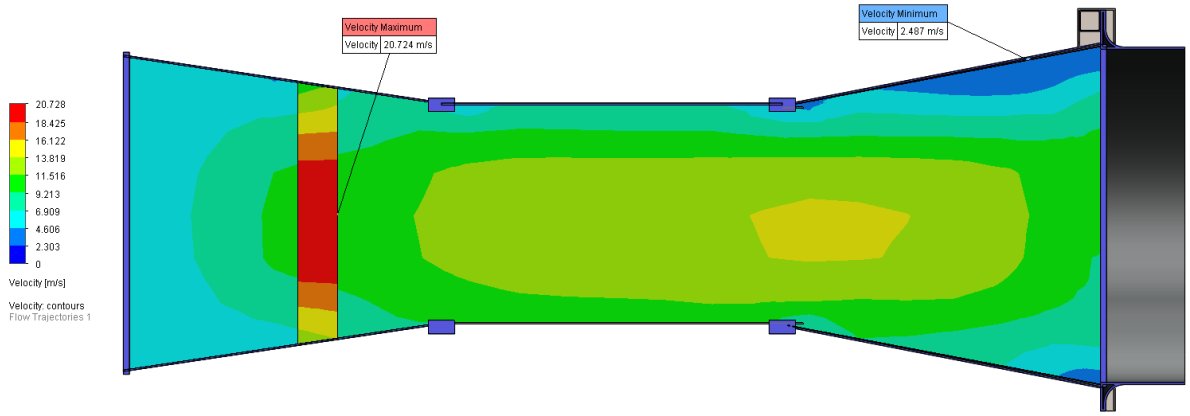
resultater fra tidligere kjøringer.

Dette har bakgrunn i hvordan SolidWorks lagrer data, i motsetning til et skriveprogram som Word. I Solidworks når man lager sammensetninger (assemblies) så lagres det referanser til de opprinnelige delene. Dette gjør at man kan ha mange sammensetninger med de samme delene. Men det betyr også at om man prøver å bruke en eller flere subassemblier mange steder så blir de også oppdatert utifra hvordan de brukes. Dette har ført til at mye data ikke har blitt lagret skikkelig.

Denne flowsimuleringen ble også brukt til å plassere hvor man bør ha den stasjonære fartsmåleren som er tenkt at skal brukes som referanse for fartsreguleringssystemet. Med bakgrunn i resultatene fra analysen kom vi fram til at fartsmåleren kan plasseres ca 2 cm til venstre inn fra lukeåpningen og ellers midt mellom tak og bunn i testkammeret. Der er det en jevn hastighet ifølge simuleringen og den vil kunne ha lite innvirkning på testmodellen slik den har blitt plassert i simuleringen.



Figur 12.10.: Fartsfordeling gjennom ny vindtunnel fra høyre.

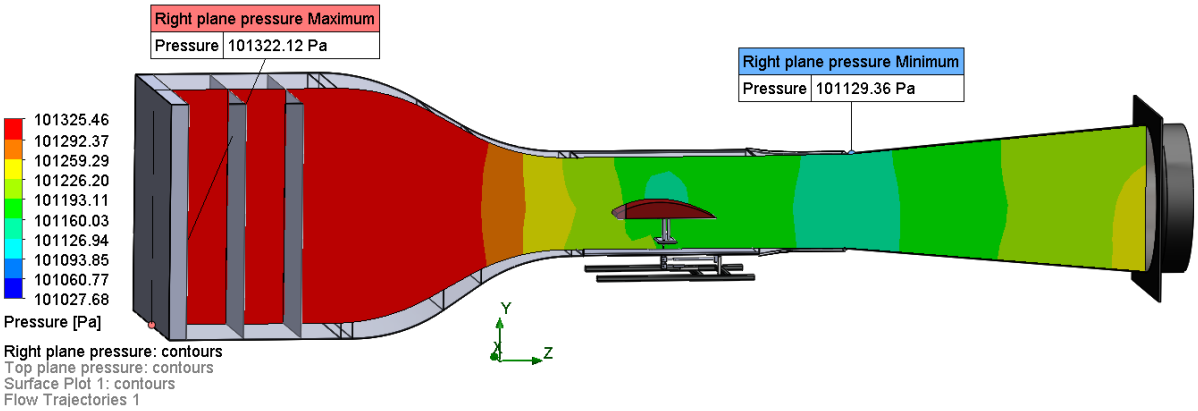


Figur 12.11.: Fartsfordeling gjennom gammel vindtunnel fra høyre.

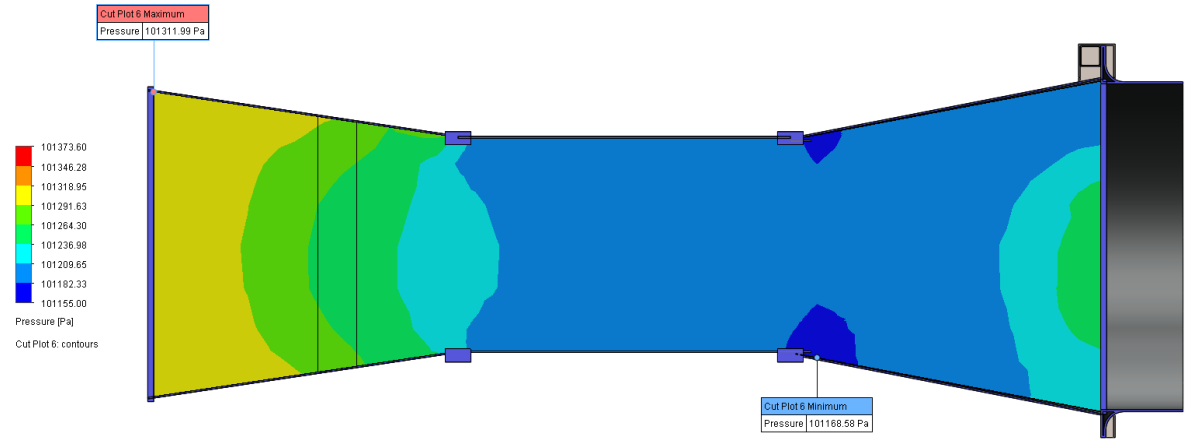
Hastigheten gjennom testkammeret på den nye tunnelen er ifølge simuleringene rundt 13 meter per sekund og er noe raskere enn det vi har klart å måle på vindtunnelen som den er idag. Og hvis estimatene fra vindtunnelflowsimuleringen stemmer er det også forholdsvis jevn hastighet gjennom testkammeret slik det har vært ønsket av oppdragsgiver.

En av tingene som kan observeres på figurene for den nye vindtunnelen som viser hastighet er at man ikke får høyeste hastigheten i testkammeret. Den er i overgangen som vi har lagt inn mellom testkammer og dyse. Det er her tunnelen blir smalest og der det er lettest å få høyest vindhastighet siden tverrsnittet til tunnellen er minst.

Det er også lavest trykk der som henger sammen med høyest hastighet i overgangen mellom



Figur 12.12.: Trykkfordeling gjennom ny vindtunnel fra høyre side.



Figur 12.13.: Trykkfordeling gjennom gammel vindtunnel fra høyre side.

testkammer og diffuser.

I flowsimuleringen av den gamle vindtunnelen kan vi se at vi har fått simulert rundt den samme hastigheten 9.25 den gamle vindtunnelen. Dette tyder på at simuleringen er mer ideel enn den opprinnelige tunnelen, siden den opprinnelige tunnelen ikke viste de såpass høy hastighet. I den simulerte utgaven av den gamle vindtunnelen sitter også honeycomben perfekt inni mot kanten, i motsetning til hvordan den sitter i virkeligheten.

Dette peker på at man bør være varsom med å hevde at dataene fra simuleringene kan regnes som nøyaktige estimater med de dataene vi har funnet. Det trengs noen tilpasninger for å

kunne få et mer nøyaktig estimat med simuleringene. Dette er arbeid som vi ikke har tid til å gjennomføre nå.

12.3. Optimalisering

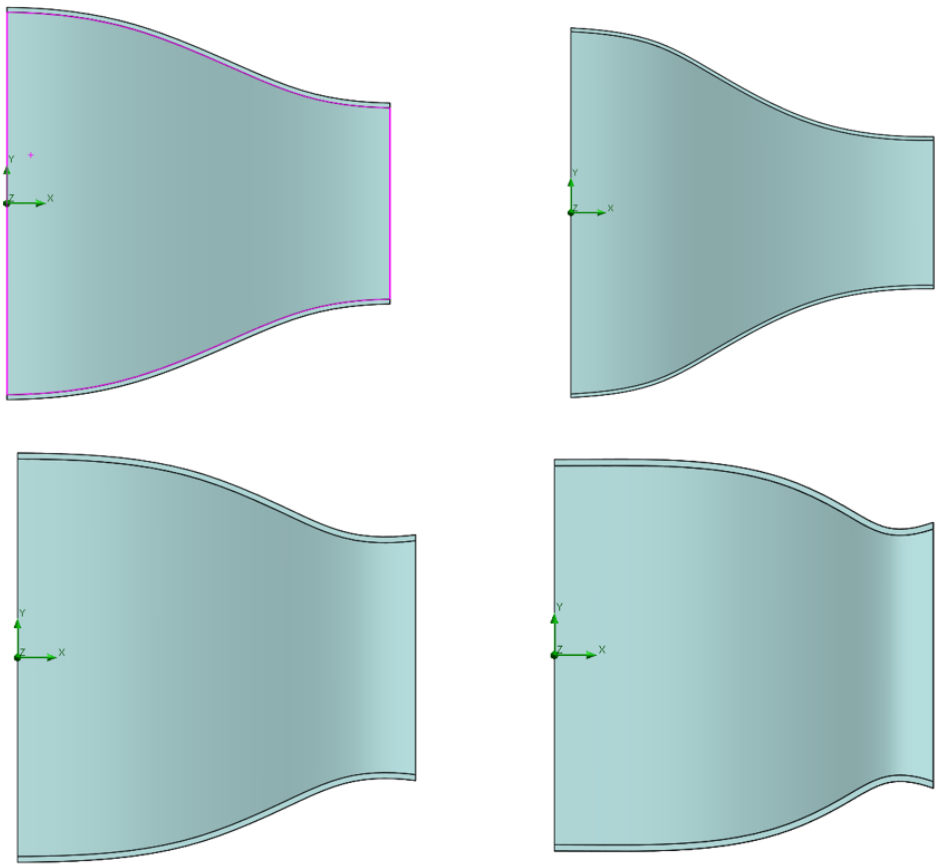
Optimalisering er definert som å finne den best mulige løsningen. Vi har bestemt oss for å kjøre parametrisk optimalisering på dysens kurve og arealforhold. Dette for å finne en utforming som gir best mulig gjennomflyt for luften som skal komprimeres gjennom den. Dette gjennomføres med SolidWorks Flow Simulation (Heretter SWFS) , et tilleggsverktøy i SolidWorks. Verktøyet kjører da gjennom et stort antall intervaller med forskjellige dimensjoner, som vi for eksempel ser på figur 12.14.

12.3.1. Gjøremåte

I tidligere iterasjoner fant vi ut at utgangen til dysen må være firkantet på grunn av testkammeret. Vi vet kurven og forholdstall mellom tverrsnittsarealene til inn og utløp må være velutviklet for å minimalisere turbulens i luftstrømmen.

Parameter	**Current Value**	**Variation Type**	#	Values
D7	40	Range with Number	3	40, 55, 70
D5	379.6	Range with Number	3	329, 455, 582
D6	22	Range with Number	3	22, 36, 50
D1	520	Range with Number	3	460, 490, 520

Tabell 12.1.: Variasjon og Oppsett (Verdier i mm).

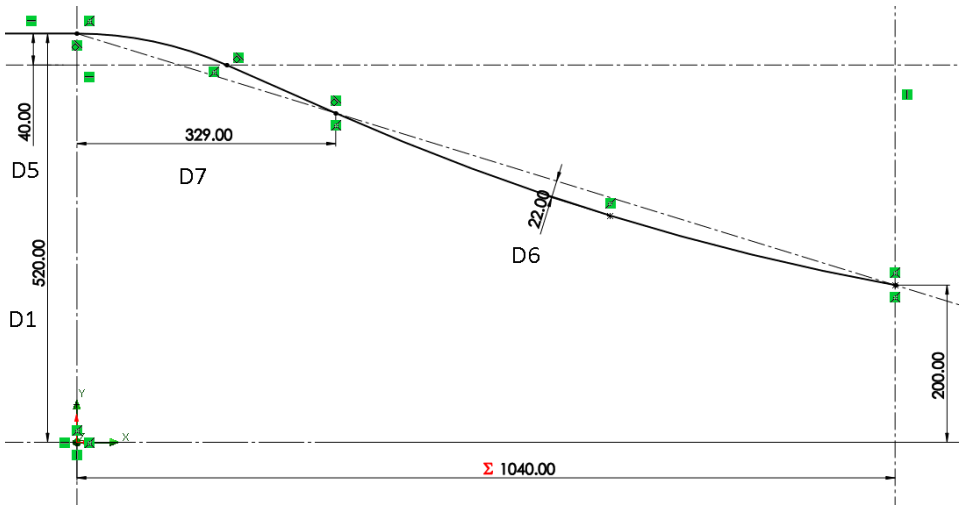


Figur 12.14.: Eksempel på fire iterasjoner med optimalisering.

Vi laget derfor en iterativ analyse med en bestemt variasjon. For eksempel kunne vi brukt variasjon på 1-2 meter med 3 steg. Da får man simulert turbulens for variasjonen 1, 1.5 og 2 meter. Denne variasjonen heter ‘’Range’’ i programmet. Denne variasjonen ser vi i tabell 12.1 .Disse dimensjonene måler blant annet avvik fra en rett linje, og i figur 12.15 ser man oppsettet og visualisering av dimensjonene.

12.3.2. Resultater

På tabell 12.2 ser man SWFS sine kalkuleringer av de forskjellige variasjonene i dimensjoner. Ved funn av mest gunstige “Design point” lager man en ny studie hvor dysens dimensjonene blir erstattet med optimale verdier, og man har optimalisert dysens dimensjoner med tanke



Figur 12.15.: Designkurve brukt i optimalisering.

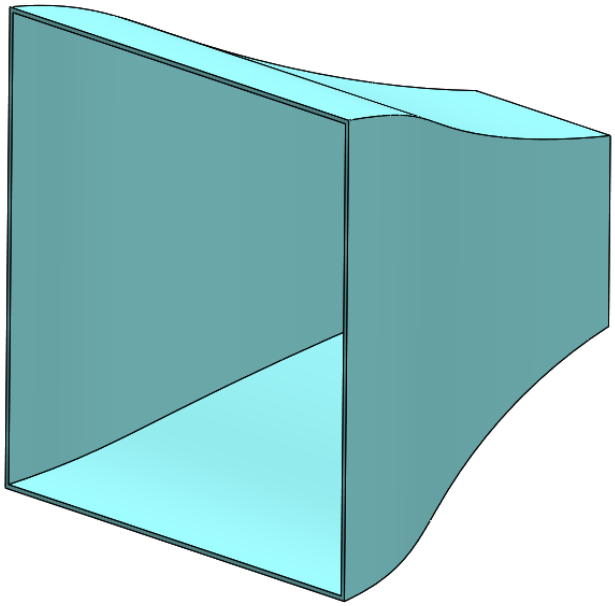
på minimalisering av turbulens. Et design pointer ên variasjon av en av de 4 dimensjonene, 5 hvis man teller med lengden som er satt til å være dobbelt så stor som halve bredden på inntaket.

Summary	Design Point 1	Design Point 2	Design Point 3
D7	40	40	40
D5	329	329	329
D6	22	22	22
D1	460	490	520
Turbulens intensitet (%)	1.035	1.008	0.971

Tabell 12.2.: Design points (Verdier i mm).

12.4. Styrkesimulering

Med bakgrunn i flowsimuleringen som har blitt gjennomført av den designede vindtunnelen var det mulig å bruke disse flowstrømmene som utgangspunkt for de kreftene som testmodell



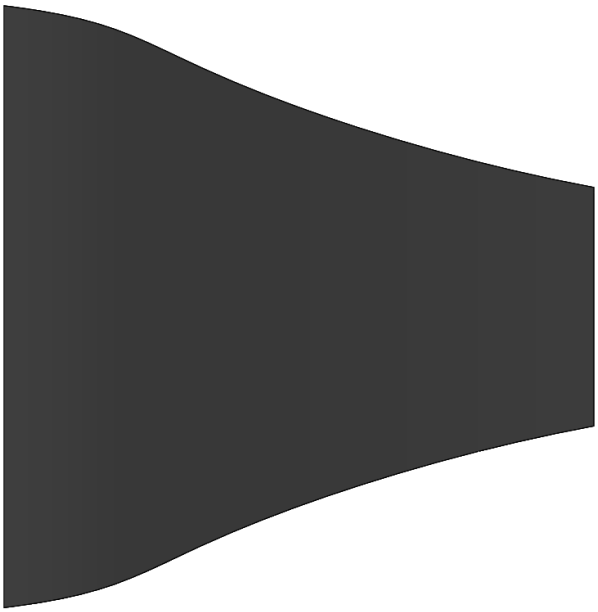
Figur 12.16.: Endelig dyse sett fra et isometrisk perspektiv .

og rigg vil bli utsatt for. Et av de viktigste kravene som har blitt etterspurt av oppdragsgiver er at innfestingen ikke skal bevege seg under utføring av eksperimenter. I tilegg må også systemet som skal måle krefter også selvfølgelig tåle kreftene det blir utsatt for uten å bli deformert permanent eller midlertidig.

Det er tross alt bra å ha et system som kan brukes til å måle kreftene som en modell blir utsatt for, men det er enda bedre å ha et system som skal måle disse kreftene som også tåler å bli utsatt for de kreftene de skal måle uten å gå i stykker eller at de blir forskjøvet så mye at de kan gi feil utslag.

Der kommer styrkeanalysen inn, som også er kjent som en FEM-analyse (Finite Element Method) som gjør en modell om til en samling med mindre elementer som så blir undersøkt for hvordan de reagerer når de blir utsatt for de kreftene som er i simuleringen.

Etter analyse med FEM har det da vært mulig å se at modellen beveger seg som følge av



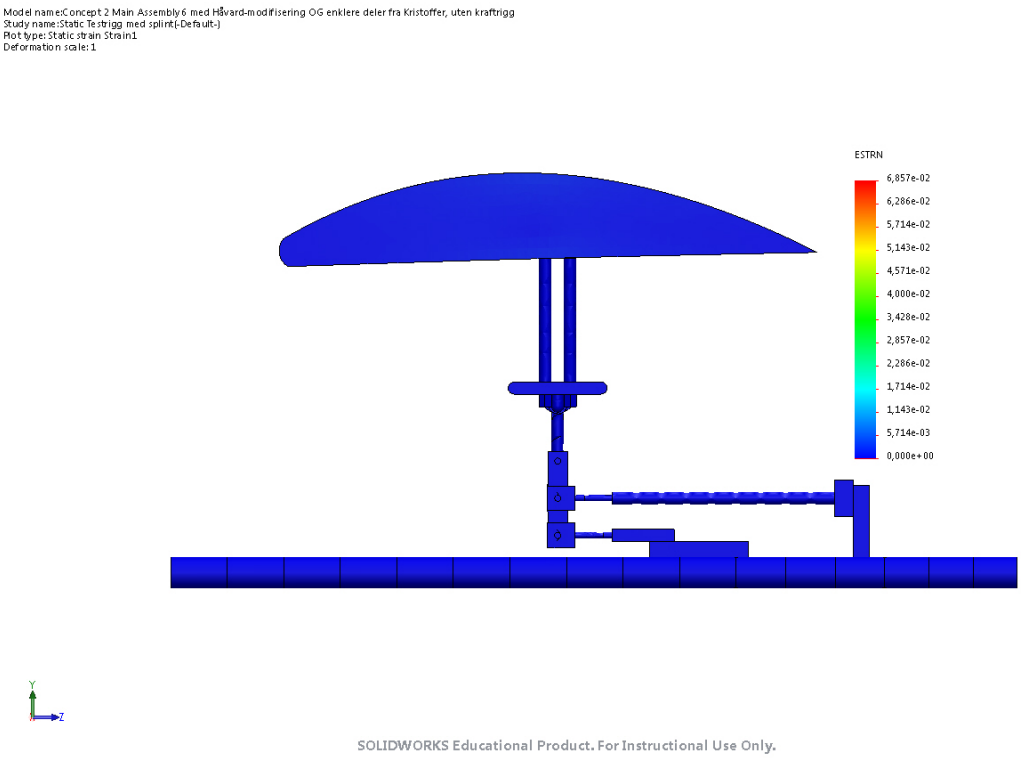
Figur 12.17.: Endelig dyse sett fra siden.

kreftene den blir påført av luftflyten og tyngdekraft gjennom vindtunnelen. Dette fører til at modellen beveger seg på innfestingen, rundt den aksen som har blitt dannet mellom de to festestengene.

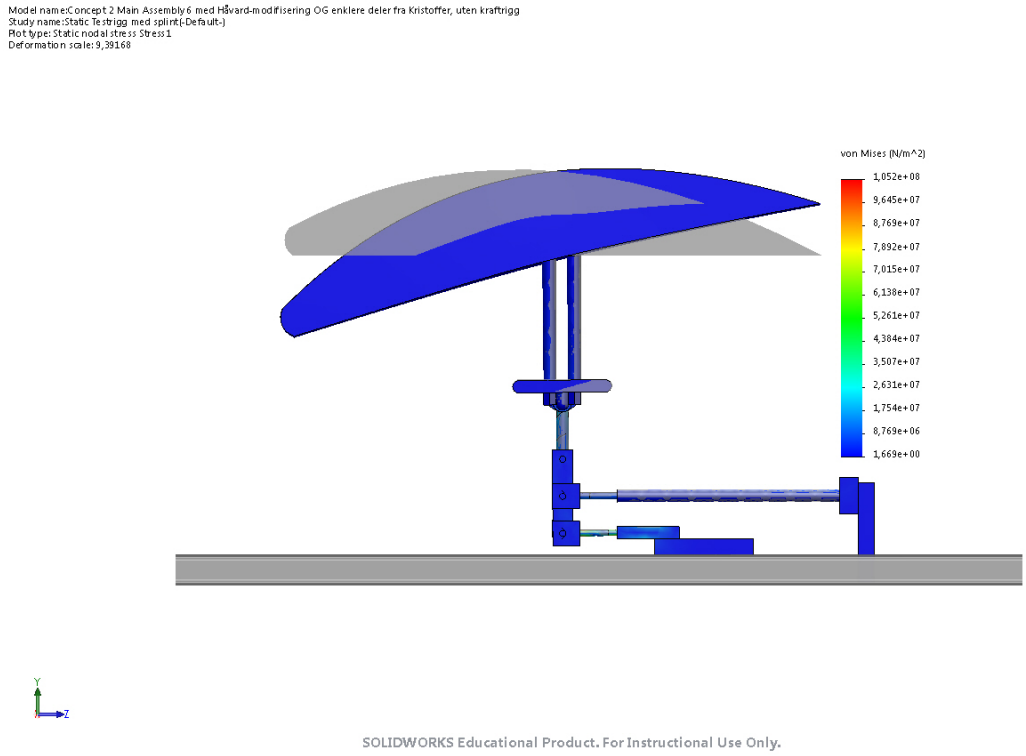
Etter nytt oppsett med ny innfesting beveger fortsatt modellen seg på samme måte, og dette kan dermed tyde på at materialet som har blitt valgt at skal brukes for modellen er den skyldige, siden dette også skjer når det brukes 4 innfestingsstenger framfor bare 2. Etter nøyere gjennomgang av kraftsimuleringen som var involvert kunne man se at det var minimale krefter involvert bortsett fra tyngdekraften. Dette betyr at man ikke har gode resultater fra styrkesimuleringen med tanke på krefter fra strømning.

Et av funnene som ble gjort med kjøringen av FEM-analysen var at modellen som ble brukt hadde anledning til å vippe om en linje mellom løftestengene som ble brukt. Mellom stengene ble det dannet en akse som modellen kunne vippe rundt. Et omdesign har da brukt 4 stenger sånn at modellen kan bli bedre holdt fast og ikke anledning til å kunne vippe rundt den samme aksen. Dette later til å ikke hatt stor effekt.

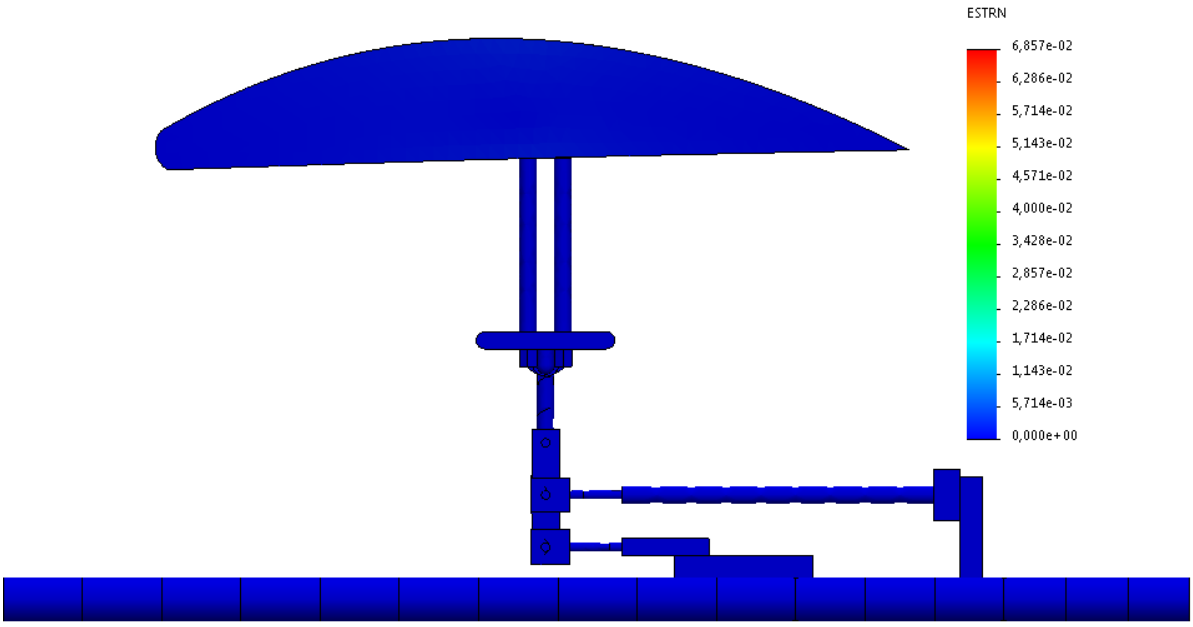
Et annet problem som vi har oppdaget etter å ha sett nærmere på simuleringen er at modellen blir utsatt for minimale krefter og later til å være lite påvirket. Det som har påvirket disse modellene mest er tyngdekraften. Som er satt til å være 9.81 m/s^2.



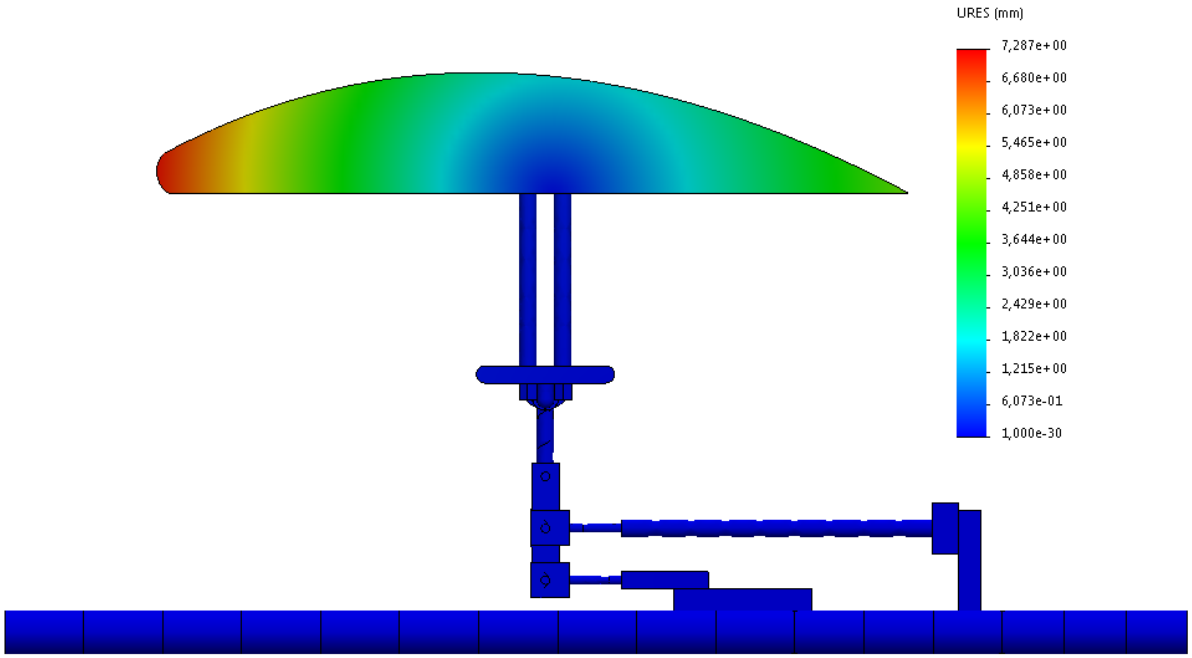
Figur 12.18.: Strekking i modell som følge av krefter påført.



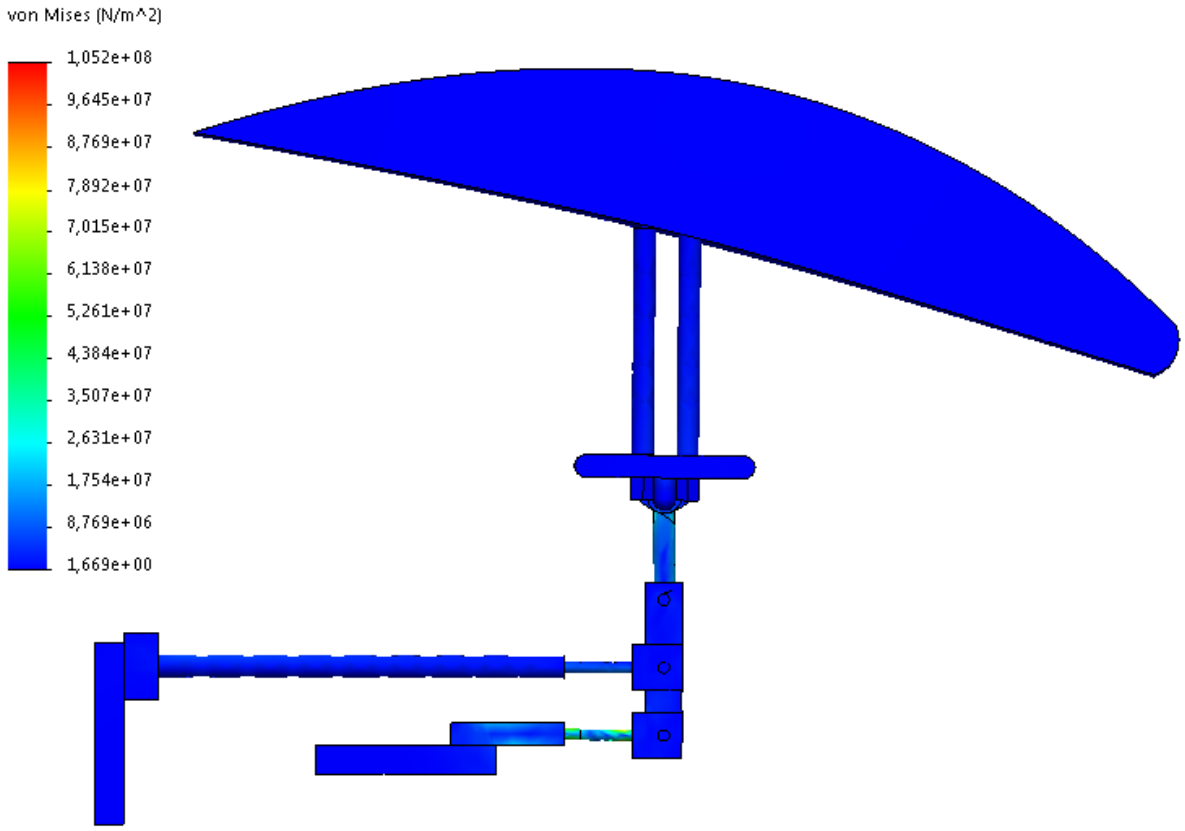
Figur 12.19.: Bevegelse av komponenter som følge av krefter påført.



Figur 12.20.: Forflyttelse av modell med bakgrunn i krefter påført.



Figur 12.21.: Hvor mye komponentene har flyttet seg som følge av påførte krefter.



Figur 12.22.: Stressfordeling i figur basert på påførte krefter.

13. IDM-programvare

I IDMen er programvaren til den innebygde datamaskinen delt i to, et program som kjører på Arduinoen og et på Raspberry Pien.

13.1. Arduino

Målinger og avlesing av sensordata foregår på Arduinoen, men på grunn av arkitekturen til Arduino-kortet kan det kun kjøres én prosess som består av en evig sløyfe som utfører funksjonen `loop()` kontinuerlig. All lesing av sensordata, regulering av prosessparametre, og kommunikasjon må foregå i denne sløyfa.

For å unngå blokkering, og fordi forskjellige funksjoner trenger forskjellig eksekveringsintervall, kan ikke `delay()`-funksjonen benyttes til å styre timingen i programmet. Isteden sammenliknes tiden hver gang loopen kjører med tiden sist gang en funksjon ble utført i en `if (current_time - previous_time >= interval)`-test.

13.1.1. Innsamling av sensordata

Sensorer leses i `loop()`-funksjonen med et samplingsintervall på 200 ms, med unntak av lufttemperatursensoren, som på grunn av hardwarebegrensninger kun kan leses hvert 2000 ms. Målingene fra sensorene lagres i en `struct` som vist i Kodeutdrag 6.

```
struct SensorData {
    int windSpeed;
    int temperature;
    int humidity;
    int pitch;
    int airPressure;
    int dragForce;
    int liftForce;
    bool hatchClosed;
    bool fanRunning;
};
```

Kodeutdrag 6: Struct som lagrer sensordata på Arduinoen.

Sanntidsklokke

IDMen benytter en batteridreven sanntidsklokkemodul til å angi tidspunktet for målingene. Koden er basert på eksempelkoden i [18].

13.1.2. Kommunikasjon med Raspberry Pi

Kommunikasjonen med Raspberry Pien foregår via serieportkommunikasjon over USB som beskrevet i Avsnitt 10.1. På Arduinoen benyttes funksjonene i det innebygde `Serial`-biblioteket.

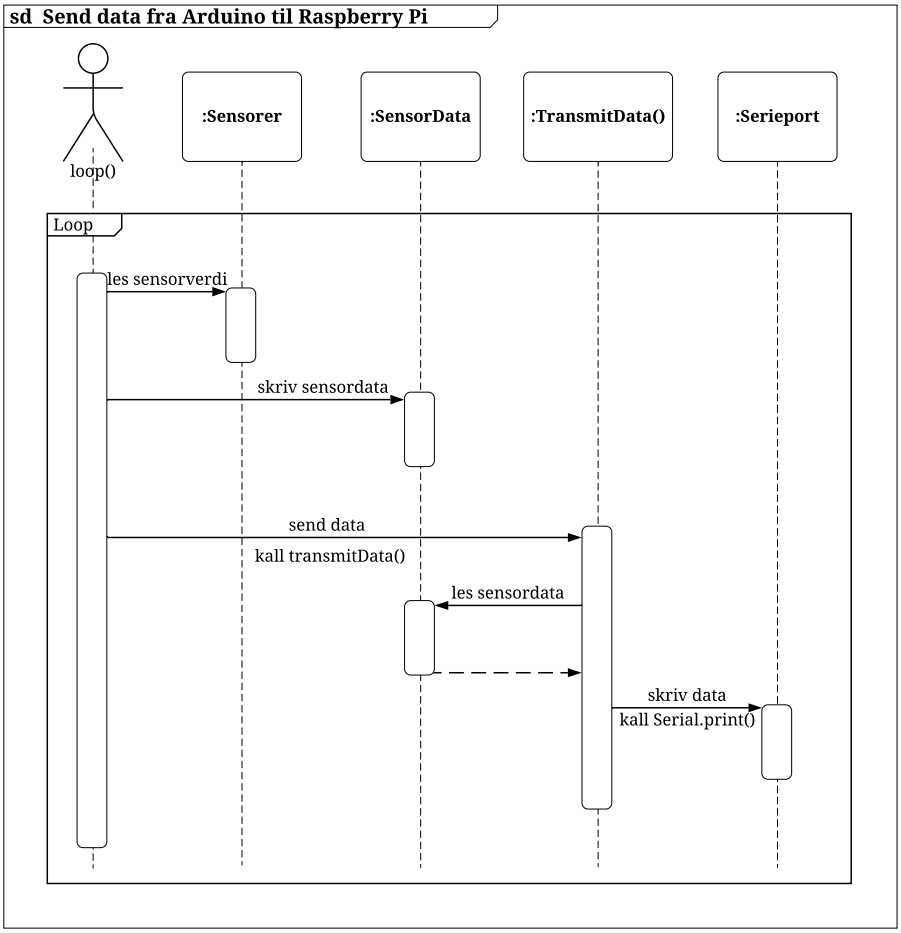
Sending av data utføres av `transmitData`-funksjonen (Kodeutdrag 7), som leser `sensorData`-structen og tiden fra sanntidsklokka, og sender det som en linje[1] ved hjelp av `Serial.print()`. Se også Figur 13.1

Når Arduinoen mottar meldinger på serieporten, kalles den innebygde funksjonen `serialEvent()` automatisk. Denne leser hver karakter som mottas og legger disse i en string som termineres av tegnet `X`. Se Kodeutdrag 8 og 9. Meldingene som mottas forventes å være på for-

[1] Serial.print()-kommandoen sender ikke en linjeskift når den er ferdig, det gjøres av Serial.println() på slutten av funksjonen. Derfor sendes hele datapakken som én linje

```
void transmitData(struct SensorData data, char* datestring) {
    Serial.print(datestring); Serial.print("|");
    Serial.print(data.windSpeed); Serial.print("|");
    Serial.print(data.temperature); Serial.print("|");
    Serial.print(data.humidity); Serial.print("|");
    Serial.print(data.pitch); Serial.print("|");
    Serial.print(data.airPressure); Serial.print("|");
    Serial.print(data.dragForce); Serial.print("|");
    Serial.print(data.liftForce); Serial.print("|");
    Serial.print(boolToByte(data.hatchClosed, data.fanRunning));
    Serial.println();
}
```

Kodeutdrag 7: TransmitData-funksjon for sending av måledata fra Arduino til Raspberry Pi.



Figur 13.1.: Sekvensdiagram for sending av data fra Arduino til Raspberry Pi.

men `WnnX` eller `PnnX` der W og P definerer om det mottas et settpunkt for vindhastighet eller pitchvinkel, og nn er den nåværende verdien. Meldingsformatet er illustrert i figur 10.2 på side 208.

13.2. Raspberry Pi

Raspberry Pien kjører programmet som håndterer brukergrensesnittet og lagringen av sensordataene til en database som data kan eksporteres fra. Programmet er skrevet i Python versjon 3.7[8] fordi det var det mest tilgjengelige når det kom til å lage et grafisk brukergrensesnitt. Det er fleksibelt nok til å utføre alle oppgavene programmet må utføre, og vi hadde erfaring med det.

13.2.1. Kommunikasjon med Arduino

For å kommunisere med Arduinoen brukes pyserial-biblioteket[17], som i IDMen er bygd inn i pakken `idmserial`. Denne implementerer `SerialCommunicator`-klassen (Figur 13.5 og Vedlegg B.2), som inneholder to andre komponentklasser: `SerialReceiver` og `SerialTransmitter`, for henholdsvis mottak og sending av meldinger.

I `SerialCommunicator` instansieres sender- og mottakerobjektene, og en referanse til serieporten sendes med (Kodeutdrag 10). `stop_event` er et `threading.Event()`-objekt som brukes til å sende en melding til `SerialReceiver` for å stoppe tråden når programmet skal avsluttes og hindre at en hodeløs prosess kjører i bakgrunnen.

```
void serialEvent() {
    while (Serial.available()) {
        // get the new byte:
        char inChar = (char)Serial.read();
        // add it to the inputString:
        if (inChar != '\n') {
            inputString += inChar;
        }
        // if the incoming character is a newline, set a flag so the main
        ↪  loop can do something about it:
        if (inChar == 'X') {
            stringComplete = true;
        }
    }
}
```

Kodeutdrag 8: SerialEvent-funksjon for mottak av settpunkter.

```
if (stringComplete) {
    switch (inputString.charAt(0)) {
        case 'W': case 'w':  // Received new windspeed set point
            inputString.remove(0,1);
            inputString.remove(inputString.length());
            inputData.setWindSpeed = inputString.toInt();
            break;
        case 'P': case 'p':  // Received new pitch angle set point
            inputString.remove(0,1);
            inputString.remove(inputString.length());
            inputData.setPitch = inputString.toInt();
            break;
        default:
            break;
    }
    inputString = "";
    stringComplete = false;
}
```

Kodeutdrag 9: Utdrag fra loop() som lagrer mottatte settpunkt.

```
# class SerialCommunicator
def __init__(self, data, stop_event,
                port='/dev/ttyACM0', rate=57600, timeout=1):
    self.data = data
    self.stop_event = stop_event

    try:
        self.ser = serial.Serial(port=port, baudrate=rate, timeout=timeout)
    except serial.SerialException:
        print("Exception in serialport")

    try:
        self.transmitter = self.SerialTransmitter(self.ser)
    except:
        print("Exception in transmitter")

    try:
        self.receiver = self.SerialReceiver(self.ser, self.data,
        ↪  self.stop_event)
        self.receiver.start()
    except:
        print("Exception in receiver")
```

Kodeutdrag 10: Init-funksjonen til SerialCommunicator.

SerialReceiver

Mottakerklassen er implementert som en underklasse av `Thread`-klassen fra `threading`-biblioteket til Python. Da kan hele klassen kjøres som en egen tråd i programmet, slik at dataene mottas når de kommer og ikke risikerer at en annen del av programmet blokkerer mottakerfunksjonen.

13.2.2. SerialTransmitter

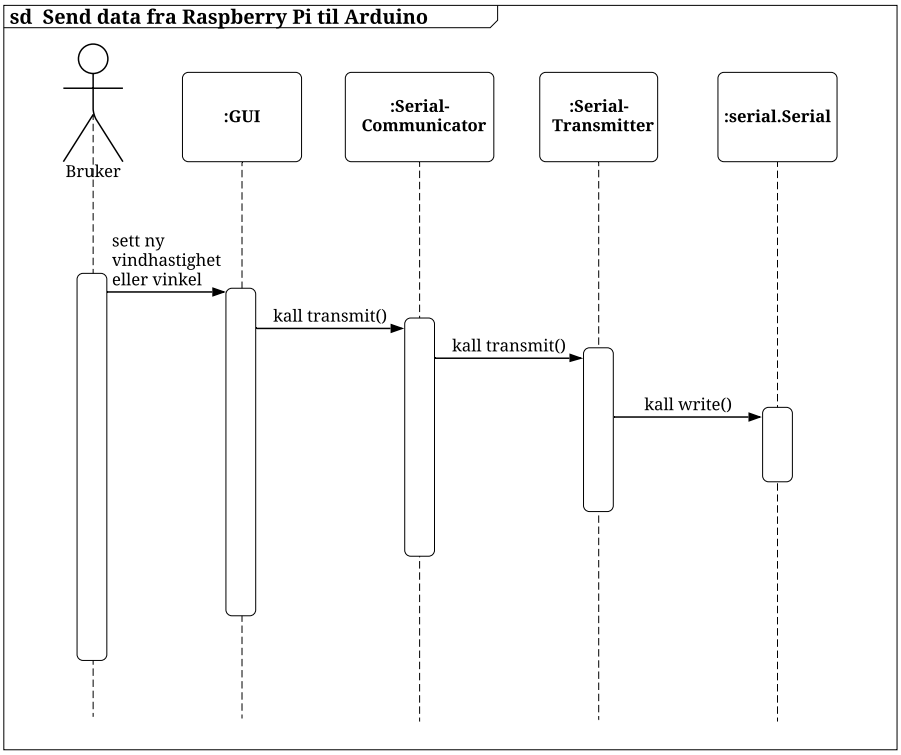
`SerialTransmitter`-klassen inneholder en `transmit`-funksjon (Kodeutdrag 12) som utfører to oppgaver: Først sjekker den at strengen som sendes avsluttes med en `X`, så sendes strengen over serieporten. Sekvensen til avsendingen er illustrert i Figur 13.2 og meldingsformatet er illustrert i Figur 10.2

```
class SerialReceiver(threading.Thread):
    def __init__(self, serialport, data, stop_event):
        self.ser = serialport
        self.data = data
        self.stop_event = stop_event
        threading.Thread.__init__(self)

    def run(self):
        while not self.stop_event.is_set():
            in_data = self.ser.readline()
            if len(in_data) > 0:
                self.data.parse_datastring(in_data)
```

Kodeutdrag 11: SerialReceiver-klassen i idmserial.

```
class SerialTransmitter:
    def __init__(self, serialport):
        self.ser = serialport

    def transmit(self, message):
        # Ensure message is properly terminated
        if message[-1:] != "X":
            if message[-1:] == "x":
                message = message[:-1]
            message = message + "X"
        self.ser.write(message.encode())
```

Kodeutdrag 12: SerialTransmitter-klassen i idmserial.



Figur 13.2.: Sekvensdiagram for sending av data fra Raspberry Pi til Arduino.

13.2.3. Lagring av sensordata

Når sensordataene mottas på Raspberry Pien lagres de i et DataObject-objekt (Vedlegg B.3) der strengen som ble sendt over serieporten parses og deles opp i individuelle variabler av `parse_datastring`-funksjonen (Kodeutdrag 13).

Dataene som er lagret i dataobjektet kan hentes ut via `get_data`-funksjonen, som returnerer et Python dictionary (Kodeutdrag 14).

Siden seriemottakeren er asynkron er både parse- og get-funksjonene låst med et `threading.RLock`-objekt. Denne låsen kan bare eies av en tråd om gangen, slik at når et nytt datapunkt mottas kan ikke data hentes ut fra dataobjektet før `parse_datastring`-funksjonen er ferdig. Vi hindrer dermed konflikter som kan blokkere skrivefunksjonen og at lesefunksjonen kan lese data fra to forskjellige datapunker dersom en skriveoperasjon er halvveis.

Database

Databasefilen opprettes i hovedklassen `IDMGUI` i `GUI.py` ved hjelp av `sqlite3`-biblioteket.

Programmet oppretter en databasefil og deretter et `cursor`-objekt som eksekverer SQL-spørringer i databasen (Kodeutdrag 15). Denne pekeren lagres som et medlemsobjekt av hovedklassen slik at den kan hentes i underklasser og andre funksjoner som trenger tilgang til databasen.

Under initialiseringen lages `data`-tabellen i databasen dersom den ikke eksisterer, og endringene lagres deretter til databasefila gjennom `self.database.commit()`-funksjonen.

Når data skal hentes ut av databasen, eksempelvis i Kodeutdrag 16, hentes en referanse til cursoren ut fra `IDMGUI`, spørringen uføres ved hjelp av `cursor.execute()`, og resultat av spørringen hentes ut via `cursor.fetchone()`/`cursor.fetchall()`

```
def parse_datastring(self, data):
    with self.__lock:
        try:
            (datestring, windspeed, temperature, humidity, pitch,
            ↪  airpressure,
                dragforce, liftforce, bools) = data.split(b"|")
        except ValueError:  # Potential missing bools variable, try again
            try:
                (datestring, windspeed, temperature, humidity, pitch,
                ↪  airpressure,
                    dragforce, liftforce) = data.split(b"|")
                bools = -1
            except ValueError:  # Received string is in wrong format
                datestring, windspeed, temperature = b"1900-01-01T12:00:00",
                ↪  -1, -1
                humidity, pitch, bools = -1, -1, -1
                airpressure, dragforce, liftforce = -1, -1, -1

        if isinstance(self.__datestring, str):
            # Make sure date string is not in bytes format
            self.__datestring = datestring.decode("utf-8")
        else:
            self.__datestring = datestring

        if self.re.match(self.__datestring):
            # Make sure date string is in the correct ISO-8601 format
            self.__datetime =
            ↪  datetime.datetime.fromisoformat(self.__datestring)
        else:
            self.__datetime =
            ↪  datetime.datetime.fromisoformat("2100-01-01T12:00:00")

        self.__windspeed = int(windspeed)
        self.__temperature = int(temperature)
        self.__humidity = int(humidity)
        self.__pitch = int(pitch)
        self.__airpressure = int(airpressure)
        self.__dragforce = int(dragforce)
        self.__liftforce = int(liftforce)
        self.__bools = int(bools)
```

Kodeutdrag 13: parse_datastring-funksjonen i dataobject.

```
def get_data(self):
    with self.__lock:
        return dict(time=self.__datetime, timestring=self.__datestring,
        ↪  windspeed=self.__windspeed,
                    temperature=self.__temperature, humidity=self.__humidity,
                    pitch=self.__pitch, airpressure=self.__airpressure,
                    dragforce=self.__dragforce, liftforce=self.__liftforce)
```

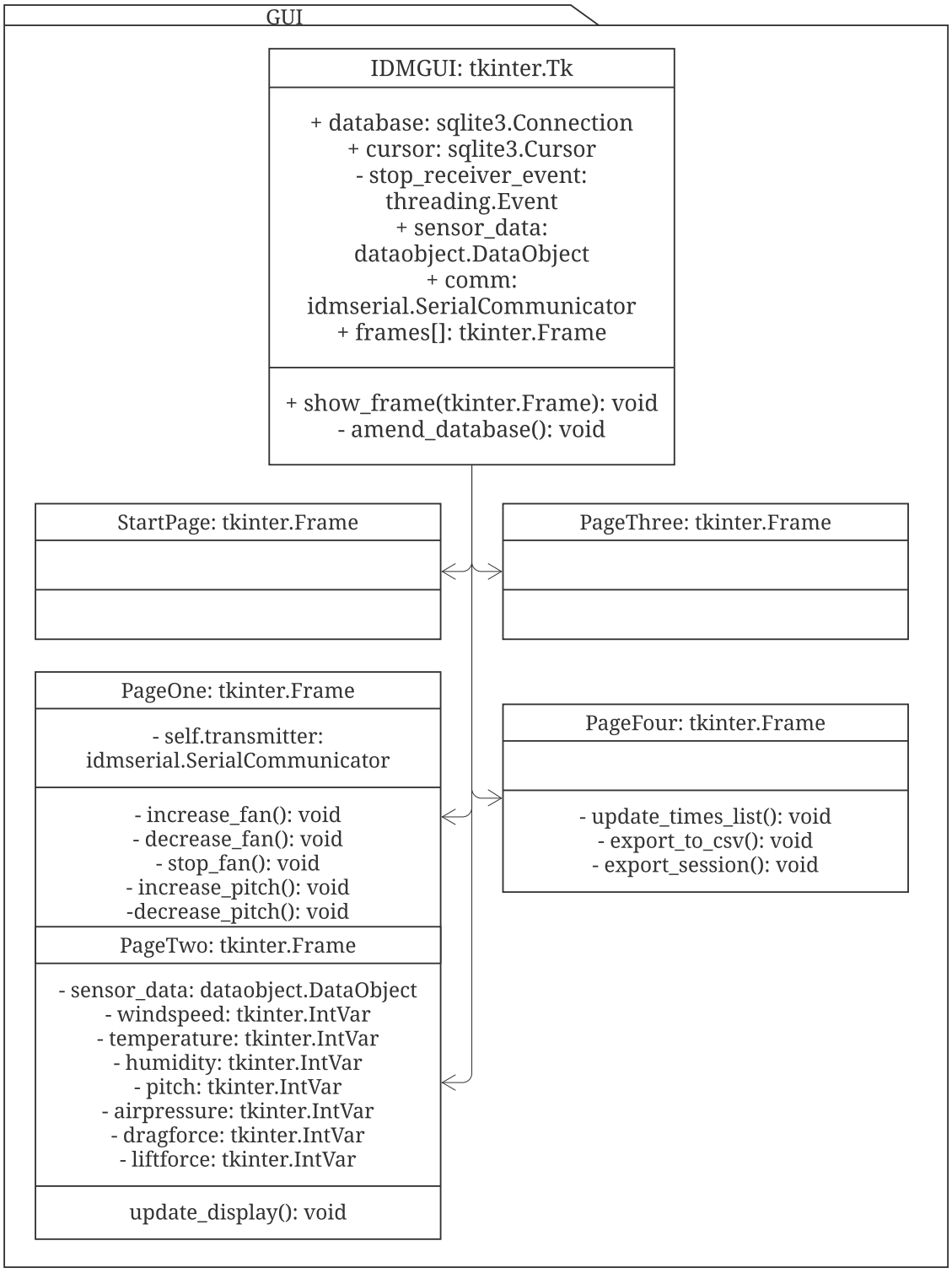
Kodeutdrag 14: get_data-funksjonen i dataobject.

```
#  Make a new database file for each month
db_month = datetime.date.today().isoformat()[:-3]
db_folder = f"/usr/db/idm/"
#  Don't store more than 6 months of data. Delete the oldest database file
↪  when it turns over.
dbs = [f for f in os.listdir(db_folder) if os.path.isfile(f) and f[-3:] ==
↪  ".db"]
if len(dbs) > 5:
    f = dbs.pop(0)
    if f != f"data_{db_month}.db":
        os.remove(dbs[0])
#  Connect to database
self.database = sqlite3.connect(db_folder + f"data_{db_month}.db")
self.cursor = self.database.cursor()
#  Check if data table exists in database and create it if not
self.cursor.execute("""CREATE TABLE IF NOT EXISTS data
    (time date, windspeed int, temperature int, humidity int, pitch int,
    airpressure int, dragforce int, liftforce int)""")
self.database.commit()
```

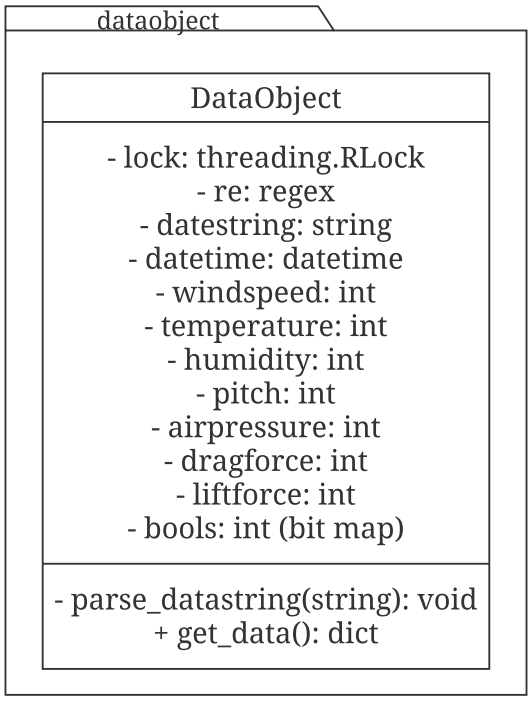
Kodeutdrag 15: Initialisering av database.

```
c = controller.cursor
c.execute('SELECT time FROM data')
cblist = c.fetchall()
```

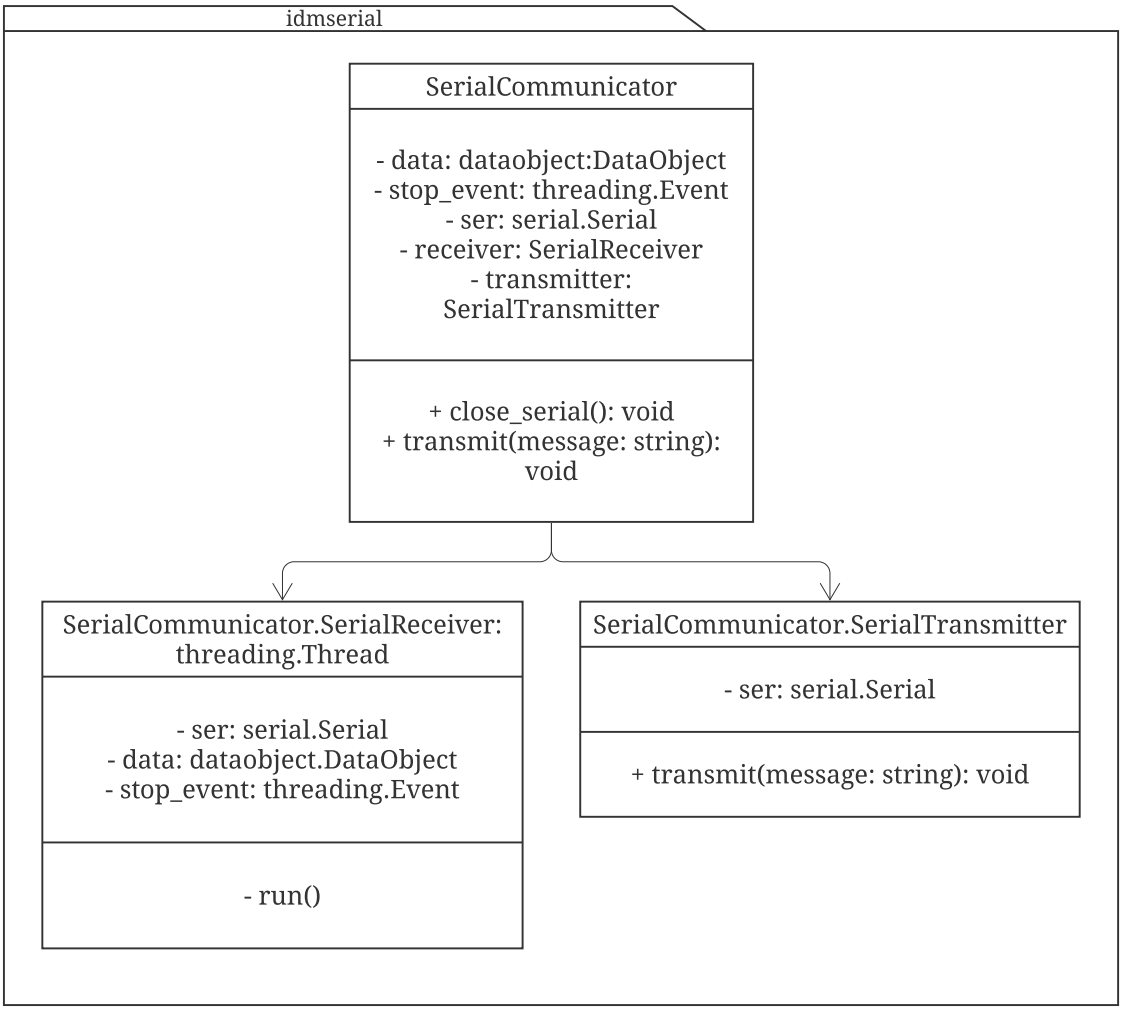
Kodeutdrag 16: Eksempel på uthenting fra database.



Figur 13.3.: Klassediagram for grafisk brukergrensesnitt.



Figur 13.4.: Klassediagram for dataobjectpakke dataobject.



Figur 13.5.: Klassediagram for seriekommunikasjonspakke idmreceiver.

13.3. Brukergrensesnitt

Intro

Vi hadde en plan for hvordan vi ønsket at brukergrensesnittet skulle se ut, og hva slags funksjoner den skulle inneholde i brukergrensesnittet, se 10.1.3. Vi hadde en del endringer utifra konseptet vi hadde på brukergrensesnittet, blant annet ved eksporteringen. Vi endte opp med å lage en database som sendte til en webserver som opprinnelig ikke var planlagt, men viste seg å være en god løsning. Det gjør at brukeren ikke trenger å koble seg på vindtunnelen med sin egen PC for å få ut dataene. For du kan nå enkelt gå på webserveren og hente den.

Tkinter

Tkinter er et brukergrensesnitt verktøy som er laget til Python, hvor det er relativet enkelt å sette opp et brukergrensesnitt med forskjellige sider og graffremvisning. Valget med å ta i bruk Tkinter kom av at det var enkelt å koble opp med Arduino, samt mange kilder på nett om hvordan man skulle få til funksjoner for brukergrensesnittet. Men vi fant ut at Tkinter ikke var ideell for å kjøre kontinuerlige funksjoner, hvor vi for eksempel testet å legge inn data kontinuerlige i en liste eller animerte en graf. Da ville programmet enten henge seg etter kort tid eller jobbe sakte. Måten vi løste det på var å legge inn en oppdateringsknapp for listen, og med grafen åpner vi heller et helt nytt vindu hvor grafen kan jobbe individuelt ifra hovedprogrammet.

13.3.1. Gjennomføring

Til å begynne med måtte vi sette opp grunnlaget for brukergrensesnittet, og vi vet at det kom til å være flere sider som i eksemplet på brukergrensesnittet10.1.3. Så vi satt det opp slik at det kom til å være enkelt å legge inn flere sider. Vi begynte med å sette opp klassen for brukergrensesnittet: Dette gjorde at vi kunne begynne å sette opp flere sider i brukergrensesnittet, og

```
class IDMGUI(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs) #Initialiserer vi den arvede
        ↪ klassen
        container = tk.Frame(self) #Setter opp frame for å organiserer
        ↪ widgets(eks knapper elle labels)
        container.pack(side="top", fill="both", expand = True) #Bruker pack
        ↪ for å plassere widgets
        container.grid_rowconfigure(0, weight=1) #Ekspanderer rader med
        ↪ vinduet
        container.grid_columnconfigure(0, weight=1) #Ekspandere kolonner med
        ↪ vinduet
```

Kodeutdrag 17: Grunnlag for sidene Del-1.

da begynte vi å sette opp antall sider. Det gjør vi ved å definere en tom ordbok som vi fyller inn etterhvert. Det finnes også to måter å kunne plassere widgets på, enten med grid(), eller med pack(). Forskjellen er at med grid() så kan du plassere widgets og tekst, hvor du selv vil, ved å fortelle i hvilken row og column det skal plasseres. Mens pack() plasserer widgets hierarkisk.

Dermed kan vi enkelt sette opp så mange sider vi ønsker for brukergrensesnittet ved dette oppsettet, og måten vi lager en ny side på er da:

Til slutt har vi et brukegrensesnitt som er veldig liknende slik vi planla i design fasen for brukergrensesnittet som se slikt ut:

1. Forside: Figur 13.6.

```
    self.frames = {} #tom ordbok
    for F in (StartPage, PageOne, PageTwo, PageThree,PageFour): #Definerer
    ↪  antall sider som skal lages i programmet
        frame = F(container, self) #definerer hva framen er
        self.frames[F] = frame #PLasserer antall sider i frame
        frame.grid_rowconfigure(0, weight=0) #Gjør at alt expander av seg
        ↪  selv grid
        frame.grid_columnconfigure(0, weight=1) #Ekspandering i Grid
        frame.grid(row=0, column=0, sticky="nsew") #Plassere widget med grid
    self.show_frame(StartPage) # Viser første side
    self.title("IDM") #Titel på vindu

def show_frame(self, cont): #Vis frame funksjon
    frame = self.frames[cont] #Definerer frame som self.frame fra ordboken
    ↪  med cont som brukes i kallet av siden
    frame.tkraise()# Henter siden og bringer til fram
```

Kodeutdrag 18: Grunnlag for sidene Del-2.

```
class StartPage(tk.Frame): #Setter klasse navn og kaller frame
    def __init__(self, parent, controller): #Initialiserer vi den arvede
    ↪  klassen
        tk.Frame.__init__(self,parent) #Initialiserer frame
```

Kodeutdrag 19: Grunnlag for sidene Del-3.

2. Endre hastighet og pitch: Figur 13.7.

3. Målinger: Figur 13.8.

4. Graf visualisering: Figur 13.9.

5. Røykprobe: Figur 13.10.

6. Eksportering: Figur 13.11.

13.3.2. Skjerm

Oppsettet av skjermen var enkel, det var bare å koble den opp på Raspberry pien og installere biblioteket for keyboardet. Det var «Plug and play» med skjermen. For Raspberry-pien kjenner Linux(operativsystem) igjen dette som en annen monitor, og noen av utfordringen var da å kunne gjøre det slikt at software startet på touch-skjermen, og ved det brukte vi geometri posisjon i Tkinter. Måten vi gjør dette på er slikt: Her sier vi at oppløsningen skal være 800x480 piksler som er oppløsningen på skjermen vi har, å at Y og X posisjonen er 0, som vil tilsi at den blir puttet lengest øverst til venstre i skjermen. Årsaken til det kommer av den andre skjermen, siden touchskjermen blir regnet som en sekundær skjerm av Linux.

13.3.3. Animert graf

For den animerte grafen våres har vi brukt matplot animasjons-biblioteket som samkjører med Tkinter programvaren. Vanligvis med matplot ville man plottet det i ett figurvindu som matplot skaper. Men siden man har mindre kontroll over vinduplasseringen eller at det kan lukkes med en knapp,har vi integrert den i et Tkinter vindu og det gjør vi slikt at den enkelt plottet i det som heter et «Canvas» i Tkinter. Det gjør at vi da kan animere grafen i et nytt Tkinter vindu, og legge til alle funksjoner som vi har av knapper som vi trenger for å komme oss ut av grafen sitt vindu. Med tanke på at dette skal være på et touch-display, trenger vi en «Quit» knapp for å lukke den.

```
app.geometry("800x480+0+0")
```

```
def graph_window(database_val,label,yaxislab): #Definerer funksjone
    root = tk.Tk() # Setter opp tkinter
    root.wm_title("Embedding in Tk") #Navn på vinduet
    graf_name = tk.Label(root, text= label,   bg='green', fg='white',
    ↪  font=('helvetica', 15, 'bold')) #Label
    graf_name.pack(side=tk.TOP) #Plasserer med pack i vinduet
    f = Figure(figsize=(5,4), dpi=100) #Setter opp figuren
    a = f.add_subplot(111) # Lager plotting for grafen
    insert_val =database_val # Lagrer database verdien
    f.ylabel = 'test' # Label

    def animate(i):
        c1 = app.cursor
        c1.execute("SELECT time FROM data ORDER BY time DESC LIMIT 1")
        ↪  #Henter tiden
        now_time_string = c1.fetchone()[0]
        now_time = datetime.datetime.fromisoformat(now_time_string)
        then_time = now_time - datetime.timedelta(seconds=13) # Setter antall
        ↪  verdier X-aksen skal vise
        then_time_string = then_time.isoformat()
        b = (then_time_string, now_time_string)
        c = app.cursor
        c.execute(insert_val,b)
        fetch = c.fetchall()
        Xaxes = [x[-5:] for (x, y) in fetch]
        Yaxes = [y for (x, y) in fetch]
        pltXaxes = np.array(Xaxes) # Setter plot verid fo X-aksen
        pltYaxes = np.array(Yaxes) # Setter plot verid for Y-aksen
        a.clear() #Refresher plottet
        a.plot(pltXaxes,pltYaxes) # Plotter grafen
        a.set_ylabel(yaxislab) # Y-akse lable
        a.set_xlabel("Time")# X-akse lable
    canvas = FigureCanvasTkAgg(f, master=root)  # A tk.DrawingArea. # Setter
    ↪  området grafen skal tegnes i
    canvas.draw() #Viser grafen
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)
    def _quit(): #Exit knapp
        root.quit()
        root.destroy()
    button = tk.Button(master=root, text="Quit", command=_quit, bg='red',
    ↪  fg='white', font=('helvetica', 30, 'bold')) #Knapp
    button.pack(side=tk.BOTTOM)
    root.ani = animation.FuncAnimation(f,animate, interval=1000) #Starter
    ↪  animasjonen
    root.geometry("800x480+0+0") #Vindu størrelse og plassering
    root.attributes("-fullscreen", True)
    root.mainloop()
```

Kodeutdrag 20: Animerte graf funksjonen.

13.3.4. Eksport av sensordata

Eksporteringsfunksjonen har vi laget til å kunne sende sensordatafilene i et CSV format, slik at du kan åpne det i en rekke dokument programmer som eksempel Excel og du får en oversikt med hver data i sin egen rad med tilte for hvilket data som er i raden. Vi får dette til ved å importere CSV biblioteket i Python, det gir oss tilgang til å skrive ut data fra databasen. Vi gjør dette på to følgende måter:

1. Du kan velge ifra våre to lister, hvor du setter start tiden og slutt tiden du ønsker å eksportere ifra. Samt kan du oppdatere listen for å få de siste tidene. Vi så på måter å gjøre dette automatisk på, men fant ut at en slik funksjon alltid gikk i bakgrunnen som gjorde programmet tregt.

2. Den andre metoden for å få ut data på er å eksportere fra «Session», det vil kun ta data ifra tiden du startet programmet eller tiden du klikket på «Clear session data» som gjør at du kan starte på en ny session.

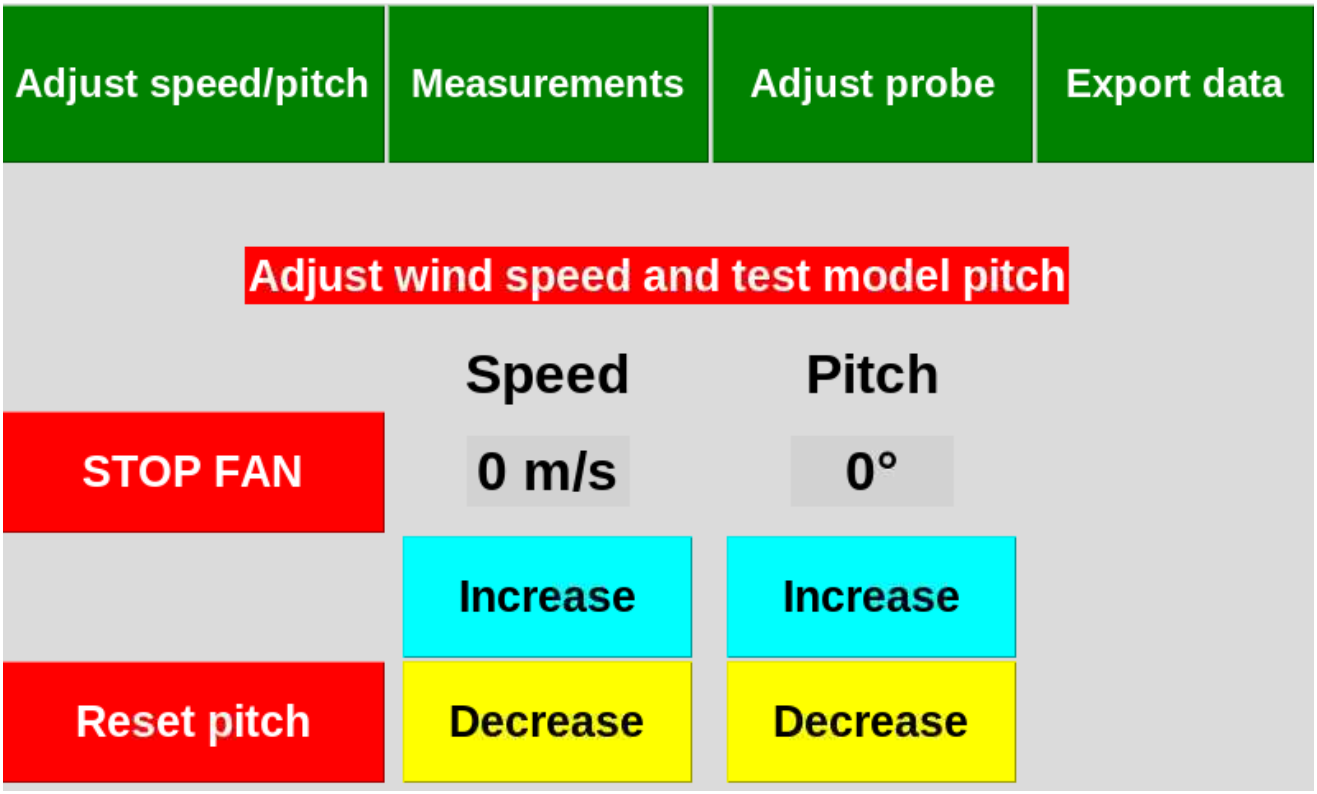
```
def update_times_list(): # Denne funksjonene henter inn daten fra databasen
↪  og oppdatere dropdown menyen
    c = controller.cursor
    c.execute('SELECT time FROM data') #Henter tid fra databasen
    cblist = c.fetchall()
    cb['values'] = cblist #Dropdown liste 1
    cb2['values'] = cblist#Dropdown liste 2
update_times_list()

def export_to_csv(startTime=cb.get(), endTime=cb2.get()): #Henter data fra
↪  valgt tid ifra dropdown menyen
    if (startTime>=endTime): #Start tid kan ikke være etter slutt tid
        tk.messagebox.showerror("Error", "Start time can not be less or equal
        ↪  to end time") #gir feil om det stemmer
    else: #Ellers eksporterer som vanlig
        export_file_path = f"/var/www/idm.com/public_html/{endTime}.csv"
        ↪  #Setter path filen blir lagret

        cursor = controller.cursor
        q = (startTime, endTime)
        cursor.execute("SELECT * FROM data WHERE time BETWEEN ? AND ?", q)
        ↪  #Henter data mellom start og slutt tid som er valgt

        with open(export_file_path, "w") as csv_file: #Eksporterer i
        ↪  filformatet CSV
            csv_writer = csv.writer(csv_file)
            csv_writer.writerow([i[0] for i in cursor.description])
            csv_writer.writerows(cursor)

def export_session(): #Eksporterer ifra tiden da du startet softwaren
    session_end_time = datetime.datetime.now().isoformat()[:19]
    export_to_csv(self.session_start_time, session_end_time)

def export_session_refresh():#Refresher session til å starte å samle data
↪  ifra denne blir klikket på
    self.session_start_time = datetime.datetime.now().isoformat()[:19]
```

Kodeutdrag 21: Eksportering funksjonen.

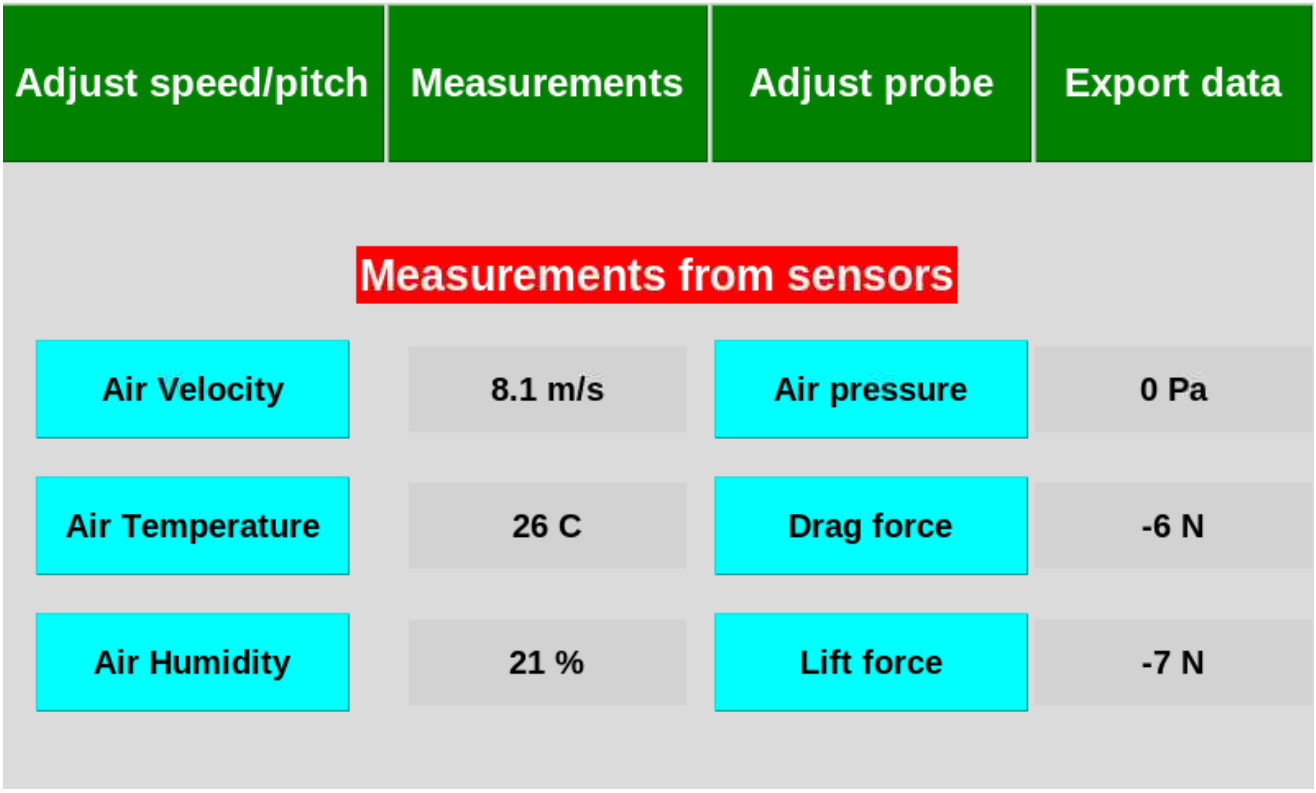
time	windspeed	temperature	humidity	pitch	airpressure	dragforce	liftforce
2020-05-08T17:08:05	32	23	22	2	4	-16	3
2020-05-08T17:08:06	5	23	23	1	5	-9	4
2020-05-08T17:08:07	4	23	24	3	1	-19	-7
2020-05-08T17:08:08	31	19	22	5	3	-16	5
2020-05-08T17:08:09	8	21	22	4	8	-5	-7
2020-05-08T17:08:10	7	20	22	-1	5	-2	6
2020-05-08T17:08:11	56	25	24	2	0	-3	-7
2020-05-08T17:08:12	56	26	22	3	4	2	-7

Tabell 13.1.: Eksempel hvordan eksportert data ser ut.

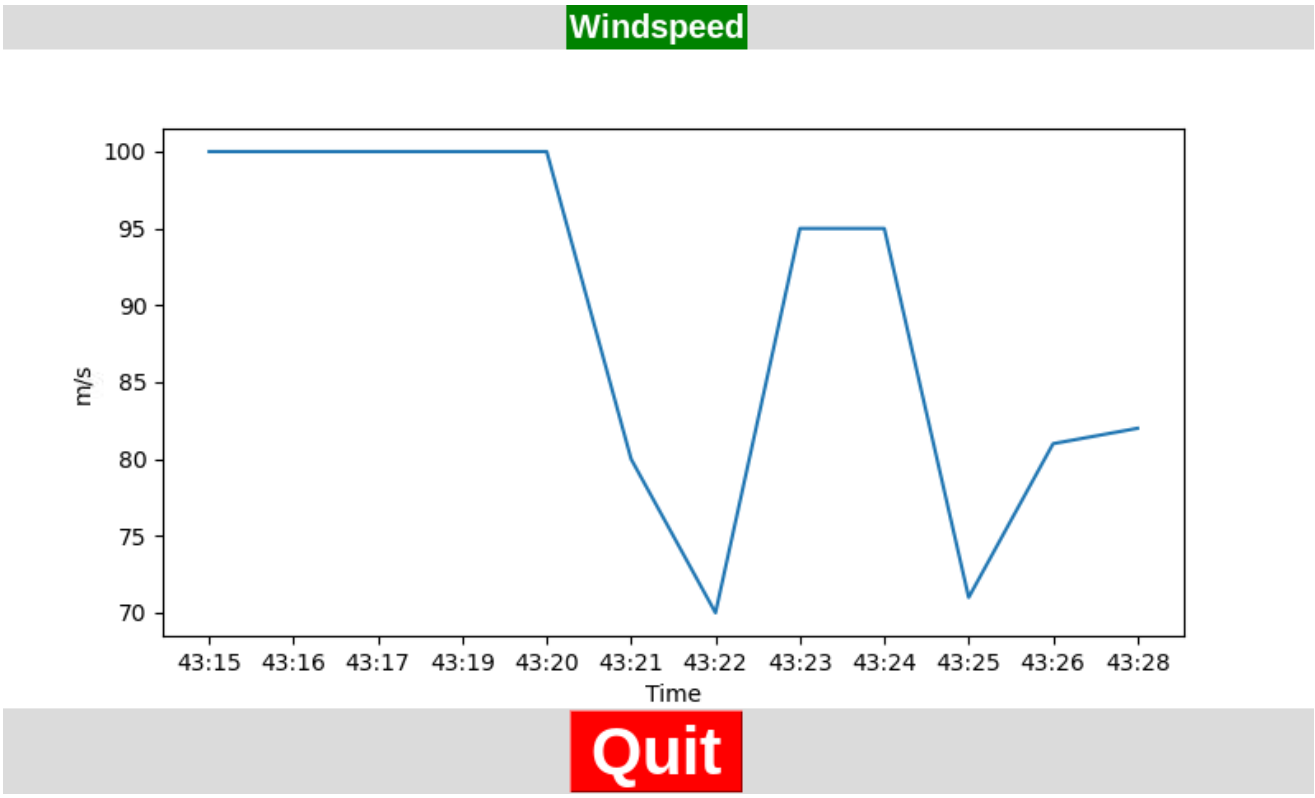
Figur 13.6.: GUI forside.



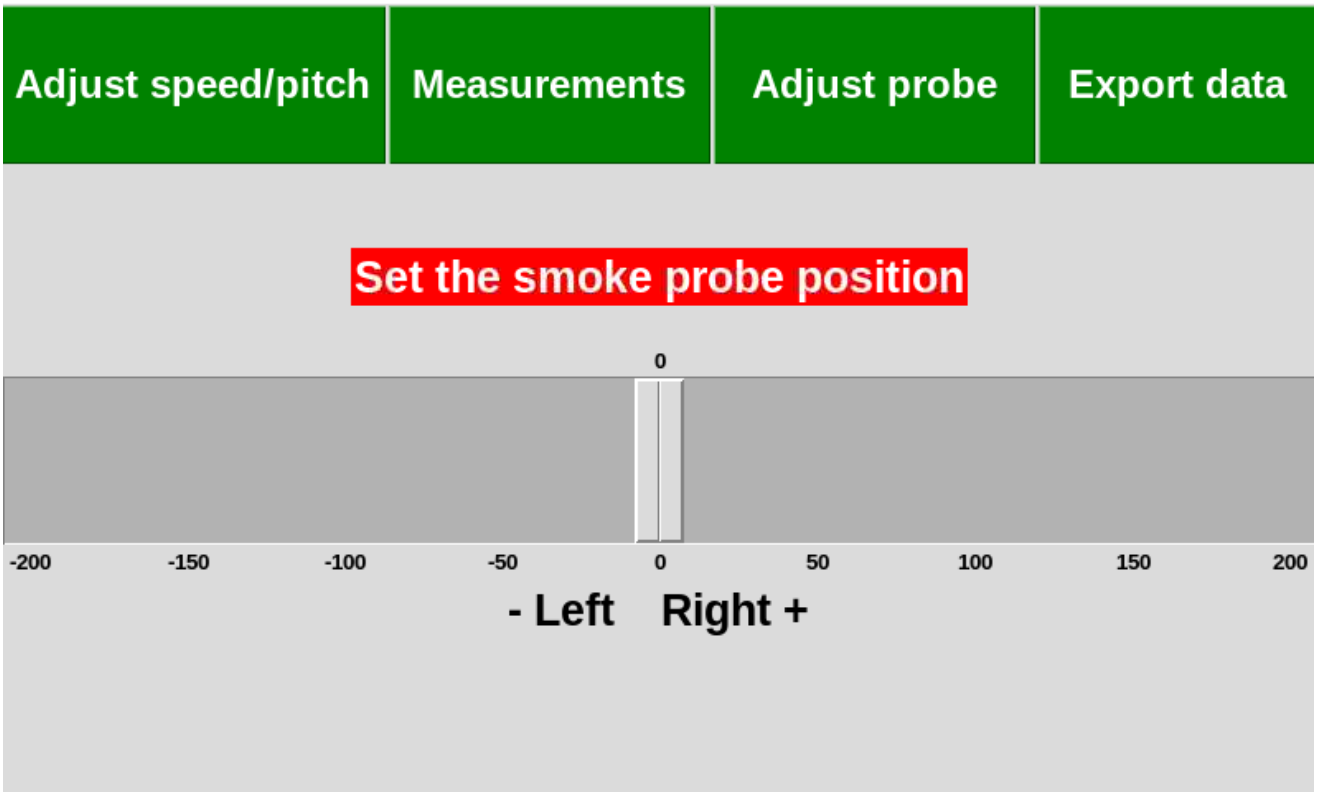
Figur 13.7.: Endre hastighet og pitch.



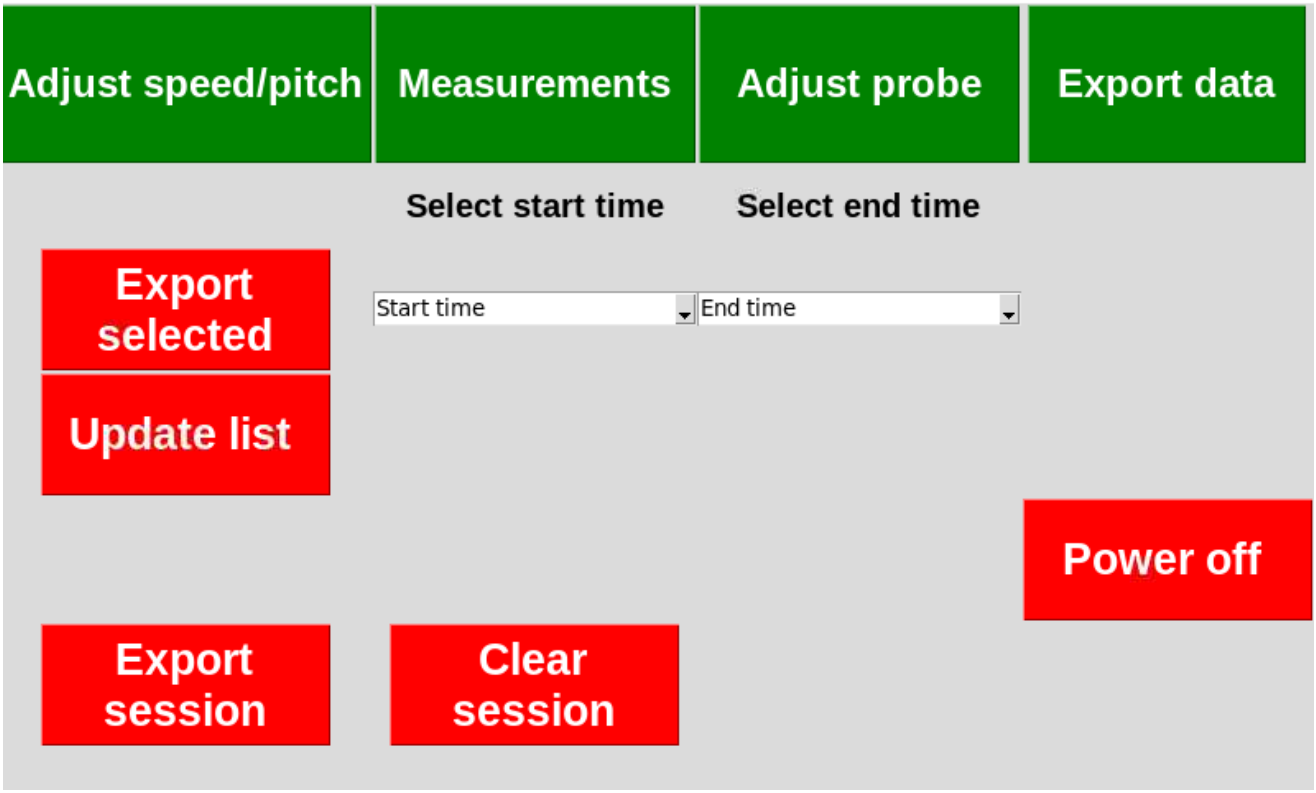
Figur 13.8.: Sensor målinger.



Figur 13.9.: Animert graf.



Figur 13.10.: Røykeprobe posisjon.



Figur 13.11.: Eksportering.

Del IV.

Verifisering, konklusjon, og videre arbeid

14. Oppfylte og ikke-oppfylte krav

Den følgende lista er en gjennomgang av kravene til prosjektet i henhold til om gruppa mener vi oppfylte eller ikke oppfylte kravene. En diskusjon om hvordan vi oppfylte kravene eller ikke følger i konklusjonen.

Tabell 14.1.: Oppfylte krav

Krav ID	**Krav**	**Oppfylt**
K.VT 02	Det skal være jevn (+/- 6%) vindhastighet over målesnittet (snittet i testkammeret der lufta møter testmodellen) i testkammeret	Delvis
K.VT 04	Lufthastigheten i testkammeret skal kunne settes av bruker	Ja
K.VT 05	Dimensjoner på testkammeret skal være minst 40cm x 40cm x 40cm (+/- 20%)	Ja
K.VT 06	Brukermanual og vedlikeholdsmanual skal lages for vindtunnelen	Delvis
K.VT 08	Vindtunnelen skal kunne flyttes av én person uten hjelpemidler.	Ja
K.VT 09	Vindtunnelen må ikke ha eksterne mål større enn 210 x 85 x 190 cm (LxBxH), og vekt større enn 800 kg, for å kunne transporteres til måleteknisk lab og i heisene på Krona	Ja
K.VT 09.03	Dersom maksverdiene for eksterne mål og vekt ikke kan overholdes, skal vindtunnelen kunne deles i moduler som overholder kravene individuelt	Ja
K.VT 10	Luftstrømmen i vindtunnelen skal kunne visualiseres ved hjelp av røyk(skjer i den laminære delen)	Nei
K.VT 10.03	Røykens utløp skal styres i et plan normalt på vindretningen	Nei
K.VT 11	Komponenter som trenger vedlikehold skal være lett tilgjengelige	Delvis

Krav ID	Krav	Oppfylt
K.VT 12	Testkammeret skal ikke slippe inn falsk luft gjennom sprekker og utette åpninger.	Ja
K.M 01	Lufthastigheten i testkammeret skal måles og stemme med innstilt lufthastighet (+/- 5%)	Ja
K.M 01.01	Lufthastighet skal måles i et tverrsnitt normalt til strømretning foran testmodellen for kartlegging av hastighetsfordeling	Nei
K.M 02	Lufttrykket i testkammeret skal måles	Delvis
K.M 03	Lufttemperaturen i testkammeret skal måles	Ja
K.M 04	Luftfuktigheten i testkammeret skal måles	Ja
K.M 05	Interne sensorer plassert før luftstrømmen treffer modellen skal ikke forårsake turbulens	Delvis
K.M 06	Målinger skal ikke påvirkes av vibrasjon fra vifta	Nei
K.M 06.01	Dersom vibrasjon ikke kan forhindres skal den måles	Nei
K.M 07	Pitchvinkel til testmodellen skal måles	Nei
K.M 08	Krefter på testmodellen skal måles i horisontal og vertikal retning	Delvis
K.DB 01	Måledata skal samles digitalt og kunne overføres til ekstern PC	Ja
K.DB 01.02	Måledata som overføres til ekstern PC skal være på et organisert format som kan importeres til et Excel-ark	Ja
K.DB 02	Måledata skal vises til bruker på skjerm	Ja
K.TM 01	Testkammeret skal kunne åpnes for å bytte testmodell	Ja
K.TM 02	Testmodellen skal kunne byttes	Delvis
K.TM 03	Innfestningen for testmodeller skal ikke kunne bøye seg, rotere, eller flytte seg under bruk	Delvis
K.TM 04	Innfestningen for testmodeller skal kunne justeres i høyde	Nei
K.TM 05	Pitchvinkel til testmodellen skal kunne styres av bruker	Delvis
K.HMS 01	Vindtunnelen skal ikke kunne startes når testkammeret er åpent	Delvis

Krav ID	Krav	Oppfylt
K.HMS 02	Roterende deler skal beskyttes slik de ikke kan skade fingre eller lemmer til bruker	Nei
K.HMS 03	Røyk må ikke utgjøre fare for eller være til bry for brukere eller andre i rommet	Nei
K.HMS 04	Støynivået fra vindtunnelen skal ikke overskride 80 dB	Delvis
K.HMS 05	Vindtunnelen skal kunne stoppes i et nødstilfelle	Delvis
K.HMS 06	Vindtunnelen skal være elektrisk isolert og/eller jordet	Ja
K.HMS 06.01	Elektriske kretser og koblinger skal tildekkes elles innkapsles	Delvis
K.HMS 07	Vindtunnelen skal ikke trekke mer strøm enn sikringskapasiteten.	Ja

15. Konklusjon

15.1. Innebygd datamaskin

Hovedmålet med den innebygde datamaskinen var å samle måledataene fra vindtunnelen og presentere disse for brukeren på en måte som gjorde det mulig å utføre videre analyse på dataene.

Sekundærmålet var å gi brukeren et grensesnitt for å kunne styre vindhastigheten og andre funksjoner, og å vise måledataene til brukeren.

Når det kommer til hovedmålet er det i prinsippet gjennomført. IDMen er i stand til å motta data, lagre dem, og presentere dem på et format som kan importeres direkte til Excel eller andre regnearkprogrammer. Dataene kan enkelt overføres til brukerens PC via et trådløst aksesspunkt og en webserver, som er mer praktisk enn å bruke kabler. På grunnlag av dette regner vi krav K.DB 01 og K.DB 01.02 som oppfylt.

Men, selv om IDMen er klargjort for å *behandle* data fra alle de planlagte sensorene, er det noen sensorer som det ikke er klart til å lese fra. Spesifikt trykksensoren og kraftsensorene for testmodellen er ikke implementert i Arduino-koden på grunn av tidspress. Krav K.M 02 og K.M 08 kan derfor ikke bli mer enn delvis oppfylt.

Brukergrensesnittet presenteres på en touch-skjerm som festes på vindtunnelen. Her vises måledataene fra vindtunnelen og kontrollknapper for å styre vifta og pitchvinkelen til testmodellen i testkammeret, samt kontrollene for å bestemme hvilke måledata som skal eksporteres.

Brukeren kan også se grafer som viser måledataene oppdatert i sanntid på skjermen. Dette oppfyller krav K.DB 02 og IDM-siden av K.VT 04.

15.2. Konklusjon mekanisk design

Fra begynnelsen var det meningen å kartlegge og eventuelt utbedre den eksisterende tunnelen, men pga Covid-19 og i etterkant av presentasjon 2 skiftet fokuset til en mer teoretisk oppgave.

Hovedmålet til maskin har vært å designe en komplett arkitektur for konsept 2 av vindtunnelen klar for integrering og fremstilling. Mye har blitt designet ferdig slik at det bare trengs å tilpasses og deretter bygges, men det mangler noe ferdig design og videre noe funksjonalitet. Det mangler også noe integrering mellom maskin og elektro noen steder, og mellom data og maskin.

15.2.1. Testkammer

Testkammeret later til å være designet bedre enn det var opprinnelig. Det er også mulig å bruke testkammeret som det er idag men det vil være vanskeligere å skulle bygge det om, med tanke på produksjon av luke.

15.2.2. Testmodell

Testmodellen later til å kunne brukes i tunnelen, men den må også da tilpasses å festes med skruer på testriggen som er designet her. I dag er den bare limt på en stang.

15.2.3. Innfesting

Innfestingen som vi har sett for oss er et forslag for hvordan man kan få holdt fast modellen og overført krefter gjennom testriggen. Den er ikke optimal men kan brukes som et utgangspunkt.

15.2.4. Endre pitch på modell

Maskin har lagt inn forenklet stepmotor i design under innfesting for testmodell og data har lagt inn mulighet i GUI for å endre pitch på modell, men elektro har ikke sensor eller programmering for å gjennomføre dette.

15.2.5. Kraftrigg

Kraftriggen kunne trengt en måte å støtte modellen bedre slik at man kan unngå at modellen som blir festet kan vri seg.

15.2.6. Visualisering

Å skulle visualisere foran modellen i tunnelen ligger utenfor hva vi har rukket å få til i dette prosjektet. Dette har også hengt sammen med at vi ikke har rukket å designe en løsning for scanning av hastigheten foran modellen sitt tverrsnitt.

15.2.7. Scanning hastighet foran testmodell

Fartsmålingen foran modellen rekker vi ikke å få gjennomført, og dette henger tett sammen med den tenkte visualiseringsmetoden. Siden vi ikke har rukket å designe en løsning for visualiseringen har vi da ikke en felles platform som kan gjøre begge deler.

15.2.8. Størrelse testkammer

Dette kravet har blitt oppfylt per oppdraggivers ønske, og det passer godt sammen med de beregningene som har blitt gjort for å komme fram til de dimensjonene som trengs for testkammeret.

15.2.9. Størrelseskrav

Per kravet om maksimal størrelse på vindtunnelen har dette blitt oppfyllt siden tunnelen er planlagt og designet for å kunne deles opp.

15.2.10. Mulighet bytte testmodell

Testmodellen kan byttes, men det krever også at man designer eventuelle testmodeller på en slik måte at man kan feste dem til innfestingen som bruker bolter.

15.2.11. Ikke kunne starte vindtunnel uten at luken er igjen

Denne funksjonen er delvis oppfylt. Fra maskin sin side har det blitt designet inn en bryter utenfor døren som kun er nær sine respektive motparter når døren til vindtunnelen er igjen. Men funksjonen som skal gjøre dette har ikke blitt jobbet med fra elektro sin side.

15.2.12. Krav om skjerming av roterende deler.

Har ikke blitt gjennomført siden vi ikke har en god modell av viften designet slik at den kan designes inn og skjermes.

15.2.13. Krav om støy

Kravet er møtt via verifisering i datablad der støynivået til viften er oppgitt som under 80 dB.

15.2.14. Krav om nødstoppbryter

Maskin har designet nødstoppbryter på begge brukssider av vindtunnelen, men de har ingen funksjon som de er koblet til.

Med bakgrunn i kravene som har blitt møtt og ikke møtt har vi ikke kommet i mål med arkitektur av vindtunnelen.

15.3. Konklusjon for elektriske delsystemer

Denne seksjonen tar for seg konklusjoner og oppsummering av hvert elektriske delsystem som gruppen har jobbet med, men også en konklusjon for elektro delen av prosjektet i sin helhet.

15.3.1. Konklusjon for styringssystem og måling av vindhastighet

Siden måling av vindhastighet og styringssystemet er knyttet så tett opp mot hverandre, så dekker vi begge delsystemer i denne underseksjonen og bedømmer det som ett og samme system som er knyttet til krav K.VT 04 og krav K.M 01 som begge er A-krav med høyest prioritet. For å oppsummere så har vi for dette delsystemet valgt ut alle de nødvendige komponentene som kreves for implementasjon, vi har begrunnet våre valg av komponenter og dokumentert dem veldig grundig. For vindhastighet sensor og mikrokontroller bruker vi eksisterende produkt, mens resten av maskinvaren og programvaren (bortsett fra software bibliotek) som kreves har vi designet selv.

DAC kretskortet som var en nødvendig ekstra komponent til styringssystemet har vi designet og simulert selv. Simulasjonen ga nøyaktige resultater som stemte overens med de kretsberegningene for knekkfrekvens og forsterkningssfaktor som vi fikk med de komponentverdiene vi valgte. Kretskortet vi designet var basert på dette grunnlaget. Vi kom aldri sålangt at vi fikk fysisk printet dette kortet, så vi fikk aldri testet om det ville fungert i virkeligheten. Grunnet tidsbegrensninger og det at vi var nødt å tilpasse oss til et nytt PCB software kan også ha gjort at det er noen viktige tester av designet vårt som vi har glemt å gjøre, så om vi hadde hatt bedre tid ville vi nok gått tilbake og testet dette designet mer nøye, at alle kobberspor har god nok avstand fra hverandre og at de har optimal størrelse samt andre tester for optimalisering

iforhold til elektromagnetisk kompatibilitet. Det kan være tidkrevende å få et kort ferdigprintet, derfor er det ekstremt viktig med god nok testing av det endelige designet før man fysisk printer det.

Vi simulerte også hele styringssystemet, men denne simulasjonen var ikke en nøyaktig representasjon med vår motor, men heller en mer generell simulering for å teste lukket sløyfe konseptet vårt. Her hadde vi noen problemer, for det første skulle vi gjerne hatt de eksakte motor-parameterne slik at vi kunne gjort en enda mer relevant simulering, for det andre var det også feil med selve simuleringen som gjorde at motoren ga ut veldig lave hastigheter selvom vi kjørte maks spenning inn, dette var noe vi dessverre ikke rakk å løse innen tidsfristen.

Å simulere et 3-fase AC system er mye mer komplekst, og ikke like rett fram som å simulere et DC system. All kunnskap om styringssystemer for 3-fase AC motor har vi etter beste evne innhentet oss på egenhånd. Men selv med unøyaktig simulering så koster det veldig lite å bestille de nødvendige komponentene, begynne med fysisk implementering og testing av styringssystemet. Det er mulig å få ønskede resultater ved å bare koble opp systemet og prøve seg fram med forskjellige parametere siden hele PID-regulatoren er software basert.

Siden mye av styringssystemet var software basert, så har vi også fått gjort unna endel klargjøring med tanke på fysisk implementasjon, vi har laget en enkel guide for hardware implementasjonen og skrevet alt av nødvendig kode. Her gjenstår det bare å bestille deler og sette igang med implementasjon, testing og feilsøking. For å avslutte vil vi konkludere med at vi syntes vi har gjort riktig valg av løsning, og at det er fult mulig for neste prosjektgruppe å fortsette å implementere denne løsningen, men vi skulle ønske vi hadde motor-parameterne tilgjengelig og at vi fikk til en simulering som virket som den skulle.

Validering og verifisering

Måling av vindhastigheten i vindtunnelen og styringssystemet er knyttet til krav K.M 01 som sier at lufthastigheten i testkammeret skal måles og stemme med innstilt lufthastighet +- 5 %. Siden oppgaven vår ble gjort om til en teoretisk oppgave så kom vi aldri så langt at vi fikk testet og validert dette kravet. Dette kravet burde egentlig hatt flere underkrav knyttet til seg. Først et underkrav der man fokuserer på sensoren for måling av vindhastighet alene der man lager et krav for hvor høy målefeil iforhold til den faktiske vindhastigheten er. Og utifra det kunne vi lagd et eget krav for styringssystemet basert på hvordan målefeilen fra vindhastighet sensoren vil videre påvirke styringssystemet for å finne ut hva som er realistisk å oppnå istedenfor å bare gi et tall. Noen ganger kan det også hende i et prosjekt at kunden har sine ønsker, da er det viktig å kunne forklare og bevise om kravet teknisk sett er mulig eller ikke.

Krav K.M 01 er et godt definert overordnet krav som bør inngå i styringssystemet, men vi burde også lagd underkrav for hvor høy overshoot, stigningstid og settlingtid vi ønsker for å knytte det opp mot kravet. Velger neste gruppe å beholde kravet som det er, så burde det ihvertfall legges til forskjellige underkrav for hver komponent som er involvert i hele systemet. For hvis man bare har dette kravet, og systemet viser seg å ikke oppfylle dette kravet, så er det enklere å feilsøke hvis man har flere underkrav å forholde seg til. Når det kommer til validering, så vil vi si at vårt valg av teknisk løsning for både vindhastighetsmålingen og styringssystemet var de beste alternativene vi kunne valgt utifra de begrensningene som den eksisterende vindtunnelen ga oss.

Vi er på riktig spor når det kommer til å oppfylle krav K.M 01 og krav K.VT 04 iforhold til validering, men vi skulle gjerne ønsket vi hadde klart å få tak i de nødvendige motor-parameterne for å gjøre en mer nøyaktig simulering av styringssystemet for da kunne vi gitt en bedre validering av det. Siden vi ikke rakk å fysisk implementere styringssystemet, så får vi ikke testet og validert dette kravet. Fra et verifiseringsstandpunkt mener vi at at vi har gjort nok arbeid på

planlegging, design og implementasjon til å konkludere med at krav K.M 01 og krav K.VT 04 er oppfylt.

15.3.2. Konklusjon for delsystem for måling av lift og drag

Delsystemet for måling av lift og drag er knyttet til krav K.M 08 som er et A-krav med høyest prioritet. For dette delsystemet har vi også valgt en løsning som vi mener er optimal for dette prosjektet. Vi har valgt ut alle de nødvendige hardware komponentene som trengs for å implementere delsystemet og vi har tatt med alle software bibliotek som kreves for å teste lastcellene. Lastceller kommer i mange forskjellige former med forskjellig kapasitet og egenskaper, det kan selvfølgelig drøftes om akkurat den lastcelle typen som vi valgte er den mest optimale typen. Til den drøftingen vil vi legge ved at vi gjorde grundig research på lastceller og kikket på andre vindtunnel prosjekter og der oppdaget vi at single-point lastceller med 5-10 kg kapasitet var mest standard brukt i vindtunnel prosjektene som vi kikket på. Jeg tenker utvalg av lastcelle kapasitet henger sammen med hvor kraftig vifte man bruker, her burde vi kanskje vært flinkere på å studere forholdet mellom disse to variablene for å ha et sterkere grunnlag for hvorfor vi gikk for akkurat 10 kg lastceller. Velger man for høy eller lav kapasitet, så vil vi få dårligere sensitivitet på sensorene som vil gi mindre nøyaktige målinger.

Til dette delsystemet trengte vi også en ekstra komponent iform av en lastcelleforsterker. Vi er skråsikre på at denne komponenten var absolutt nødvendig da det er en veldig universal komponent som brukes i så og si alle systemer som sender kraftmålinger til en Arduino mikrokontroller, vi så at de brukte HX711 kort i de andre vindtunnel prosjektene vi kikket på også. Den har tilkoblingsmuligheter for to lastceller som var midt i blinken for vårt prosjekt. Siden vi gikk over til et teoretisk prosjekt, så bestemte vi oss for å gi oss selv litt ekstra arbeid å vise til ved å designe dette kortet fra bunn av. Det hadde strengt tatt ikke vært nødvendig og ville bare vært bortkasta tid hvis prosjektet fortsatt var et praktisk prosjekt. Konklusjonen for design prosessen av HX711 kortet er identisk med DAC kortet som vi skrev i forrige seksjon,

så vi kommer ikke til å repetere oss selv. Eneste forskjellen er at vi ikke fikk simulert kretsen vi designet for dette kortet da vi hadde problemer med å finne ekstern modell for selve HX711 IC komponenten for simuleringsverktøyet LTspice. Man kan simulere kretser i KiCad også, men heller ikke der klarte vi å finne noen ekstern simulerbar komponent. Men som sagt så ville både design av dette kortet og denne simulasjonen vært overflødig om prosjektet hadde forblitt et praktisk prosjekt.

Vi fikk også gjort unna mye i implementasjonsfasen av dette delsystemet da konverteringen av måledata er softwarebasert, det er programvaren som er lastet inn på mikrokontrolleren som gjør disse beregningene. Vi fant eksisterende software-bibliotek som brukes for å kalibrere lastcellene og konvertere måledataene om til kilo. Siden vi ikke fikk bestilt lastcellene så fikk vi heller ikke muligheten til å teste disse software-bibliotekene og da så vi heller ingen grunn til å bruke mye tid på å prøve å skrive hovedprogrammet som skal konvertere kilo om til Newton krefter for måling av lift og drag. Men vi fant en kildekode fra et annet prosjekt som vi har referert til som inneholder kode for denne konverteringen [33].

For å avslutte vil vi bare konkludere med at vi mener vi har absolutt gjort riktig valg av løsning for dette delsystemet. Lastcelle typen vi valgte har en error som er veldig lav på 0.05 %, så ved å implementere disse lastcellene så ville vi fått veldig nøyaktige målinger hvis software delen av systemet blir skrevet riktig. Det eneste som bør revurderes er om man burde gå for lastceller med lavere kapasitet for å øke måle-sensitiviteten.

Validering og verifisering

Krav K.M 08 er riktig definert, men vi syntes i etterkant at vi burde lagt til i kravet hvor høy målefeil vi tolererer for å ha noe konkret å måle opp mot kravet, for å enklere bedømme kvaliteten på delsystemet. Siden vi ikke fikk bestilt lastcellene så fikk vi ikke testet og validert kravet, men utifra et verifiseringsstandpunkt så vil vi påstå at kravet er delvis oppfylt både

fra den datatekniske, elektrotekniske og maskintekniske siden, disse lastcellene skal jo ha et grensesnitt med festestrukturen til testmodellen, dette er noe vi ikke har fått tid til å dokumentere godt nok hvordan denne samhandlingen mellom lastceller og festestruktur skal foregå. Det samme gjelder kommunikasjon mellom mikrokontroller og brukergrensesnitt, vi har ikke fått skrevet noe hovedprogram som konverterer fra kilo til Newton som kan leses av på brukerdisplayet. Vi la ved et eksempel fra et annet vindtunnelprosjekt som inkluderer konvertering av krefter, men vi har ikke fått tid å tilpasse det til vårt system.

For å konkludere så mener vi at vi absolutt er på riktig spor her også når det gjelder å oppfylle dette kravet for neste gruppe i framtiden. Det gjenstår litt arbeid med interfacingen med festestrukturen for testmodellen og høynivå kommunikasjonen med brukergrensesnitt samt skrive hovedprogrammet så bør man være i mål med å oppfylle dette kravet. Den eneste endringen som bør revurdes på vårt eksisterende design er å se på andre typer lastceller og gjøre en sammenligning.

15.3.3. Konklusjon for delsystem for måling av pitchvinkel

Dette delsystemet er knyttet til krav K.M 07 som er et B-krav. Dette kravet gikk litt i glemmeboka under Corona-distraksjonen der prosjektet vårt ble endret fra praktisk til teoretisk med mer fokus på design og simulering. Derfor fokuserte vi mest på å gjøre en god jobb på A-kravene. Kan dermed kort konkludere med at krav K.M 07 er ikke oppfylt.

15.3.4. Helhetlig elektroteknisk konklusjon

Hvis vi ser på begge de elektrotekniske delsystemene vi har snakket om i de to forrige seksjonene, så kan vi konkludere med at vi er fornøyd med å oppfylle de viktigste A-kravene for den elektrotekniske delen av prosjektet fra et verifiseringsstandpunkt. Hvis vi tenker tilbake så

burde dette prosjektet egentlig hatt minst to gruppemedlemmer med elektro bakgrunn da den elektrotekniske delen av vindtunnelen er veldig omfattende med mange komponenter og del-systemer som måtte designes, simuleres og implementeres. Det gjenstår også fremdeles endel elektrotekniske krav igjen, så hadde vi vært flere med elektrobakgrunn så kunne vi kommet litt lenger. Vi hadde nok hatt tid til å jobbet mer med krav K.M 07 hvis vi hadde droppet å designe egne kretskort, men vi prioriterte heller å gjøre godt kvalitetsarbeid på de viktigste A-kravene framfor å prøve oppfylle flest krav som mulig. Men vi syntes samtidig det var synd at vi ikke rakk å se så mye på krav K.M 07, da dette virket som en oppgave som kunne vært interessant å utvikle.

16. Videre arbeid

16.1. Maskin

En god anbefaling til videre grupper som skal jobbe med design i SolidWorks er å bruke funksjonen som heter ”Pack and Go” framfor å lagre nye assemblies eller deler med funskjonen ”Save As”. Jeg har desverre brukt ”Save As” funksjonen siden jeg tenkte at Solidworks da sikkert oppførte seg som om jeg jobbet i Word. Dette har ikke vært tilfelle og dette har også ført til at jeg måtte kjøre noen raske simuleringer nå på slutten av prosjektet slik at jeg kunne illustrere resultatene jeg har fått.

En funksjon som ikke har blitt møtt som følge av at vi har gått tom for tid er visualisering. Det var tenkt i begynnelsen å ha en løsning med kombinert fartsmåling og visualisering med røyk på en platform foran modellen, men dette har vist seg å ikke være veldig gjennomførbart siden vi har gått tom for tid. Det er også utfordringen med å skulle ha noe som kan beveges, opp/ned og høyre/venstre foran modellen i et såpass begrenset testkammer.

En funksjon som var ønsket men ikke like viktig som visualisering var muligheten til å heve og senke modellen i testkammeret slik at man kan få en best mulig plassering i forhold til luftstrømmen.

Det trengs åpenbart for de som skal ta over denne oppgaven at de tar å bruker beregninger for å verifisere om de estimatene vi har fått fra simuleringen stemmer.

Innfestingen for modellen burde hatt et spor som passer i stangen i bare en retning slik at man ikke kan montere på festeplaten i feil retning. Dette vil da sørge for at man bare kan montere dette sammen en riktig vei, og ikke kan feilmontere dette.

Om mulig for de som skal jobbe videre med denne vindtunnelen bør dere ha en mulighet for å kunne bruke en annen vifte, eller ihvertfall undersøke grundigere hvor mye verdi det kan gi til arbeidsgiver om man bruker en annerledes vifte. På den måten kan man også få verifisert hvor godt den viften som brukes idag passer.

Med tanke på FEM analyse bør man gjøre beregninger for hånd eller med excel slik at man kan finne gode estimater for kreftene som man skal utsette testmodeller og kraftrigg for.

Kraftriggen kan trenge et omdesign siden det endelige designet ikke har støtte for modellen og innfestingen annet enn direkte på sensorene for lift og drag. Eventuelt kan det hende at det som trengs bare er sensorer som kan måle større krefter slik at man kan fortsette å hvile infestingen på sensoren. Dette vil fortsatt ikke hindre mulig vridning av innfesting og testmodell.

Det bør sees nærmere på hvordan man kan simulere viften som er i vindtunnelen i dag med tanke på simulering og for å få data som trengs i beregninger videre for tunnelen.

Med tanke på diffuser kan det være en ide å vurdere en diffuser som begynner med å være firkantet og går over til en rund profil sånn at den passer best mulig til viften som skal brukes.

Med tanke på rammen som kan brukes til å holde diffuseren kan det være en ide å ordne med en rund del som står i vinkel slik at man kan bedre holde diffuseren i ro. Som rammedesignet er idag så står diffuseren rett oppå en endekant på en avrundet firkantstålprofil.

Med tanke på CFD kan det være en ide å vurdere å gå til studentkjøp av Simscape sin software som heter Cadmium, slik at man kan få et perspektiv til på simulering av flow gjennom vindtunnelen. Dette ble ikke gjort i dette prosjektet fordi det ble lite tid igjen og det ble betraktet som dyrt å gå til anskaffelse av den programvaren som krever et årlig abonnement.

De som fullfører prosjektet må lage bruks- og vedlikeholdsmanual for maskindelen av prosjektet, grunnet at tunnelen ikke er ferdig designet.

De som tar over må også lage eksperimenter som kan gjøres i sammenheng med fluidmekanikk. Dette kan også kreve nye modeller som har blitt designet for forsøk.

Risiko er noe som bør jobbes med jevnt gjennom hele prosessen slik at man har en klar oversikt over risikoene i prosjektet, og det kan ære en god ide å vurdere usannsynlige hendelser. Men man bør også begrense seg til ting man kan gjøre noe med. Eksempelvis kan vi ikke gjøre noe med risikoen for at en meteoritt lander i Kongsberg, men vi kan la være å teste vindtunnelen utendørs slik at det ville ha mindre sjanse til å påvirke oss.

Testing burde også ha blitt jobbet mer jevnt med slik at vi kunne bedre verifisert at vi har møtt krav. Det var vanskelig å jobbe med testing av systemet i virkeligheten, men på data hadde de Raspberry Pi og kunne dermed teste en del ting. Og siden det var vanskelig å få testet noe fysisk har det også vært vanskelig å få testet integrering av delsystemer. Dvs integrering av systemer slik at man kan få satt sammen delsystemene våres slik at vi kan ende opp med en funksjonell arkitektur, og til slutt en funksjonell vindtunnel.

En form for testing til som burde vært innført ville vært i form av beregninger slik at det er lettere å verifisere tenkt design mot krav. Dette ville også gjort det lettere å skulle verifisere grad av gyldighet med tanke på maskinsimuleringer.

Festestenger burde kanskje kombineres med muttere oppe ved modell slik at man lettere kan feste tynnere modeller.

De neste som tar over denne oppgaven bør gjennomgå simuleringene som har blitt kjørt og tilpasse variablene.

Trenger funksjon for kalibrering av kraftsensorer med tanke på å måle hvor mye krefter som påvirker av skifting av pitch for hver grad. Må også kunne ha funksjon for å kunne kalibrere etter modellen har blitt festet inn slik at man kan kalibrere for disse kreftene uten å ha på vind.

Må også kalibrere for eventuell drag og lift som systemet kan få ved bruk av innfesting og stenger på forhånd slik at disse kreftene kan sees bort ifra.

16.2. Data

16.2.1. Innebygd datamaskin

IDMen er kanskje den mest fullførte komponenten til prosjektet, men det betyr ikke at den er fullstendig.

I Arduinokoden mangler implementeringen av lastcelleforsterkerne og lufttrykkmålingen. Det bør innføres bedre feilsjekking av serieportmeldingene på Raspberry Pi-siden; foreløpig blir feil ignorert til feilen slutter, som kan gi problemer når batteriet til sanntidsklokka dør.

I tillegg fikk vi aldri tid til å designe systemet som skulle slå av Raspberry Pien automatisk og trygt når strømmen gikk.

16.2.2. Brukergrensesnitt

Som nevnt har vi ikke fått koblet dette opp mot vindtunnelen og dermed mangler vi noen funksjoner som skulle vært med om vi hadde hatt et fysisk sluttprodukt. Vi har lagt opp koden slik at det skal kunne enkelt bli implementert for den neste gruppen som eventuelt tar over dette prosjektet og videre bygger på arbeidet som gruppen har gjort. Det som brukergrensesnittet ikke kan gjør nå idag, som resultat av at vi ikke koblet dette opp mot vindtunnelen er:

- Justere Posisjon på modellen.
- Justere Posisjon på røykprobe.
- Måle lift fra sensor.
- Måle drag fra sensor.
- Måle lufttrykk fra sensor.

Dette gjenstår å bli koblet opp. Koden er satt opp for å ta i mot de forskjellige sensor verdiene, lagre og vise dem på brukergrensesnittet. Med tanke på at en stor del av softwaren er satt opp, så antar vi at det skal holde med én fra data under neste prosjekt. Vi tror det ikke er tilstrekkelig arbeid med to på data i neste gruppe med arbeidmengden som står igjen.

16.3. Elektro

Først og fremst burde neste prosjektgruppe vurdere å ha minst to gruppemedlemmer med elektrobakgrunn med tanke på hvor omfattende den elektrotekniske delen av dette prosjektet er. Veien videre vil involvere mye fysisk implementasjon, testing og verifisering og da er det greit å ha et ekstra par med øyne, men også for å kunne kryss-sjekke hverandres tekniske arbeid

for kvalitetssikring. Vi ser for oss at for neste prosjektgruppe vil prosjektet gå tilbake til sin ordinære plan, nemlig praktisk oppgave. Da ville jeg anbefalt elektro folkene å bestille ferdig kort for HX711 lastcelle forsterker kortet da dette er et velkjent og godt dokumentert produkt som gjør den jobben den skal uten at man trenger å bekymre seg så mye for feil og mangler og det vil bli mer tid å fokusere på de andre kravene. DAC kortet kan også fåes tak i ferdig løsning, så her er det litt opp til gruppen selv å gjøre en vurdering, man får gjerne litt bedre vurdering hvis man har endel egne design å vise til også.

Når det kommer til styringssystemet så var den simuleringen vi gjorde såpass generell, at vi anbefaler neste gruppe på det sterkeste å prøve å få tak i de nødvendige motor-parameterne for den motoren som står i vindtunnelen som kreves i Simulink simuleringen for å få en mer nøyaktig representasjon av det virkelige styringssystemet. Prøv å mas mer på produsenten av motoren. Vår simulering hadde også noen feil som nevnt tidligere i denne rapporten som vi også anbefaler neste gruppe å forsøke å løse. Men om neste gruppe ikke klarer å få til en god simulering, så kan man fremdeles gå videre til den fysiske implementasjonen og prøve seg fram med forskjellige PID parametere til kravet er oppfylt.

Når det kommer til delsystemet for måling lift og drag, så anbefaler vi å kjøre på videre med vår tekniske løsning, men ta gjerne å dobbelsjekk om det kan lønne seg å gå for lastceller med lavere kapasitet. Som nevnt tidligere i rapporten så mangler hovedprogrammet for konvertering av kilo til Newton. Vi la med en kode vi fant fra et annet vindtunnelprosjekt, så ta gjerne og bruk det som utgangspunkt.

Krav K.M 07 rakk vi heller ikke begynne på, her er det mye som må gjøres, sammen med alle de andre kravene vi ikke rakk å begynne på eller fullføre.

Ellers anbefaler vi dere å ha høyt fokus på hvordan alle de datatekniske, elektrotekniske og maskintekniske delsystemene skal ha grensesnitt med hverandre for effektiv samhandling.

Del V.

Vedlegg

A. Eksterne filer

Vedlegg og andre eksterne filer kan hentes på Google Drive:

`https://drive.google.com/open?id=1sBl08vhm5lLqath1hiACnGUhvPQM_ofc`

B. Kildekode

Kildekoden er også tilgjengelig på GitHub:

`https://github.com/VKing6/Bachelor_USN2020-09_IDM`

B.1. GUI.py

```
#!/usr/bin/env python3
#
# Author: Kristian Auestad & Steffen Barskrind
#

import matplotlib
matplotlib.use('Agg')
import tkinter as tk
from tkinter import filedialog
import serial
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
↪  NavigationToolbar2Tk)
from matplotlib.figure import Figure
from tkinter import ttk
import sqlite3
import os, sys
import csv
import datetime
import threading
import time
sys.path.append("/home/pi/Projects/IDM/GUI/")
import dataobject
import idmserial
```

```
from PIL import Image,ImageTk


########## PAGE FUNCTION ##########

class IDMGUI(tk.Tk):

    def __init__(self, *args, **kwargs):

        tk.Tk.__init__(self, *args, **kwargs)

        container = tk.Frame(self)

        container.pack(side="top", fill="both", expand = True)

        container.grid_rowconfigure(0, weight=1) #gjør at alt expander av
        ↪  seg selv med pack
        container.grid_columnconfigure(0, weight=1)


        #  Make a new database file for each month
        db_month = datetime.date.today().isoformat()[:-3]
        db_folder = f"/usr/db/idm/"
        #  Don't store more than 6 months of data. Delete the oldest
        ↪  database file when it turns over.
        dbs = [f for f in os.listdir(db_folder) if os.path.isfile(f) and
        ↪  f[-3:] == ".db"]
        if len(dbs) > 5:
            f = dbs.pop(0)
            if f != f"data_{db_month}.db":
                os.remove(dbs[0])
        #  Connect to database
        self.database = sqlite3.connect(db_folder + f"data_{db_month}.db")
        self.cursor = self.database.cursor()
        #  Check if data table exists in database and create it if not
        self.cursor.execute("""CREATE TABLE IF NOT EXISTS data
                    (time date, windspeed int, temperature int, humidity
↪  int, pitch int,
                        airpressure int, dragforce int, liftforce int)""")
        self.database.commit()

        #  Initialize communication thread
        self.stop_receiver_event = threading.Event()
        self.sensor_data = dataobject.DataObject()
        self.comm = idmserial.SerialCommunicator(self.sensor_data,
        ↪  self.stop_receiver_event)
```

```
67
68          #  Start database write loop
69          self._afterjob = self.after(2000, self.amend_database)
70
71
72          self.frames = {}
73
74          for F in (StartPage, PageOne, PageTwo, PageThree,PageFour): #antall
            ↪  sider som skal lages i programmet
75
76              frame = F(container, self)
77
78              self.frames[F] = frame
79
80              frame.grid_rowconfigure(0, weight=0) #gjør at alt expander av
                ↪  seg selv grid
81              frame.grid_columnconfigure(0, weight=1)
82
83              frame.grid(row=0, column=0, sticky="nsew")
84
85          self.show_frame(StartPage) # viser første side
86          self.title("IDM")
87
88
89      def show_frame(self, cont):
90
91          frame = self.frames[cont]
92          frame.tkraise()
93
94
95      def amend_database(self):
96          """
97          Copy the sensor data from the data object to the sqlite3 database
98          """
99          c = self.cursor
100         d = self.sensor_data.get_data()
101         if d["time"].year > 2000 and d["time"].year < 2100:  # Don't store
            ↪  bad/debug values
102             c.execute("INSERT INTO data VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
103                     (d["timestring"], d["windspeed"], d["temperature"],
                        ↪  d["humidity"],
104                     d["pitch"], d["airpressure"], d["dragforce"],
                        ↪  d["liftforce"]))
105             self.database.commit()
106         self._afterjob = self.after(1000, self.amend_database)
107
```

```
108
109     def power_shutdown(self):
110         """
111         Stop the IDM program and Shut down the IDM
112         """
113         self.comm.close_serial()
114         self.after_cancel(self._afterjob)
115         os.system("sudo shutdown now")
116
117
118  ##########  Start page ##########
119
120 class StartPage(tk.Frame):
121
122     def __init__(self, parent, controller):
123         tk.Frame.__init__(self,parent)
124
125         SpeedAndPitch = tk.Button(self, text="Adjust speed/pitch",height =
            ↪  3, width = 18,command=lambda: controller.show_frame(PageOne),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
126         SpeedAndPitch.grid(row = 0 , column = 0)
127
128         Measurments = tk.Button(self, text="Measurements",height = 3, width
            ↪  = 13, command=lambda: controller.show_frame(PageTwo),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
129         Measurments.grid(row = 0 , column = 1)
130
131         probe = tk.Button(self, text="Adjust probe",height = 3, width =
            ↪  13,command=lambda: controller.show_frame(PageThree), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
132         probe.grid(row = 0, column = 2)
133
134         export = tk.Button(self, text="Export data",height = 3, width =
            ↪  11,command=lambda: controller.show_frame(PageFour), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
135         export.grid(row = 0, column = 3)
136
137         labelsp33 = tk.Label(self, text="",  font=('helvetica', 30, 'bold'))
138         labelsp33.grid(row = 1 , column = 1)
139
140         labelsp3 = tk.Label(self, text="                    ",
            ↪  font=('helvetica', 30, 'bold'))
141         labelsp3.grid(row = 1 , column = 0)
142
143         labelsp4 = tk.Label(self, text="Welcome to the wind tunnel software
            ↪  ", bg='red', fg='white', font=('helvetica', 30, 'bold'))
```

```
144         labelsp4.grid(row = 2 , column = 0, columnspan = 4)
145
146         Kristian = tk.Label(self, text="Kristian Auestad",
            ↪  font=('helvetica', 25, 'bold'))
147         Kristian.grid(row = 3 , column = 0, columnspan = 4)
148
149         steffen = tk.Label(self, text="Steffen Barskrind",
            ↪  font=('helvetica', 25, 'bold'))
150         steffen.grid(row = 4 , column = 0, columnspan = 4)
151
152         kristoffer = tk.Label(self, text="Kristoffer Andersen ",
            ↪  font=('helvetica', 25, 'bold'))
153         kristoffer.grid(row = 5 , column = 0, columnspan = 4)
154
155         marisu = tk.Label(self, text="Marius Balsvik ",  font=('helvetica',
            ↪  25, 'bold'))
156         marisu.grid(row = 6 , column = 0, columnspan = 4)
157
158
159         Havard = tk.Label(self, text="Håvard Gaska ",  font=('helvetica',
            ↪  25, 'bold'))
160         Havard.grid(row = 7, column = 0, columnspan = 4)
161
162
163         joachim = tk.Label(self, text="Joachim Haug ",  font=('helvetica',
            ↪  25, 'bold'))
164         joachim.grid(row = 8, column = 0, columnspan = 4)
165
166  ##########  PAGE 1 Adjust speed and pitch ##########
167
168 class PageOne(tk.Frame):
169
170     def __init__(self, parent, controller):
171         tk.Frame.__init__(self, parent)
172
173         self.transmitter = controller.comm
174
175         counterFan = tk.IntVar()
176         windspeed_display = tk.StringVar()
177         windspeed_display.set(str(counterFan.get()) + " m/s")
178         maxFan, minFan = 100, 0
179         def increase_fan():
180             current = counterFan.get()
181             if current < maxFan:
182                 new = current + 1
183                 self.transmitter.transmit(f"W{new}X")
```

```
                counterFan.set(new)
                windspeed_display.set(str(counterFan.get()) + " m/s")

        def decrease_fan():
            current = counterFan.get()
            if current > minFan:
                new = current - 1
                self.transmitter.transmit(f"W{new}X")
                counterFan.set(new)
                windspeed_display.set(str(counterFan.get()) + " m/s")

        def stop_fan():
            counterFan.set(0)
            self.transmitter.transmit(f"W0X")
            windspeed_display.set(str(counterFan.get()) + " m/s")


        counterPitch = tk.IntVar()
        pitch_display = tk.StringVar()
        pitch_display.set(str(counterPitch.get()) + u"\N{DEGREE SIGN}")
        maxPitch, minPitch = 10, -10
        def increase_pitch():
            current = counterPitch.get()
            if current < maxPitch:
                new = current + 1
                self.transmitter.transmit(f"P{new}X")
                counterPitch.set(new)
                pitch_display.set(str(counterPitch.get()) + u"\N{DEGREE
                ↪  SIGN}")

        def decrease_pitch():
            current = counterPitch.get()
            if current > minPitch:
                new = current - 1
                self.transmitter.transmit(f"P{new}X")
                counterPitch.set(new)
                pitch_display.set(str(counterPitch.get()) + u"\N{DEGREE
                ↪  SIGN}")

        def reset_pitch():
            counterPitch.set(0)
            self.transmitter.transmit(f"P0X")
            pitch_display.set(str(counterPitch.get()) + u"\N{DEGREE SIGN}")
```

```
228         SpeedAndPitch = tk.Button(self, text="Adjust speed/pitch",height =
            ↪  3, width = 18,command=lambda: controller.show_frame(PageOne),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
229         SpeedAndPitch.grid(row = 0 , column = 0)
230
231         Measurments = tk.Button(self, text="Measurements",height = 3, width
            ↪  = 13, command=lambda: controller.show_frame(PageTwo),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
232         Measurments.grid(row = 0 , column = 1)
233
234         probe = tk.Button(self, text="Adjust probe",height = 3, width =
            ↪  13,command=lambda: controller.show_frame(PageThree), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
235         probe.grid(row = 0, column = 2)
236
237         export = tk.Button(self, text="Export data",height = 3, width =
            ↪  11,command=lambda: controller.show_frame(PageFour), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
238         export.grid(row = 0, column = 3)
239
240
241         labelsp33 = tk.Label(self, text="",  font=('helvetica', 30, 'bold'))
242         labelsp33.grid(row = 1 , column = 1)
243
244
245         labeltitle2 = tk.Label(self, text="Adjust wind speed and test model
            ↪  pitch",   bg='red', fg='white', font=('helvetica', 20, 'bold'))
246         labeltitle2.grid(row = 2 , column =0, columnspan = 5)
247
248         spacer3 = tk.Label(self, text="")
249         spacer3.grid(row = 3 , column = 0)
250
251
252         labelSpeed = tk.Label(self, text=" Speed ",height = 1,
            ↪  font=('helvetica', 25, 'bold'))
253         labelSpeed.grid(row = 4 , column =1)
254
255
256         labelpitch = tk.Label(self, text="  Pitch  ",height = 1,
            ↪  font=('helvetica', 25, 'bold'))
257         labelpitch.grid(row = 4 , column =2)
258
259
260         labelspeedinc = tk.Label(self, textvariable = windspeed_display, bg
            ↪  = "lightgrey", width = 5, font=('helvetica', 25, 'bold'))
261         labelspeedinc.grid(row = 5 , column =1)
```

```
262
263         pitchinc = tk.Label(self, textvariable = pitch_display, bg =
            ↪  "lightgrey" ,  width = 5, font=('helvetica', 25, 'bold'))
264         pitchinc.grid(row = 5 , column =2)
265
266
267         Speedinc = tk.Button(self, text="Increase",command=increase_fan,
            ↪  bg='cyan', fg='black',height = 2 , width =10, font=('helvetica',
            ↪  20, 'bold')) #
268         Speedinc.grid(row = 6 , column = 1)
269
270
271
272         Speeddec = tk.Button(self, text="Decrease",command=decrease_fan,
            ↪  bg='yellow', fg='black',height = 2 , width =10,
            ↪  font=('helvetica', 20, 'bold')) #
273         Speeddec.grid(row = 7 , column = 1)
274
275
276         pitchinc = tk.Button(self, text="Increase",command=increase_pitch,
            ↪  bg='cyan', fg='black', height = 2 , width =10,font=('helvetica',
            ↪  20, 'bold')) #
277         pitchinc.grid(row = 6 , column = 2)
278
279         pitchdec = tk.Button(self, text="Decrease",command=decrease_pitch,
            ↪  bg='yellow', fg='black',height = 2 , width =10,
            ↪  font=('helvetica', 20, 'bold')) #
280         pitchdec.grid(row = 7, column = 2)
281
282
283         Stopbtn = tk.Button(self, text="STOP FAN",command=stop_fan,height =
            ↪  2 , width =15, bg='red', fg='white', font=('helvetica', 20,
            ↪  'bold')) #
284         Stopbtn.grid(row = 5, column = 0)
285
286         spacer5 = tk.Label(self, text="")
287         spacer5.grid(row = 6 , column = 0)
288
289         resetpitch = tk.Button(self, text="Reset
            ↪  pitch",command=reset_pitch,height = 2 , width =15, bg='red',
            ↪  fg='white', font=('helvetica', 20, 'bold')) #
290         resetpitch.grid(row = 7 , column = 0)
291
292         spacer6 = tk.Label(self, text="")
293         spacer6.grid(row = 8 , column = 0)
294
```

```
295
296  ##########  PAGE 2 Measurements  ##########
297
298 class PageTwo(tk.Frame):
299
300     def __init__(self, parent, controller):
301         tk.Frame.__init__(self, parent)
302
303
304         self.windspeed   = tk.StringVar()
305         self.temperature = tk.StringVar()
306         self.humidity    = tk.StringVar()
307         self.pitch       = tk.StringVar()
308         self.airpressure = tk.StringVar()
309         self.dragforce   = tk.StringVar()
310         self.liftforce   = tk.StringVar()
311
312         def update_display():
313             """
314             Update measurement display with values from sensor_data object
315             Runs every 500ms
316             """
317             self.sensor_data = controller.sensor_data.get_data()
318             self.windspeed.set(str(self.sensor_data["windspeed"] / 10) + "
                ↪  m/s")  # Divide by 10 since windspeed is sent as tenths int
319             self.temperature.set(str(self.sensor_data["temperature"]) + "
                ↪  C")
320             self.humidity.set(str(self.sensor_data["humidity"]) + " %")
321             self.pitch.set(str(self.sensor_data["pitch"]) + u"\N{DEGREE
                ↪  SIGN}")
322             self.airpressure.set(str(self.sensor_data["airpressure"]) + "
                ↪  Pa")
323             self.dragforce.set(str(self.sensor_data["dragforce"]) + " N")
324             self.liftforce.set(str(self.sensor_data["liftforce"]) + " N")
325
326             self.after(500, update_display)
327
328         self.after(0, update_display)
329
330
331         SpeedAndPitch = tk.Button(self, text="Adjust speed/pitch",height =
            ↪  3, width = 18,command=lambda: controller.show_frame(PageOne),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
332         SpeedAndPitch.grid(row = 0, column = 0)
333
334         Measurments = tk.Button(self, text="Measurements",height = 3, width
            ↪  = 13, command=lambda: controller.show_frame(PageTwo),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
```

```
335         Measurments.grid(row = 0, column = 1)
336
337         probe = tk.Button(self, text="Adjust probe",height = 3, width = 13,
            ↪  command=lambda: controller.show_frame(PageThree), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
338         probe.grid(row = 0, column = 2)
339
340         export = tk.Button(self, text="Export data",height = 3, width = 11,
            ↪  command=lambda: controller.show_frame(PageFour), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
341         export.grid(row = 0, column = 3)
342
343
344         labelsp33 = tk.Label(self, text="",  font=('helvetica', 30, 'bold'))
345         labelsp33.grid(row = 1 , column = 1)
346
347
348         labeltitle2 = tk.Label(self, text="Measurements from sensors",
            ↪  bg='red', fg='white', font=('helvetica', 20, 'bold'))
349         labeltitle2.grid(row = 2 , column =0, columnspan = 5)
350
351         spacer3 = tk.Label(self, text="")
352         spacer3.grid(row = 3 , column = 0)
353
354
355
356         AirVelocity = tk.Button(self, text="Air Velocity", command=lambda:
            ↪  graph_window("SELECT time, windspeed FROM data WHERE time
            ↪  BETWEEN ? AND ?", "Windspeed", "m/s"),height = 2 , width =15,
            ↪  bg='cyan', fg='black', font=('helvetica', 15, 'bold')) #
357         AirVelocity.grid(row = 4, column = 0)
358         labelAirV = tk.Label(self, textvariable = self.windspeed, height = 2
            ↪  , width =15, bg='lightgrey', fg='black', font=('helvetica', 15,
            ↪  'bold')) #
359         labelAirV.grid(row = 4, column = 1)
360
361         spacer4 = tk.Label(self, text="")
362         spacer4.grid(row = 5 , column = 0)
363
364         Airtemp = tk.Button(self, text="Air Temperature",command=lambda:
            ↪  graph_window("SELECT time, temperature FROM data WHERE time
            ↪  BETWEEN ? AND ?","Temperature","Celsius"),height = 2 , width
            ↪  =15, bg='cyan', fg='black', font=('helvetica', 15, 'bold')) #
365         Airtemp.grid(row = 6, column = 0)
366         labeltemp = tk.Label(self, textvariable = self.temperature ,height =
            ↪  2 , width =15, bg='lightgrey', fg='black', font=('helvetica',
            ↪  15, 'bold')) #
```

```
367         labeltemp.grid(row = 6, column = 1)
368
369         spacer5 = tk.Label(self, text="")
370         spacer5.grid(row = 7 , column = 0)
371
372
373         Airhum = tk.Button(self, text="Air Humidity",command=lambda:
            ↪  graph_window("SELECT time, humidity FROM data WHERE time BETWEEN
            ↪  ? AND ?","Humidity","percent"),height = 2 , width =15,
            ↪  bg='cyan', fg='black', font=('helvetica', 15, 'bold')) #
374         Airhum.grid(row = 8, column = 0)
375         labelAirhum = tk.Label(self, text="1500",
            ↪  textvariable=self.humidity, height = 2 , width =15,
            ↪  bg='lightgrey', fg='black', font=('helvetica', 15, 'bold')) #
376         labelAirhum.grid(row = 8, column = 1)
377
378
379         Airpress = tk.Button(self, text="Air pressure",command=lambda:
            ↪  graph_window("SELECT time, airpressure FROM data WHERE time
            ↪  BETWEEN ? AND ?","Airpressure","Pascal"),height = 2 , width =15,
            ↪  bg='cyan', fg='black', font=('helvetica', 15, 'bold')) #
380         Airpress.grid(row = 4, column = 2)
381         labelAirpress = tk.Label(self, textvariable=self.airpressure, height
            ↪  = 2 , width =15, bg='lightgrey', fg='black', font=('helvetica',
            ↪  15, 'bold')) #
382         labelAirpress.grid(row = 4, column = 3)
383
384
385         forceH = tk.Button(self, text="Drag force",command=lambda:
            ↪  graph_window("SELECT time, dragforce FROM data WHERE time
            ↪  BETWEEN ? AND ?","Dragforce", "Newton"),height = 2 , width =15,
            ↪  bg='cyan', fg='black', font=('helvetica', 15, 'bold')) #
386         forceH.grid(row = 6, column = 2)
387         labelforceH = tk.Label(self, text="2 N",
            ↪  textvariable=self.dragforce, height = 2 , width =15,
            ↪  bg='lightgrey', fg='black', font=('helvetica', 15, 'bold')) #
388         labelforceH.grid(row = 6, column = 3)
389
390
391         froceV = tk.Button(self, text="Lift force",command=lambda:
            ↪  graph_window("SELECT time, liftforce FROM data WHERE time
            ↪  BETWEEN ? AND ?","Liftforce", "Newton"),height = 2 , width =15,
            ↪  bg='cyan', fg='black', font=('helvetica', 15, 'bold')) #
392         froceV.grid(row = 8, column = 2)
393         labelforceV = tk.Label(self, text="15 N",
            ↪  textvariable=self.liftforce, height = 2 , width =15,
            ↪  bg='lightgrey', fg='black', font=('helvetica', 15, 'bold')) #
```

```
        labelforceV.grid(row = 8, column = 3)


 ##########  PAGE 3 Røykprobe  ##########

class PageThree(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        SpeedAndPitch = tk.Button(self, text="Adjust speed/pitch",height =
        ↪  3, width = 18,command=lambda: controller.show_frame(PageOne),
        ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
        SpeedAndPitch.grid(row = 0 , column = 0)

        Measurments = tk.Button(self, text="Measurements",height = 3, width
        ↪  = 13, command=lambda: controller.show_frame(PageTwo),
        ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
        Measurments.grid(row = 0 , column = 1)

        probe = tk.Button(self, text="Adjust probe",height = 3, width =
        ↪  13,command=lambda: controller.show_frame(PageThree), bg='green',
        ↪  fg='white', font=('helvetica', 18, 'bold')) #
        probe.grid(row = 0, column = 2)

        export = tk.Button(self, text="Export data",height = 3, width =
        ↪  11,command=lambda: controller.show_frame(PageFour), bg='green',
        ↪  fg='white', font=('helvetica', 18, 'bold')) #
        export.grid(row = 0, column = 3)


        labelsp33 = tk.Label(self, text="",  font=('helvetica', 30, 'bold'))
        labelsp33.grid(row = 1 , column = 1)


        labeltitle2 = tk.Label(self, text="Set the smoke probe position",
        ↪  bg='red', fg='white', font=('helvetica', 20, 'bold'))
        labeltitle2.grid(row = 2 , column =0, columnspan = 5)

        spacer3 = tk.Label(self, text="")
        spacer3.grid(row = 3 , column = 0)


        bar = tk.Scale(self, from_=-200, to=200,orient=tk.HORIZONTAL,
        ↪  length=2000,tickinterval=50, width = 100 ,font=('helvetica', 10,
        ↪  'bold'))
```

```
429         bar.grid(row = 5 , column = 0, columnspan =4)
430 
431         labeinstructions = tk.Label(self, text="- Left    Right +",
            ↪  fg='black', font=('helvetica', 20, 'bold'))
432         labeinstructions.grid(row = 6 , column =0, columnspan = 5)
433 
434 
435 
436 ##########  PAGE 4 Eksport  ##########
437 
438 class PageFour(tk.Frame):
439 
440     def __init__(self, parent, controller):
441         tk.Frame.__init__(self, parent)
442 
443         self.session_start_time = datetime.datetime.now().isoformat()[:19]
444 
445 
446         SpeedAndPitch = tk.Button(self, text="Adjust speed/pitch",height =
            ↪  3, width = 18,command=lambda: controller.show_frame(PageOne),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
447         SpeedAndPitch.grid(row = 0 , column = 0)
448 
449         Measurments = tk.Button(self, text="Measurements",height = 3, width
            ↪  = 13, command=lambda: controller.show_frame(PageTwo),
            ↪  bg='green', fg='white', font=('helvetica', 18, 'bold')) #
450         Measurments.grid(row = 0 , column = 1)
451 
452         probe = tk.Button(self, text="Adjust probe",height = 3, width =
            ↪  13,command=lambda: controller.show_frame(PageThree), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
453         probe.grid(row = 0, column = 2)
454 
455         export = tk.Button(self, text="Export data",height = 3, width =
            ↪  11,command=lambda: controller.show_frame(PageFour), bg='green',
            ↪  fg='white', font=('helvetica', 18, 'bold')) #
456         export.grid(row = 0, column = 3)
457 
458         labelsp33 = tk.Label(self, text="",  font=('helvetica', 30, 'bold'))
459         labelsp33.grid(row = 1 , column = 1)
460 
461 
462         tkvar = tk.StringVar(self)
463         tkvar.set('Select start time')
464 
465         tkvar2 = tk.StringVar(self)
```

```
        tkvar2.set('Select ')

        labelmen= tk.Label(self, text = "Select start time",
        ↪  font=('helvetica', 15, 'bold'))
        labelmen.grid(row = 1 , column = 1)

        cb = ttk.Combobox(self)
        cb.set("Start time")
        cb.grid(row = 2, column = 1)

        cb2 = ttk.Combobox(self)
        cb2.set("End time")
        cb2.grid(row = 2, column = 2)

        labelmen= tk.Label(self, text = "Select end time",
        ↪  font=('helvetica', 15, 'bold'))
        labelmen.grid(row = 1 , column = 2)

        def update_times_list():
            """
            Read database and list all times in the dropdown boxes
            """
            c = controller.cursor
            c.execute('SELECT time FROM data')
            cblist = c.fetchall()
            cb['values'] = cblist
            cb2['values'] = cblist
        update_times_list()

        def export_to_csv(startTime=None, endTime=None):
            """
            Write data between the selected times (inclusive) to a CSV-file
            Files are placed in the public_html folder for the Apache web
↪   server
            """
            if startTime is None or endTime is None:
                startTime = cb.get()
                endTime = cb2.get()
            if (startTime>=endTime):
                tk.messagebox.showerror("Error", "Start time can not be less
                ↪  or equal to end time")
            else:
                export_file_path =
                ↪  f"/var/www/idm.com/public_html/{endTime}.csv"

                cursor = controller.cursor
```

```
                q = (startTime, endTime)
                cursor.execute("SELECT * FROM data WHERE time BETWEEN ? AND
                ↪  ?", q)

                with open(export_file_path, "w") as csv_file:
                    csv_writer = csv.writer(csv_file)
                    csv_writer.writerow([i[0] for i in cursor.description])
                    csv_writer.writerows(cursor.fetchall())

        def export_session():
            session_end_time = datetime.datetime.now().isoformat()[:19]
            export_to_csv(self.session_start_time, session_end_time)

        def export_session_refresh():
            self.session_start_time =
            ↪  datetime.datetime.now().isoformat()[:19]

        export_selected = tk.Button(self, text="Export\nselected",height =
        ↪  2, width = 10,command=export_to_csv, bg='red', fg='white',
        ↪  font=('helvetica', 20, 'bold')) #
        export_selected.grid(row = 2, column = 0)

        updateliste = tk.Button(self, text="Update list ",height = 2, width
        ↪  = 10,command=update_times_list, bg='red', fg='white',
        ↪  font=('helvetica', 20, 'bold')) #
        updateliste.grid(row = 3, column = 0)


        labelspacer= tk.Label(self, text = "", font=('helvetica', 20,
        ↪  'bold'))
        labelspacer.grid(row = 43, column = 0)

        export_session_button = tk.Button(self, text="Export\nsession",
        ↪  height = 2, width = 10,command=export_session, bg='red',
        ↪  fg='white', font=('helvetica', 20, 'bold')) #
        export_session_button.grid(row = 5, column = 0)


        Clear_session_button = tk.Button(self, text="Clear\nsession", height
        ↪  = 2, width = 10,command=export_session_refresh, bg='red',
        ↪  fg='white', font=('helvetica', 20, 'bold')) #
        Clear_session_button.grid(row = 5, column = 1)

        poweroff = tk.Button(self, text="Power off ", height = 2, width =
        ↪  10, command=controller.power_shutdown, bg='red', fg='white',
        ↪  font=('helvetica', 20, 'bold')) #
```

```
        poweroff.grid(row = 4, column = 3)


def graph_window(database_val,label,yaxislab):
    root = tk.Tk()
    root.wm_title("Embedding graf in new Tk window")
    graf_name = tk.Label(root, text= label,   bg='green', fg='white',
    ↪  font=('helvetica', 15, 'bold'))
    graf_name.pack(side=tk.TOP)
    f = Figure(figsize=(5,4), dpi=100)
    a = f.add_subplot(111)
    insert_val =database_val
    f.ylabel = 'test'

    def animate(i):
        c1 = app.cursor
        c1.execute("SELECT time FROM data ORDER BY time DESC LIMIT 1")
        now_time_string = c1.fetchone()[0]
        now_time = datetime.datetime.fromisoformat(now_time_string)

        then_time = now_time - datetime.timedelta(seconds=13)
        then_time_string = then_time.isoformat()

        b = (then_time_string, now_time_string)

        c = app.cursor
        c.execute(insert_val,b)
        fetch = c.fetchall()

        Xaxes = [x[-5:] for (x, y) in fetch]
        Yaxes = [y for (x, y) in fetch]

        pltXaxes = np.array(Xaxes)
        pltYaxes = np.array(Yaxes)

        a.clear()
        a.plot(pltXaxes,pltYaxes)
        a.set_ylabel(yaxislab)
        a.set_xlabel("Time")


    canvas = FigureCanvasTkAgg(f, master=root)  # A tk.DrawingArea.

    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

```

```
    def _quit():
        root.quit()
        root.destroy()


    button = tk.Button(master=root, text="Quit", command=_quit, bg='red',
    ↪  fg='white', font=('helvetica', 30, 'bold'))
    button.pack(side=tk.BOTTOM)

    root.ani = animation.FuncAnimation(f,animate, interval=1000)

    root.geometry("800x480+0+0")
    root.attributes("-fullscreen", True)
    root.mainloop()


# Instantiate main GUI class
app = IDMGUI()

# Remove header and force window to touchscreen
app.geometry("800x480+0+0")
app.update_idletasks()
app.attributes("-fullscreen", True)
app.update_idletasks()

# Run the program
app.mainloop()

# Shut down the communicator thread when the GUI closes
app.comm.close_serial()
```

B.2. idmserial.py

```
#!/usr/bin/env python3
#
# Author: Kristian Auestad
```

```
#
import threading
import dataobject

import time
import serial

class SerialCommunicator:
    class SerialReceiver(threading.Thread):
        def __init__(self, serialport, data, stop_event):
            self.ser = serialport
            self.data = data
            self.stop_event = stop_event
            threading.Thread.__init__(self)

        def run(self):
            #print(__name__, "Running")
            while not self.stop_event.is_set():
                in_data = self.ser.readline()
                if len(in_data) > 0:
                    self.data.parse_datastring(in_data)
                    #print(self.data.get_data())  # Debug
            #print(__name__, "Stopping")


    class SerialTransmitter:
        def __init__(self, serialport):
            self.ser = serialport

        def transmit(self, message):
            # Ensure message is properly terminated
            if message[-1:] != "X":
                if message[-1:] == "x":
                    message = message[:-1]
                message = message + "X"
            self.ser.write(message.encode())
            #print(__name__, "Transmitted", message.encode())


    def __init__(self, data, stop_event,
                 port='/dev/ttyACM0', rate=57600, timeout=1):
        #print(__name__, "Init")
        self.data = data
        self.stop_event = stop_event

        while True:  # Wait for the serial port to connect
```

```
            self.ser = self.connect_serial(port, rate, timeout)
            if self.ser is not None:
                break

        self.transmitter = self.SerialTransmitter(self.ser)
        self.receiver = self.SerialReceiver(self.ser, self.data,
                                            self.stop_event)
        self.receiver.start()

    def connect_serial(self, port, rate, timeout):
        try:
            return serial.Serial(port=port, baudrate=rate, timeout=timeout)
        except:
            return None

    def close_serial(self):
        #print(__name__, "Close")
        self.stop_event.set()
        self.receiver.join()
        self.ser.close()

    def transmit(self, message):
        self.transmitter.transmit(message)


if __name__ == "__main__":
    print("Start")
    data = dataobject.DataObject()
    event = threading.Event()
    comm = SerialCommunicator(data, event)
    for i in range(4):
        print(i, data.get_data())
        time.sleep(1.5)
    comm.transmit("Fnord")
    comm.close()
    print("End")
```

B.3. dataobject.py

```
#!/usr/bin/env python3
#
# Author: Kristian Auestad
#
import threading
import datetime
import sqlite3
import re

class DataObject(object):
    def __init__(self, lock=None):
        self.__lock = lock or threading.RLock()

        # Match format 0000-00-00T00:00:00
        self.re = re.compile(r"\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}")

        self.__datestring = "fnord"
        self.__datetime = datetime.datetime(1900,1,1)
        self.__windspeed = -1
        self.__temperature = -1
        self.__humidity = -1
        self.__pitch = -1
        self.__airpressure = -1
        self.__dragforce = -1
        self.__liftforce = -1
        self.__bools = -1

    def parse_datastring(self, data):
        with self.__lock:
            try:
                (datestring, windspeed, temperature, humidity, pitch,
                ↪  airpressure,
                 dragforce, liftforce, bools) = data.split(b"|")
            except ValueError:  # Potential missing bools variable, try
            ↪  again
                try:
                    (datestring, windspeed, temperature, humidity, pitch,
                    ↪  airpressure,
                     dragforce, liftforce) = data.split(b"|")
                    bools = -1
                except ValueError:  # Received string is in wrong format
                    datestring, windspeed, temperature =
                    ↪  b"1900-01-01T12:00:00", -1, -1
```

```
                    humidity, pitch, bools = -1, -1, -1
                    airpressure, dragforce, liftforce = -1, -1, -1

            if isinstance(self.__datestring, str):
                # Make sure date string is not in bytes format
                self.__datestring = datestring.decode("utf-8")
            else:
                self.__datestring = datestring

            if self.re.match(self.__datestring):
                # Make sure date string is in the correct ISO-8601 format
                self.__datetime =
                ↪  datetime.datetime.fromisoformat(self.__datestring)
            else:
                self.__datetime =
                ↪  datetime.datetime.fromisoformat("2100-01-01T12:00:00")

            self.__windspeed = int(windspeed)
            self.__temperature = int(temperature)
            self.__humidity = int(humidity)
            self.__pitch = int(pitch)
            self.__airpressure = int(airpressure)
            self.__dragforce = int(dragforce)
            self.__liftforce = int(liftforce)
            self.__bools = int(bools)

    def get_data(self):
        with self.__lock:
            return dict(time=self.__datetime, timestring=self.__datestring,
            ↪  windspeed=self.__windspeed,
                        temperature=self.__temperature,
                        ↪  humidity=self.__humidity,
                        pitch=self.__pitch, airpressure=self.__airpressure,
                        dragforce=self.__dragforce,
                        ↪  liftforce=self.__liftforce)
```

B.4. IDM.ino

```
// #define DEBUG
//
// Author: Kristian Auestad & Steffen Barskrind
//         + Joachim Haug
//
#include <ThreeWire.h>
#include <RtcDS1302.h> // https://github.com/Makuna/Rtc
#include <dht.h>       // https://github.com/adafruit/DHT-sensor-library
#include <PID_v1.h>    // https://github.com/br3ttb/Arduino-PID-Library/

#include "boolToByte.h"

    // Data structs
    struct SensorData {
    int windSpeed;
    int temperature;
    int humidity;
    int pitch;
    int airPressure;
    int dragForce;
    int liftForce;
    bool hatchClosed;
    bool fanRunning;
};

struct InputData {
    int setWindSpeed;
    int setPitch;
};

struct SensorData sensorData;
struct InputData inputData;

// ++ Sensors
const int startStopPin = 52;
const int hatchSafetyPin = 51;
const int potPin = A7;

// DHT11 temperature and humidity sensor
const int dht11pin = 3;
dht DHT;

```

```
// Differential pressure sensor
const int dpSensorPin = A0;
float rho = 1.204; // Air density
int offset = 0;
int offset_size = 10;
int veloc_mean_size = 20;
int zero_span = 2;
// -- Sensors


// PID controller variables
const int PIDoutputPin = 8;
double Setpoint, Input, Output;  // PID Process parameters
double Kp = 30, Ki = 50, Kd = 0;  // PID constants
PID fanController(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);  // PID
↪  controller


// Timing variables
unsigned long previousTransmitTime = 0;
const unsigned int transmitInterval = 1000;
unsigned long previousPollTime = 0;
const unsigned int pollInterval = 200;
unsigned long previousDHT11pollTime = 0;
const unsigned int DHT11pollInterval = 2000;
unsigned long previousPIDtime = 0;
const unsigned int PIDinterval = 200;

// Serial input
bool stringComplete = false;
String inputString = "";

// Real-time clock
ThreeWire myWire(6,7,5); // IO, SCLK, CE
RtcDS1302<ThreeWire> Rtc(myWire);
char datetimestring[20];

void setup() {
    Serial.begin(57600);

    // ++ Real-time clock configuration
    #ifdef DEBUG
    Serial.print("compiled: ");
    Serial.print(__DATE__);
    Serial.println(__TIME__);
    #endif //DEBUG
```

```
88
89      Rtc.Begin();
90
91      RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
92      #ifdef DEBUG
93      printDateTime(compiled);
94      Serial.println();
95      #endif //DEBUG
96
97      if (!Rtc.IsDateTimeValid()) {
98          // Common Causes:
99          //    1) first time you ran and the device wasn't running yet
100         //    2) the battery on the device is low or even missing
101         #ifdef DEBUG
102         Serial.println("RTC lost confidence in the DateTime!");
103         #endif //DEBUG
104         Rtc.SetDateTime(compiled);
105     }
106
107     if (Rtc.GetIsWriteProtected()) {
108         #ifdef DEBUG
109         Serial.println("RTC was write protected, enabling writing now");
110         #endif //DEBUG
111         Rtc.SetIsWriteProtected(false);
112     }
113
114     if (!Rtc.GetIsRunning()) {
115         #ifdef DEBUG
116         Serial.println("RTC was not actively running, starting now");
117         #endif //DEBUG
118         Rtc.SetIsRunning(true);
119     }
120
121     RtcDateTime now = Rtc.GetDateTime();
122     if (now < compiled) {
123         #ifdef DEBUG
124         Serial.println("RTC is older than compile time!  (Updating
            ↪  DateTime)");
125         #endif //DEBUG
126         Rtc.SetDateTime(compiled);
127     }
128     else if (now > compiled) {
129         #ifdef DEBUG
130         Serial.println("RTC is newer than compile time. (this is
            ↪  expected)");
131         #endif //DEBUG
```

```
132     }
133     else if (now == compiled) {
134         #ifdef DEBUG
135         Serial.println("RTC is the same as compile time! (not expected but
            ↪  all is fine)");
136         #endif //DEBUG
137     }
138     // -- Real-time clock configuration
139 
140     // Port initialisation
141     pinMode(dpSensorPin, INPUT);
142     pinMode(startStopPin, INPUT_PULLUP);
143     pinMode(hatchSafetyPin, INPUT_PULLUP);
144     pinMode(PIDoutputPin, OUTPUT);
145 
146     // ++ Faux sensors
147     sensorData.dragForce = -6;
148     sensorData.liftForce = -7;
149     // -- Faux sensors
150 
151     // Set initial values
152     inputData.setWindSpeed = 0;
153     inputData.setPitch = 0;
154 
155     // PID initialisation
156     Setpoint = 0;
157     fanController.SetMode(AUTOMATIC);
158 
159     sleep(5000); // Delay DP sensor initialisation
160                  // to make sure fan settles first
161 
162     // DP sensor initialisation
163     for (int ii = 0; ii < offset_size; ii++) {
164         offset += analogRead(dpSensorPin) - (1023 / 2);
165     }
166     offset /= offset_size;
167 }
168 
169 void loop() {
170     unsigned long currentTime = millis();
171 
172     // Read sensors every pollInterval
173     if (currentTime - previousPollTime >= pollInterval) {
174       previousPollTime = currentTime;
175       // ++ Faux sensors
176       // sensorData.windSpeed = map(analogRead(potPin), 0, 1023, 0, 200);
```

```
      sensorData.windSpeed = measureWindSpeed();
      sensorData.hatchClosed = (digitalRead(startStopPin)) ? false : true;
      sensorData.fanRunning = (digitalRead(hatchSafetyPin)) ? false : true;
      sensorData.pitch = inputData.setPitch;
      sensorData.airPressure = inputData.setWindSpeed;
      // -- Faux sensors
    }

    if (currentTime - previousPIDtime >= PIDinterval) {
        previousPIDtime = currentTime;
        fanController.Compute();  // Compute fan speed
        analogWrite(PIDoutputPin, Output);  // Output fan PWM signal
    }

    // Read the DHT11 sensor slower because of hardware limitations
    if (currentTime - previousDHT11pollTime >= DHT11pollInterval) {
        previousDHT11pollTime = currentTime;
        DHT.read11(dht11pin);
        sensorData.temperature = DHT.temperature;
        sensorData.humidity = DHT.humidity;
    }

    // Transmit data every transmitInterval
    if (currentTime - previousTransmitTime >= transmitInterval) {
        previousTransmitTime = currentTime;
        RtcDateTime now = Rtc.GetDateTime();
        formatDateTime(now, datetimestring);

        transmitData(sensorData, datetimestring);

        if (!now.IsValid()) {
            // Common Causes:
            //    1) the battery on the device is low or even missing and
            ↪  the power line was disconnected
            Serial.println("RTC lost confidence in the DateTime!");
        }
    }

    // Test
    if (stringComplete) {
        //Serial.println(inputString);
        switch (inputString.charAt(0)) {
          case 'W': case 'w':
            inputString.remove(0,1);
            inputString.remove(inputString.length());
            inputData.setWindSpeed = inputString.toInt();
```

```
            break;
          case 'P': case 'p':
            inputString.remove(0,1);
            inputString.remove(inputString.length());
            inputData.setPitch = inputString.toInt();
            break;
          default:
            break;
        }
        inputString = "";
        stringComplete = false;
    }
}

// Serial data receiving event handler
void serialEvent() {
  //Serial.print("P1: ");
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    if (inChar != '\n') {
        inputString += inChar;
    }
    // if the incoming character is a newline, set a flag so the main loop
    ↪  can
    // do something about it:
    if (inChar == 'X') {
        stringComplete = true;
    }
  }
}

// Order: time|windspeed|temperature|humidity|pitch|bools
void transmitData(struct SensorData data, char* datestring) {
    Serial.print(datestring); Serial.print("|");
    Serial.print(data.windSpeed); Serial.print("|");
    Serial.print(data.temperature); Serial.print("|");
    Serial.print(data.humidity); Serial.print("|");
    Serial.print(data.pitch); Serial.print("|");
    Serial.print(data.airPressure); Serial.print("|");
    Serial.print(data.dragForce); Serial.print("|");
    Serial.print(data.liftForce); Serial.print("|");
    Serial.print(boolToByte(data.hatchClosed, data.fanRunning));
    Serial.println();
}
```

```
267
268 int measureWindSpeed() {
269     float adc_avg = 0.0;
270     float veloc = 0.0;
271     for (int ii = 0; ii < veloc_mean_size; ii++) {
272         adc_avg += analogRead(dpSensorPin) - offset;
273     }
274     adc_avg /= veloc_mean_size;
275     // make sure if the ADC reads below 512, then we equate it to a negative
      ↪  velocity
276     if (adc_avg > 512 - zero_span && adc_avg < 512 + zero_span)
277     {
278     }
279     else {
280         if (adc_avg < 512) {
281             veloc = -sqrt((-10000.0 * ((adc_avg / 1023.0) - 0.5)) / rho);
282         } else {
283             veloc = sqrt((10000.0 * ((adc_avg / 1023.0) - 0.5)) / rho);
284         }
285     }
286     return (int)(veloc * 10);  // Velocity is stored as a decimal int
287 }
288
289
290 #define countof(a) (sizeof(a) / sizeof(a[0]))
291
292 void formatDateTime(const RtcDateTime& dt, char* returnstring) {
293     char datestring[20];
294
295     snprintf_P(datestring,
296             countof(datestring),
297             PSTR("%04u-%02u-%02uT%02u:%02u:%02u"),
298             dt.Year(),
299             dt.Month(),
300             dt.Day(),
301             dt.Hour(),
302             dt.Minute(),
303             dt.Second() );
304     strcpy(returnstring, datestring);
305 }
306
307 void printDateTime(const RtcDateTime& dt) {
308     char datestring[20];
309
310     snprintf_P(datestring,
311             countof(datestring),
```

```
            PSTR("%04u-%02u-%02u %02u:%02u:%02u"),
            dt.Year(),
            dt.Month(),
            dt.Day(),
            dt.Hour(),
            dt.Minute(),
            dt.Second() );
    Serial.print(datestring);
}
```

B.5. boolToByte.h

```
// For Arduino
// Convert booleans to a byte map
// Author: Kristian Auestad
//
#ifndef _BOOLTOBYTE_H
#define _BOOLTOBYTE_H

byte boolToByte(bool bool1) {
    byte ret = 0;
    ret = (bool1) ? ret | B00000001 : ret & B11111110;
    return ret;
}

byte boolToByte(bool bool1, bool bool2) {
    byte ret = 0;
    ret = (bool1) ? ret | B00000001 : ret & B11111110;
    ret = (bool2) ? ret | B00000010 : ret & B11111101;
    return ret;
}

byte boolToByte(bool bool1, bool bool2, bool bool3) {
    byte ret = 0;
    ret = (bool1) ? ret | B00000001 : ret & B11111110;
    ret = (bool2) ? ret | B00000010 : ret & B11111101;
```

```
    ret = (bool3) ? ret | B00000100 : ret & B11111011;
    return ret;
}

byte boolToByte(bool bool1, bool bool2, bool bool3, bool bool4) {
    byte ret = 0;
    ret = (bool1) ? ret | B00000001 : ret & B11111110;
    ret = (bool2) ? ret | B00000010 : ret & B11111101;
    ret = (bool3) ? ret | B00000100 : ret & B11111011;
    ret = (bool4) ? ret | B00001000 : ret & B11110111;
    return ret;
}

byte boolToByte(bool bool1, bool bool2, bool bool3, bool bool4, bool bool5)
 ↪  {
    byte ret = 0;
    ret = (bool1) ? ret | B00000001 : ret & B11111110;
    ret = (bool2) ? ret | B00000010 : ret & B11111101;
    ret = (bool3) ? ret | B00000100 : ret & B11111011;
    ret = (bool4) ? ret | B00001000 : ret & B11110111;
    ret = (bool5) ? ret | B00010000 : ret & B11101111;
    return ret;
}

byte boolToByte(bool bool1, bool bool2, bool bool3, bool bool4, bool bool5,
                bool bool6) {
    byte ret = 0;
    ret = (bool1) ? ret | B00000001 : ret & B11111110;
    ret = (bool2) ? ret | B00000010 : ret & B11111101;
    ret = (bool3) ? ret | B00000100 : ret & B11111011;
    ret = (bool4) ? ret | B00001000 : ret & B11110111;
    ret = (bool5) ? ret | B00010000 : ret & B11101111;
    ret = (bool6) ? ret | B00100000 : ret & B11011111;
    return ret;
}

byte boolToByte(bool bool1, bool bool2, bool bool3, bool bool4, bool bool5,
                bool bool6, bool bool7) {
    byte ret = 0;
    ret = (bool1) ? ret | B00000001 : ret & B11111110;
    ret = (bool2) ? ret | B00000010 : ret & B11111101;
    ret = (bool3) ? ret | B00000100 : ret & B11111011;
    ret = (bool4) ? ret | B00001000 : ret & B11110111;
    ret = (bool5) ? ret | B00010000 : ret & B11101111;
    ret = (bool6) ? ret | B00100000 : ret & B11011111;
    ret = (bool7) ? ret | B01000000 : ret & B10111111;
```

```
70      return ret;
71  }
72
73  byte boolToByte(bool bool1, bool bool2, bool bool3, bool bool4, bool bool5,
74                  bool bool6, bool bool7, bool bool8) {
75      byte ret = 0;
76      ret = (bool1) ? ret | B00000001 : ret & B11111110;
77      ret = (bool2) ? ret | B00000010 : ret & B11111101;
78      ret = (bool3) ? ret | B00000100 : ret & B11111011;
79      ret = (bool4) ? ret | B00001000 : ret & B11110111;
80      ret = (bool5) ? ret | B00010000 : ret & B11101111;
81      ret = (bool6) ? ret | B00100000 : ret & B11011111;
82      ret = (bool7) ? ret | B01000000 : ret & B10111111;
83      ret = (bool8) ? ret | B10000000 : ret & B01111111;
84      return ret;
85  }
86
87  #endif // _BOOLTOBYTE_H
```

B.6. PIDcontroller.ino

```
1   #include <PID_v1.h> // PID library
2
3   #define E 8 // Definerer kontrollsignal E til AC motordriver til port 8 på
    ↪  Arduino kortet
4   #define A A0 // Definerer analogt måledata A fra sensor for måling av
    ↪  vindhastighet til port A0 på Arduino kortet
5
6
7   float rho = 1.204; // tykkelse på luften
8
9   // parametere for gjennomsnitt og offset
10  int offset = 0;
11  int offset_size = 10;
12  int veloc_mean_size = 20;
13  int zero_span = 2;
```

```
14
15 double Setpoint, Input, Output; // Setpoint er ønsket hastighet, Input er
   ↪ målt vindhastighet fra sensor, Output is PWM signal til AC motordriver
16 double Kp=30, Ki=50, Kd=0; // PID parameterne
17
18 PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT); // Bruker
   ↪ definerte variabler som argumenter til PID-kontroller bibliotek
   ↪ funksjonen
19
20
21 void setup() {
22
23   pinMode(A, INPUT);
24   pinMode(DAC, OUTPUT);
25
26   Serial.begin(9600);
27   for (int ii=0;ii<offset_size;ii++){
28     offset += analogRead(A0)-(1023/2);
29   }
30   offset /= offset_size;
31   Setpoint = 15; // Definerer ønsket vindhastighet i m/s
32   myPID.SetMode(AUTOMATIC); // PID mode satt til automatic så den utfører
     ↪ hastighetsjusteringer automatisk
33
34
35 }
36
37 // Funksjon for å måle vindhastighet
38 void measureSpeed(){
39   float adc_avg = 0; float veloc = 0.0;
40   for (int ii=0;ii<veloc_mean_size;ii++){
41     adc_avg+= analogRead(A0)-offset;
42   }
43   adc_avg/=veloc_mean_size;
44
45   // make sure if the ADC reads below 512, then we equate it to a negative
     ↪ velocity
46   if (adc_avg>512-zero_span and adc_avg<512+zero_span){
47   } else{
48     if (adc_avg<512){
49       veloc = -sqrt((-10000.0*((adc_avg/1023.0)-0.5))/rho);
50     } else{
51       veloc = sqrt((10000.0*((adc_avg/1023.0)-0.5))/rho);
52     }
53   }
54   Serial.println(veloc); // print velocity
```

```
  delay(10); // delay for stability
  }

void PWMout(){ // PWM funksjon
  analogWrite(DAC, Output); // Sender PWM signal til AC motordriver
  }


void loop() { // Hovedløkke funksjon

  measureSpeed(); // Måler vindhastighet
  myPID.Compute(); // Sammenligner målt vindhastighet med ønsket vindhastighet og gjør nødvendige hastighetsjusteringer
  PWMout(); // Sender PWM spenning til AC motordriver som er proporsjonalt med hastighetsberegningen fra PID-kontrolleren
  }
```

B.7. PIDlibheader.h

```
#ifndef PID_v1_h
#define PID_v1_h
#define LIBRARY_VERSION        1.2.1

class PID
{


  public:

  //Constants used in some of the functions below
  #define AUTOMATIC        1
  #define MANUAL        0
  #define DIRECT  0
  #define REVERSE  1
  #define P_ON_M 0
  #define P_ON_E 1
```

```
18
19    //commonly used functions
20      PID(double*, double*, double*,
21          double, double, double, int, int);
22
23
24      PID(double*, double*, double*,
25          double, double, double, int);
26
27      void SetMode(int Mode);
28
29      bool Compute();
30
31
32
33
34      void SetOutputLimits(double, double);
35
36
37
38
39
40    //available but not commonly used functions
41      void SetTunings(double, double,
42                      double);
43
44      void SetTunings(double, double,
45                      double, int);
46
47          void SetControllerDirection(int);
48
49
50
51      void SetSampleTime(int);
52
53
54
55
56    //Display functions
57          double GetKp();
58          double GetKi();
59          double GetKd();
60          int GetMode();
61          int GetDirection();
62
63    private:
```

```
64          void Initialize();
65
66          double dispKp;
67          double dispKi;
68          double dispKd;
69
70          double kp;
71      double ki;
72      double kd;
73
74          int controllerDirection;
75          int pOn;
76
77      double *myInput;
78      double *myOutput;
79      double *mySetpoint;
80
81
82          unsigned long lastTime;
83          double outputSum, lastInput;
84
85          unsigned long SampleTime;
86          double outMin, outMax;
87          bool inAuto, pOnE;
88  };
89  #endif
```

B.8. PIDlibsource.cpp

```
1   /*
2    * Arduino PID Library - Version 1.2.1
3    * by Brett Beauregard <br3ttb@gmail.com> brettbeauregard.com
4    *
5    * This Library is licensed under the MIT License
6
7
```

```
#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#include <PID_v1.h>

/*Constructor (...)
 *    The parameters specified here are those for for which we can't set up
 *    reliable defaults, so we need to have the user set them.
 */
PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int POn, int ControllerDirection)
{
    myOutput = Output;
    myInput = Input;
    mySetpoint = Setpoint;
    inAuto = false;

    PID::SetOutputLimits(0, 255);


    SampleTime =
    ↪  100;

    PID::SetControllerDirection(ControllerDirection);
    PID::SetTunings(Kp, Ki, Kd, POn);

    lastTime = millis()-SampleTime;
}

/*Constructor (...)
 *    To allow backwards compatability for v1.1, or for people that just
↪  want
 *    to use Proportional on Error without explicitly saying so
 */

PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int ControllerDirection)
    :PID::PID(Input, Output, Setpoint, Kp, Ki, Kd, P_ON_E,
    ↪  ControllerDirection)
{

}

```

```
51
52  /* Compute()
53   *     This, as they say, is where the magic happens.  this function should
    ↪  be called
54   *   every time "void loop()" executes.  the function will decide for itself
    ↪  whether a new
55   *   pid Output needs to be computed.  returns true when the output is
    ↪  computed,
56   *   false when nothing has been done.
57   */
58  bool PID::Compute()
59  {
60     if(!inAuto) return false;
61     unsigned long now = millis();
62     unsigned long timeChange = (now - lastTime);
63     if(timeChange>=SampleTime)
64     {
65        /*Compute all the working error variables*/
66        double input = *myInput;
67        double error = *mySetpoint - input;
68        double dInput = (input - lastInput);
69        outputSum+= (ki * error);
70
71        /*Add Proportional on Measurement, if P_ON_M is specified*/
72        if(!pOnE) outputSum-= kp * dInput;
73
74        if(outputSum > outMax) outputSum= outMax;
75        else if(outputSum < outMin) outputSum= outMin;
76
77        /*Add Proportional on Error, if P_ON_E is specified*/
78             double output;
79        if(pOnE) output = kp * error;
80        else output = 0;
81
82        /*Compute Rest of PID Output*/
83        output += outputSum - kd * dInput;
84
85              if(output > outMax) output = outMax;
86        else if(output < outMin) output = outMin;
87              *myOutput = output;
88
89        /*Remember some variables for next time*/
90        lastInput = input;
91        lastTime = now;
92              return true;
93     }
```

```
94      else return false;
95  }
96
97  /* SetTunings(...)
98   * This function allows the controller's dynamic performance to be adjusted.
99   * it's called automatically from the constructor, but tunings can also
100  * be adjusted on the fly during normal operation
101  */
102 void PID::SetTunings(double Kp, double Ki, double Kd, int POn)
103 {
104    if (Kp<0 || Ki<0 || Kd<0) return;
105
106    pOn = POn;
107    pOnE = POn == P_ON_E;
108
109    dispKp = Kp; dispKi = Ki; dispKd = Kd;
110
111    double SampleTimeInSec = ((double)SampleTime)/1000;
112    kp = Kp;
113    ki = Ki * SampleTimeInSec;
114    kd = Kd / SampleTimeInSec;
115
116   if(controllerDirection ==REVERSE)
117    {
118       kp = (0 - kp);
119       ki = (0 - ki);
120       kd = (0 - kd);
121    }
122 }
123
124 /* SetTunings(...)
125  * Set Tunings using the last-rembered POn setting
126  */
127 void PID::SetTunings(double Kp, double Ki, double Kd){
128     SetTunings(Kp, Ki, Kd, pOn);
129 }
130
131 /* SetSampleTime(...)
132  * sets the period, in Milliseconds, at which the calculation is performed
133  */
134 void PID::SetSampleTime(int NewSampleTime)
135 {
136    if (NewSampleTime > 0)
137    {
138       double ratio  = (double)NewSampleTime
139                       / (double)SampleTime;
```

```
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

/* SetOutputLimits(...)
 *     This function will be used far more often than SetInputLimits.  while
 *  the input to the controller will generally be in the 0-1023 range (which is
 *  the default already,)  the output will be a little different.  maybe they'll
 *  be doing a time window and will need 0-8000 or something.  or maybe they'll
 *  want to clamp it from 0-125.  who knows.  at any rate, that can all be done
 *  here.
 */
void PID::SetOutputLimits(double Min, double Max)
{
   if(Min >= Max) return;
   outMin = Min;
   outMax = Max;

   if(inAuto)
   {
           if(*myOutput > outMax) *myOutput = outMax;
           else if(*myOutput < outMin) *myOutput = outMin;

           if(outputSum > outMax) outputSum= outMax;
           else if(outputSum < outMin) outputSum= outMin;
   }
}

/* SetMode(...)
 * Allows the controller Mode to be set to manual (0) or Automatic (non-zero)
 * when the transition from manual to auto occurs, the controller is
 * automatically initialized
 */
void PID::SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto && !inAuto)
    {  /*we just went from manual to auto*/
        PID::Initialize();
```

```
    }
    inAuto = newAuto;
}

/* Initialize()
 *        does all the things that need to happen to ensure a bumpless
 ↪  transfer
 *  from manual to automatic mode.
 */
void PID::Initialize()
{
   outputSum = *myOutput;
   lastInput = *myInput;
   if(outputSum > outMax) outputSum = outMax;
   else if(outputSum < outMin) outputSum = outMin;
}

/* SetControllerDirection(...)
 * The PID will either be connected to a DIRECT acting process (+Output
 ↪  leads
 * to +Input) or a REVERSE acting process(+Output leads to -Input.)  we need
 ↪  to
 * know which one, because otherwise we may increase the output when we
 ↪  should
 * be decreasing.  This is called from the constructor.
 */
void PID::SetControllerDirection(int Direction)
{
   if(inAuto && Direction !=controllerDirection)
   {
            kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
   controllerDirection = Direction;
}

/* Status Funcions
 * Just because you set the Kp=-1 doesn't mean it actually happened.  these
 * functions query the internal state of the PID.  they're here for display
 * purposes.  this are the functions the PID Front-end uses for example
 */
double PID::GetKp(){ return  dispKp; }
double PID::GetKi(){ return  dispKi;}
double PID::GetKd(){ return  dispKd;}
int PID::GetMode(){ return  inAuto ? AUTOMATIC : MANUAL;}
```

```
223 int PID::GetDirection(){ return controllerDirection;}
```

B.9. HX711kalibrering.ino

```
1  /*
2   Example using the SparkFun HX711 breakout board with a scale
3   By: Nathan Seidle
4   SparkFun Electronics
5   Date: November 19th, 2014
6   License: This code is public domain but you buy me a beer if you use this
    ↪  and we meet someday (Beerware license).
7
8   This is the calibration sketch. Use it to determine the calibration_factor
    ↪  that the main example uses. It also
9   outputs the zero_factor useful for projects that have a permanent mass on
    ↪  the scale in between power cycles.
10
11  Setup your scale and start the sketch WITHOUT a weight on the scale
12  Once readings are displayed place the weight on the scale
13  Press +/- or a/z to adjust the calibration_factor until the output readings
    ↪  match the known weight
14  Use this calibration_factor on the example sketch
15
16  This example assumes pounds (lbs). If you prefer kilograms, change the
    ↪  Serial.print(" lbs"); line to kg. The
17  calibration factor will be significantly different but it will be linearly
    ↪  related to lbs (1 lbs = 0.453592 kg).
18
19  Your calibration factor may be very positive or very negative. It all
    ↪  depends on the setup of your scale system
20  and the direction the sensors deflect from zero state
21  This example code uses bogde's excellent
    ↪  library:"https://github.com/bogde/HX711"
22  bogde's library is released under a GNU GENERAL PUBLIC LICENSE
23  Arduino pin 2 -> HX711 CLK
24  3 -> DOUT
```

```
 5V -> VCC
 GND -> GND

 Most any pin on the Arduino Uno will be compatible with DOUT/CLK.

 The HX711 board can be powered from 2.7V to 5V so the Arduino 5V power should be fine.

*/

#include "HX711.h"

#define DOUT  3
#define CLK  2

HX711 scale;

float calibration_factor = -7050; //-7050 worked for my 440lb max scale setup

void setup() {
  Serial.begin(9600);
  Serial.println("HX711 calibration sketch");
  Serial.println("Remove all weight from scale");
  Serial.println("After readings begin, place known weight on scale");
  Serial.println("Press + or a to increase calibration factor");
  Serial.println("Press - or z to decrease calibration factor");

  scale.begin(DOUT, CLK);
  scale.set_scale();
  scale.tare(); //Reset the scale to 0

  long zero_factor = scale.read_average(); //Get a baseline reading
  Serial.print("Zero factor: "); //This can be used to remove the need to tare the scale. Useful in permanent scale projects.
  Serial.println(zero_factor);
}

void loop() {

  scale.set_scale(calibration_factor); //Adjust to this calibration factor
  Serial.print("Reading: ");
  Serial.print(scale.get_units(), 1);
  Serial.print(" lbs"); //Change this to kg and re-adjust the calibration factor if you follow SI units like a sane person
  Serial.print(" calibration_factor: ");
```

```
  Serial.print(calibration_factor);
  Serial.println();

  if(Serial.available())
  {
    char temp = Serial.read();
    if(temp == '+' || temp == 'a')
      calibration_factor += 10;
    else if(temp == '-' || temp == 'z')
      calibration_factor -= 10;
  }
}
```

B.10. HX711lastcelle.ino

```
/*
 Example using the SparkFun HX711 breakout board with a scale
 By: Nathan Seidle
 SparkFun Electronics
 Date: November 19th, 2014
 License: This code is public domain but you buy me a beer if you use this
 ↪  and we meet someday (Beerware license).

 This example demonstrates basic scale output. See the calibration sketch to
 ↪  get the calibration_factor for your
 specific load cell setup.

 This example code uses bogde's excellent
 ↪  library:"https://github.com/bogde/HX711"
 bogde's library is released under a GNU GENERAL PUBLIC LICENSE

 The HX711 does one thing well: read load cells. The breakout board is
 ↪  compatible with any wheat-stone bridge
 based load cell which should allow a user to measure everything from a few
 ↪  grams to tens of tons.
 Arduino pin 2 -> HX711 CLK
```

```
 3 -> DAT
 5V -> VCC
 GND -> GND

 The HX711 board can be powered from 2.7V to 5V so the Arduino 5V power should be fine.

*/

#include "HX711.h"

#define calibration_factor -7050.0 //This value is obtained using the SparkFun_HX711_Calibration sketch

#define DOUT  3
#define CLK  2

HX711 scale;

void setup() {
  Serial.begin(9600);
  Serial.println("HX711 scale demo");

  scale.begin(DOUT, CLK);
  scale.set_scale(calibration_factor); //This value is obtained by using the SparkFun_HX711_Calibration sketch
  scale.tare(); //Assuming there is no weight on the scale at start up, reset the scale to 0

  Serial.println("Readings:");
}

void loop() {
  Serial.print("Reading: ");
  Serial.print(scale.get_units(), 1); //scale.get_units() returns a float
  Serial.print(" kg"); //You can change this to lbs but you'll need to refactor the calibration_factor
  Serial.println();
}
```

B.11. HX711libheader.h

```
/**
 *
 * HX711 library for Arduino
 * https://github.com/bogde/HX711
 *
 * MIT License
 * (c) 2018 Bogdan Necula
 *
**/
#ifndef HX711_h
#define HX711_h

#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

class HX711
{
        private:
                byte PD_SCK;        // Power Down and Serial Clock Input Pin
                byte DOUT;                // Serial Data Output Pin
                byte GAIN;                // amplification factor
                long OFFSET = 0;        // used for tare weight
                float SCALE = 1;        // used to return weight in grams,
                ↪  kg, ounces, whatever

        public:

                HX711();

                virtual ~HX711();

                // Initialize library with data output pin, clock input pin
                ↪  and gain factor.
                // Channel selection is made by passing the appropriate
                ↪  gain:
                // - With a gain factor of 64 or 128, channel A is selected
                // - With a gain factor of 32, channel B is selected
                // The library default is "128" (Channel A).
                void begin(byte dout, byte pd_sck, byte gain = 128);
```

```
40
41              // Check if HX711 is ready
42              // from the datasheet: When output data is not ready for
                ↪  retrieval, digital output pin DOUT is high. Serial clock
43              // input PD_SCK should be low. When DOUT goes to low, it
                ↪  indicates data is ready for retrieval.
44              bool is_ready();
45
46              // Wait for the HX711 to become ready
47              void wait_ready(unsigned long delay_ms = 0);
48              bool wait_ready_retry(int retries = 3, unsigned long
                ↪  delay_ms = 0);
49              bool wait_ready_timeout(unsigned long timeout = 1000,
                ↪  unsigned long delay_ms = 0);
50
51              // set the gain factor; takes effect only after a call to
                ↪  read()
52              // channel A can be set for a 128 or 64 gain; channel B has
                ↪  a fixed 32 gain
53              // depending on the parameter, the channel is also set to
                ↪  either A or B
54              void set_gain(byte gain = 128);
55
56              // waits for the chip to be ready and returns a reading
57              long read();
58
59              // returns an average reading; times = how many times to
                ↪  read
60              long read_average(byte times = 10);
61
62              // returns (read_average() - OFFSET), that is the current
                ↪  value without the tare weight; times = how many readings
                ↪  to do
63              double get_value(byte times = 1);
64
65              // returns get_value() divided by SCALE, that is the raw
                ↪  value divided by a value obtained via calibration
66              // times = how many readings to do
67              float get_units(byte times = 1);
68
69              // set the OFFSET value for tare weight; times = how many
                ↪  times to read the tare value
70              void tare(byte times = 10);
71
72              // set the SCALE value; this value is used to convert the
                ↪  raw data to "human readable" data (measure units)
```

```
                void set_scale(float scale = 1.f);

                // get the current SCALE
                float get_scale();

                // set OFFSET, the value that's subtracted from the actual
                ↪  reading (tare weight)
                void set_offset(long offset = 0);

                // get the current OFFSET
                long get_offset();

                // puts the chip into power down mode
                void power_down();

                // wakes up the chip after power down mode
                void power_up();
};

#endif /* HX711_h */
```

B.12. HX711libsource.cpp

```
/**
 *
 * HX711 library for Arduino
 * https://github.com/bogde/HX711
 *
 * MIT License
 * (c) 2018 Bogdan Necula
 *
**/
#include <Arduino.h>
#include "HX711.h"

// TEENSYDUINO has a port of Dean Camera's ATOMIC_BLOCK macros for AVR to
↪  ARM Cortex M3.
```

```
#define HAS_ATOMIC_BLOCK (defined(ARDUINO_ARCH_AVR) || defined(TEENSYDUINO))

// Whether we are running on either the ESP8266 or the ESP32.
#define ARCH_ESPRESSIF (defined(ARDUINO_ARCH_ESP8266) ||
↪  defined(ARDUINO_ARCH_ESP32))

// Whether we are actually running on FreeRTOS.
#define IS_FREE_RTOS defined(ARDUINO_ARCH_ESP32)

// Define macro designating whether we're running on a reasonable
// fast CPU and so should slow down sampling from GPIO.
#define FAST_CPU \
    ( \
    ARCH_ESPRESSIF || \
    defined(ARDUINO_ARCH_SAM)     || defined(ARDUINO_ARCH_SAMD) || \
    defined(ARDUINO_ARCH_STM32)   || defined(TEENSYDUINO) \
    )

#if HAS_ATOMIC_BLOCK
// Acquire AVR-specific ATOMIC_BLOCK(ATOMIC_RESTORESTATE) macro.
#include <util/atomic.h>
#endif

#if FAST_CPU
// Make shiftIn() be aware of clockspeed for
// faster CPUs like ESP32, Teensy 3.x and friends.
uint8_t shiftInSlow(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder) {
    uint8_t value = 0;
    uint8_t i;

    for(i = 0; i < 8; ++i) {
        digitalWrite(clockPin, HIGH);
        delayMicroseconds(1);
        if(bitOrder == LSBFIRST)
            value |= digitalRead(dataPin) << i;
        else
            value |= digitalRead(dataPin) << (7 - i);
        digitalWrite(clockPin, LOW);
        delayMicroseconds(1);
    }
    return value;
}
#define SHIFTIN_WITH_SPEED_SUPPORT(data,clock,order)
↪  shiftInSlow(data,clock,order)
#else
#define SHIFTIN_WITH_SPEED_SUPPORT(data,clock,order)
↪  shiftIn(data,clock,order)
```

```
#endif


HX711::HX711() {
}

HX711::~HX711() {
}

void HX711::begin(byte dout, byte pd_sck, byte gain) {
        PD_SCK = pd_sck;
        DOUT = dout;

        pinMode(PD_SCK, OUTPUT);
        pinMode(DOUT, INPUT_PULLUP);

        set_gain(gain);
}

bool HX711::is_ready() {
        return digitalRead(DOUT) == LOW;
}

void HX711::set_gain(byte gain) {
        switch (gain) {
                case 128:               // channel A, gain factor 128
                        GAIN = 1;
                        break;
                case 64:                // channel A, gain factor 64
                        GAIN = 3;
                        break;
                case 32:                // channel B, gain factor 32
                        GAIN = 2;
                        break;
        }

}

long HX711::read() {

        // Wait for the chip to become ready.
        wait_ready();

        // Define structures for reading data into.
        unsigned long value = 0;
        uint8_t data[3] = { 0 };
```

```
        uint8_t filler = 0x00;

        // Protect the read sequence from system interrupts.  If an interrupt occurs during
        // the time the PD_SCK signal is high it will stretch the length of the clock pulse.
        // If the total pulse time exceeds 60 uSec this will cause the HX711 to enter
        // power down mode during the middle of the read sequence.  While the device will
        // wake up when PD_SCK goes low again, the reset starts a new conversion cycle which
        // forces DOUT high until that cycle is completed.
        //
        // The result is that all subsequent bits read by shiftIn() will read back as 1,
        // corrupting the value returned by read().  The ATOMIC_BLOCK macro disables
        // interrupts during the sequence and then restores the interrupt mask to its previous
        // state after the sequence completes, insuring that the entire read-and-gain-set
        // sequence is not interrupted.  The macro has a few minor advantages over bracketing
        // the sequence between `noInterrupts()` and `interrupts()` calls.
        #if HAS_ATOMIC_BLOCK
        ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {

        #elif IS_FREE_RTOS
        // Begin of critical section.
        // Critical sections are used as a valid protection method
        // against simultaneous access in vanilla FreeRTOS.
        // Disable the scheduler and call portDISABLE_INTERRUPTS. This prevents
        // context switches and servicing of ISRs during a critical section.
        portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
        portENTER_CRITICAL(&mux);

        #else
        // Disable interrupts.
        noInterrupts();
        #endif

        // Pulse the clock pin 24 times to read the data.
        data[2] = SHIFTIN_WITH_SPEED_SUPPORT(DOUT, PD_SCK, MSBFIRST);
        data[1] = SHIFTIN_WITH_SPEED_SUPPORT(DOUT, PD_SCK, MSBFIRST);
```

```
        data[0] = SHIFTIN_WITH_SPEED_SUPPORT(DOUT, PD_SCK, MSBFIRST);

        // Set the channel and the gain factor for the next reading using
        ↪  the clock pin.
        for (unsigned int i = 0; i < GAIN; i++) {
                digitalWrite(PD_SCK, HIGH);
                #if ARCH_ESPRESSIF
                delayMicroseconds(1);
                #endif
                digitalWrite(PD_SCK, LOW);
                #if ARCH_ESPRESSIF
                delayMicroseconds(1);
                #endif
        }

        #if IS_FREE_RTOS
        // End of critical section.
        portEXIT_CRITICAL(&mux);

        #elif HAS_ATOMIC_BLOCK
        }

        #else
        // Enable interrupts again.
        interrupts();
        #endif

        // Replicate the most significant bit to pad out a 32-bit signed
        ↪  integer
        if (data[2] & 0x80) {
                filler = 0xFF;
        } else {
                filler = 0x00;
        }

        // Construct a 32-bit signed integer
        value = ( static_cast<unsigned long>(filler) << 24
                        | static_cast<unsigned long>(data[2]) << 16
                        | static_cast<unsigned long>(data[1]) << 8
                        | static_cast<unsigned long>(data[0]) );

        return static_cast<long>(value);
}

void HX711::wait_ready(unsigned long delay_ms) {
        // Wait for the chip to become ready.
```

```
183         // This is a blocking implementation and will
184         // halt the sketch until a load cell is connected.
185         while (!is_ready()) {
186                 // Probably will do no harm on AVR but will feed the
                    ↪  Watchdog Timer (WDT) on ESP.
187                 // https://github.com/bogde/HX711/issues/73
188                 delay(delay_ms);
189         }
190 }
191
192 bool HX711::wait_ready_retry(int retries, unsigned long delay_ms) {
193         // Wait for the chip to become ready by
194         // retrying for a specified amount of attempts.
195         // https://github.com/bogde/HX711/issues/76
196         int count = 0;
197         while (count < retries) {
198                 if (is_ready()) {
199                         return true;
200                 }
201                 delay(delay_ms);
202                 count++;
203         }
204         return false;
205 }
206
207 bool HX711::wait_ready_timeout(unsigned long timeout, unsigned long
    ↪  delay_ms) {
208         // Wait for the chip to become ready until timeout.
209         // https://github.com/bogde/HX711/pull/96
210         unsigned long millisStarted = millis();
211         while (millis() - millisStarted < timeout) {
212                 if (is_ready()) {
213                         return true;
214                 }
215                 delay(delay_ms);
216         }
217         return false;
218 }
219
220 long HX711::read_average(byte times) {
221         long sum = 0;
222         for (byte i = 0; i < times; i++) {
223                 sum += read();
224                 // Probably will do no harm on AVR but will feed the
                    ↪  Watchdog Timer (WDT) on ESP.
225                 // https://github.com/bogde/HX711/issues/73
```

```
                delay(0);
        }
        return sum / times;
}

double HX711::get_value(byte times) {
        return read_average(times) - OFFSET;
}

float HX711::get_units(byte times) {
        return get_value(times) / SCALE;
}

void HX711::tare(byte times) {
        double sum = read_average(times);
        set_offset(sum);
}

void HX711::set_scale(float scale) {
        SCALE = scale;
}

float HX711::get_scale() {
        return SCALE;
}

void HX711::set_offset(long offset) {
        OFFSET = offset;
}

long HX711::get_offset() {
        return OFFSET;
}

void HX711::power_down() {
        digitalWrite(PD_SCK, LOW);
        digitalWrite(PD_SCK, HIGH);
}

void HX711::power_up() {
        digitalWrite(PD_SCK, LOW);
}
```

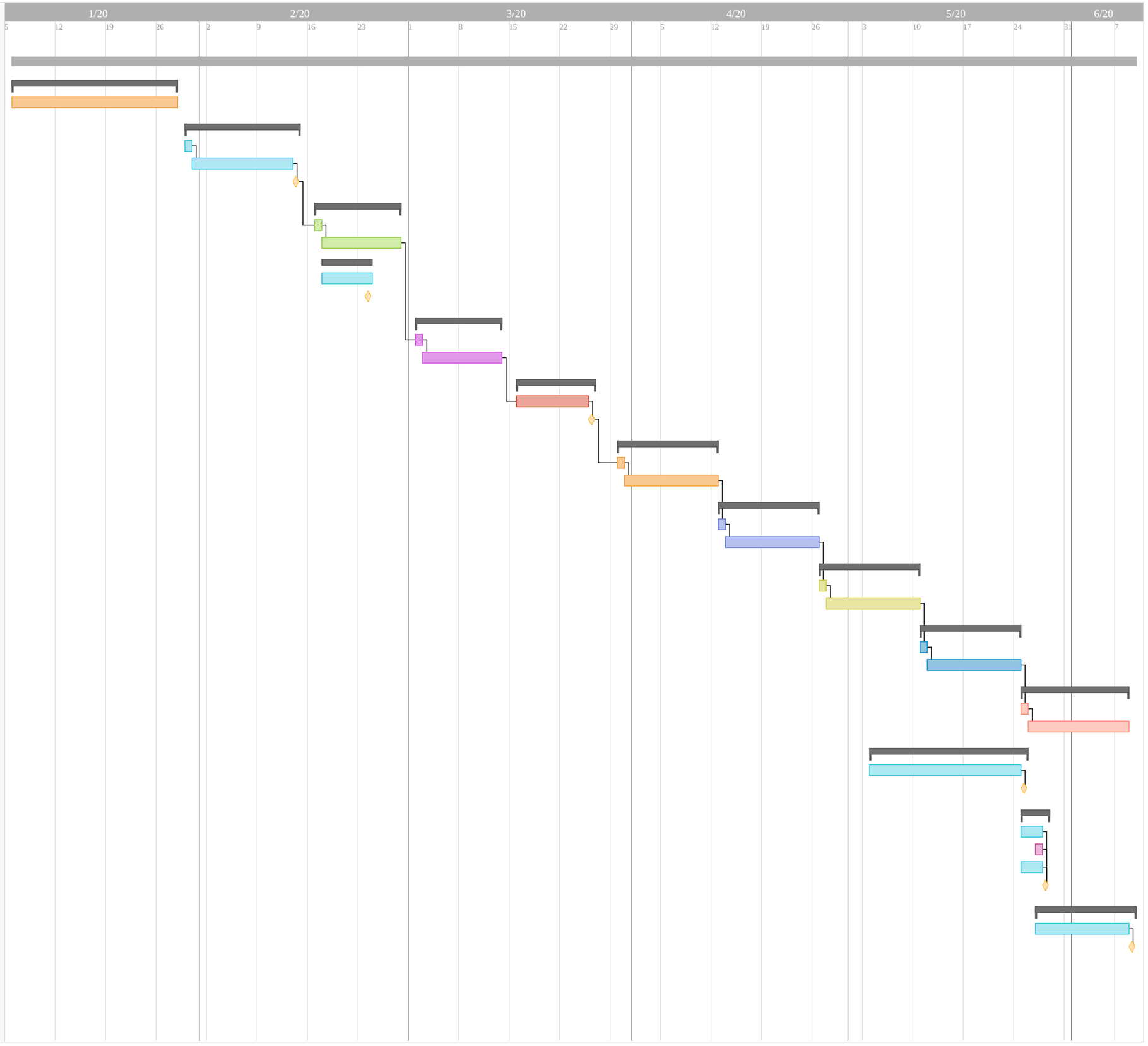
B.13. HX711konvertering.ino

```
#include "HX711-multi.h"
#define CLK A0 // clock pin to the load cell
#define N 3 // # of channels
#define DOUT1 A1 // data pin to the 1st load cell lift 1
#define DOUT2 A2 // data pin to the 2nd load cell lift 2
#define DOUT3 A3 // data pin to the 3rd load cell drag
#define TARE_TIMEOUT_SECONDS 4
// INSERT THE VALUES CORRESPONDING TO YOUR MODEL
// Inputs
double x01 = 0.02; // distance between the leading edge and the first load cell
[m]
double v = 10; // speed of the wind [m/s]
double rho = 1.225; // density [kg/m^3]
double S = 0.04; // wing surface [m^2]
double C = 0.2; // chord length [m]
//CALLIBRATION OF THE BALANCE
// To re-calibrate the balance (if a load cell is replaced or one of the existing
changes its behavior) write a 1.0 in the corresponding cell of measurements[]
// vector and tare the balance (this can be done sending any character to the
Arduino through Serial Monitor).
// Then read the value of 'WEIGHT DATA BELOW' corresponding to the load cell at
the program when a known weight is placed at the load cell.
// Substitute the 1.0 by (value obtained)/(known weight [g]).
float measurements[N] = {(-10000/21.2), -8600/21.2, 40750/21.2}; // measured
SCALES parameters
// MAIN CODE, DO NOT SAVE THE CHANGES MADE HERE
// Init
byte DOUTS[N] = {DOUT1, DOUT2, DOUT3};
long int results[N];
double weights[N];
double go = 9.80662;
double lift;
double drag;
double Mle;
double F1;
double F2;
double x02 = 0.041;
```

```
double x12 = x01 + x02;
double xcp;
double Cl;
double Cd;
double Cm;
HX711MULTI scales(N, DOUTS, CLK);
void setup() {
 Serial.begin(115200);
 Serial.flush();

 tare();
 scales.set_scales(measurements);
}
void tare() {
   bool tareSuccessful = false;
 unsigned long tareStartTime = millis();
 while (!tareSuccessful && millis() < (tareStartTime + TARE_TIMEOUT_SECONDS
* 1000)) {
 tareSuccessful = scales.tare(20, 10000); //reject 'tare' if still
ringing
 }
}
// void sendRawData() {
// scales.read(results);
// for (int i = 0; i < scales.get_count(); ++i) {
// Serial.print(-results[i]);
// Serial.print((i != scales.get_count() - 1) ? "\t" : "\n");
// }
// delay(1000);
//}
void sendWeightData() {
 scales.get_units(weights);
 for (int i = 0; i < scales.get_count(); ++i) {
 Serial.print(-weights[i]);
 Serial.print((i != scales.get_count() - 1) ? "\t" : "\n");
 }
 delay(1000);
}
void loop() {
// Serial.println("RAW DATA BELOW");
// sendRawData(); // this is for sending raw data, for where everything else
is done in processing
// Serial.println();
 Serial.println("WEIGHT DATA BELOW");
 sendWeightData();
 Serial.println();
```

```
 //calculate forces in N

F1 = weights[0]*go/1000; //[N]
F2 = weights[1]*go/1000; //[N]
lift = F1 + F2;
drag = (weights[2])*go/1000; //[N]
Mle = x01 * F1 + x02 *F2; // [Nm]
xcp = (x02 * F2 + x01 * F2) / (F1 + F2)*1000;
 //aerodynamic coefficients

Cl = (lift*2)/(rho*v*v*S);
Cd = (drag*2)/(rho*v*v*S);
Cm = (Mle*2)/(rho*v*v*S*C);
Serial.print("lift = ");
Serial.print(lift);
Serial.println(" [N]");
}

Serial.print("drag = ");
Serial.print(drag);
Serial.println(" [N]"); // REPRESENT IT IN MILINEWTONS FOR MORE PRECISSION
Serial.print("momentum at the leading edge = ");
Serial.print(Mle);
Serial.println(" [Nm]");
Serial.print("position of the centre of pressure: ");
Serial.print(xcp);
Serial.println(" [mm]");
Serial.println();
Serial.println("Aerodynamic coefficients");
Serial.print("Cl = ");
Serial.println(Cl);
Serial.print("Cd = ");
Serial.println(Cd);
Serial.print("Cm = ");
Serial.println(Cm);
Serial.println();
// On serial data (any data), re-tare
if (Serial.available() > 0) {
 while (Serial.available()) {
 Serial.read();
 }
 tare();
}
}
```

C. Gantt-diagram



Bachelor 2020-09	start	end
Forarbeid	**06/01/20**	**28/01/20**
Forarbeid	06/01	28/01
Sprint 1	**30/01/20**	**14/02/20**
Planlegging	30/01	30/01
Sprint	31/01	13/02
Presentasjon 1	14/02	14/02
Sprint 2 - Tekniske løsninger	**17/02/20**	**28/02/20**
Review og planlegging	17/02	17/02
Sprint	18/02	28/02
Krav	**18/02/20**	**24/02/20**
Krav	18/02	24/02
Krav godkjent	24/02	24/02
Sprint 3	**02/03/20**	**13/03/20**
Review og planlegging	02/03	02/03
Sprint	03/03	13/03
Rapport 2 og Presentasjon	**16/03/20**	**26/03/20**
Sprint	16/03	25/03
Presentasjon 2	26/03	26/03
Sprint 4 Programmering, Modellering, og Simul...	**30/03/20**	**12/04/20**
Review og planlegging	30/03	30/03
Sprint	31/03	12/04
Sprint 5 Begynne å ferdigstille programvare, si...	**13/04/20**	**26/04/20**
Review og planlegging	13/04	13/04
Sprint	14/04	26/04
Sprint 6 Endelig fullføring av programvare, sim...	**27/04/20**	**10/05/20**
Review og planlegging	27/04	27/04
Sprint	28/04	10/05
Sprint 7 Rapportfullføring	**11/05/20**	**24/05/20**
Review og planlegging	11/05	11/05
Gjennomføring av rapport	12/05	24/05
Sprint 8 Presentasjon	**25/05/20**	**08/06/20**
Review og planlegging	25/05	25/05
Lage presentasjonen	26/05	08/06
Ferdigstilling av rapport	**04/05/20**	**25/05/20**
Rapportarbeid	04/05	24/05
Rapportinnlevering	25/05	25/05
Expo	**25/05/20**	**28/05/20**
Lage Expo-plakat	25/05	27/05
Rigging (Avlyst)	27/05	27/05
Lage Expo-webside	25/05	27/05
Expo	28/05	28/05
Fullføring av sluttpresentasjon	**27/05/20**	**09/06/20**
Arbeid	27/05	08/06
3. Presentasjon	09/06	09/06

D. Budsjett for konsepter

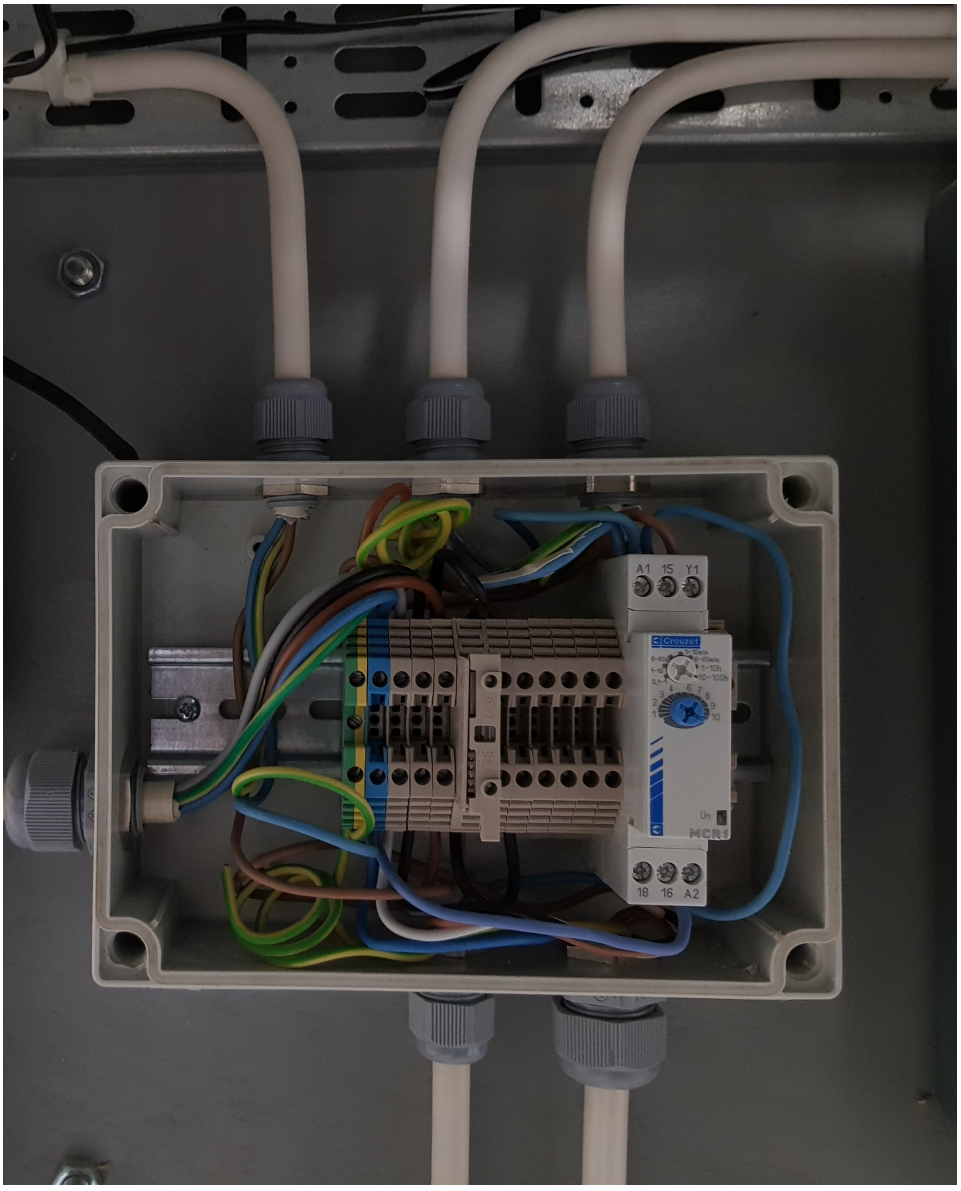
Konsepter budsjett								
	Konsept 1			**Konsept 2**			**Konsept 3**	
Ant.:	**Produkt:**	**Totalpris:**	**Ant.:**	**Produkt:**	**Pris:**	**Ant.**	**Produkt:**	**Pris:**
8	Hjul	640.00 kr	8	Hjul	640.00 kr	8	Hjul	640.00 kr
1	Lufthastighetsmåler	364.52 kr	1	Lufthastighetsmåler	364.52 kr	1	Lufthastighetsmåler	364.52 kr
2	Lastcelle	160.00 kr	2	Lastcelle	160.00 kr	2	Lastcelle	160.00 kr
	Modellfeste	500.00 kr		Modellfeste	500.00 kr		Modellfeste	500.00 kr
1	Mekanisk bryter testkammer	468.00 kr	2	Magnetisk bryter testkammer	564.00 kr	2	Magnetisk bryter	564.00 kr
2	Lukelås	456.00 kr	2	Lukelås	456.00 kr	2	Lukelås	456.00 kr
2	Tilt sensor arduino	35.00 kr	2	Tilt sensor arduino	35.00 kr	2	Tilt sensor arduino	35.00 kr
1	Røykveske	200.00 kr	1	Røykveske	200.00 kr	1	Røykveske	200.00 kr
1	Røykmaskin	500.00 kr	1	Røykmaskin	500.00 kr	1	Røykmaskin	500.00 kr
	Bolter (til innfesting) med muttere	300.00 kr		Bolter (til innfesting) med muttere	300.00 kr		Bolter (til innfesting) med muttere	300.00 kr
1	LCD-skjerm	150.00 kr	1	Touchskjerm	900.00 kr	1	Touchskjerm	900.00 kr
1	Lufttemperatur + luftfuktighetssensor	60.00 kr	1	Lufttemperatur + luftfuktighetssensor	60.00 kr	1	Lufttemperatur + luftfuktighetssensor	60.00 kr
	Start/stopp knapp	0.00 kr	1	Vri bryter av/på	140.00 kr	1	Vri bryter av/på	140.00 kr
				Honeycomb	2,000.00 kr		Honeycomb	2,000.00 kr
			1	1k motstander	21.25 kr	1	1k motstander	21.25 kr
			1	Stepmotor (pitchjustering)	54.00 kr	1	Stepmotor (pitchjustering)	54.00 kr
				Lufthasighetsskanner (grovt) anslag	1,500.00 kr		Lufthasighetsskanner (grovt) anslag	1,500.00 kr
			1	5-12V power supply	295.00 kr	1	5-12V power supply	295.00 kr
				Dyse	1,000.00 kr		Dyse	1,000.00 kr
				Diffuser	1,000.00 kr		Diffuser	1,000.00 kr
			2	Screens	1,000.00 kr	2	Screens	1,000.00 kr
			1	Kondensator 100	1.59 kr	1	Kondensator 100	1.59 kr
				Testkammer	500.00 kr		Testkammer	500.00 kr
				Innebygd datamas	1,350.00 kr		Innebygd datamas	1,350.00 kr
				Powerbank og utstyr for IDM	200.00 kr		Powerbank og utstyr for IDM	200.00 kr
			1	LM358 OP-amp	3.29 kr	1	LM358 OP-amp	3.29 kr
			2	10k motstander	17.86 kr	2	10k motstander	17.86 kr
						1	Vifte	10,000.00 kr
	Totalsum:	**3,533.52 k**		**Totalsum:**	**13,462.51 k**		**Totalsum:**	**23,462.51 kr**

Figur D.1.: Budsjett for konsepter

E. Bilde av vindtunnelen

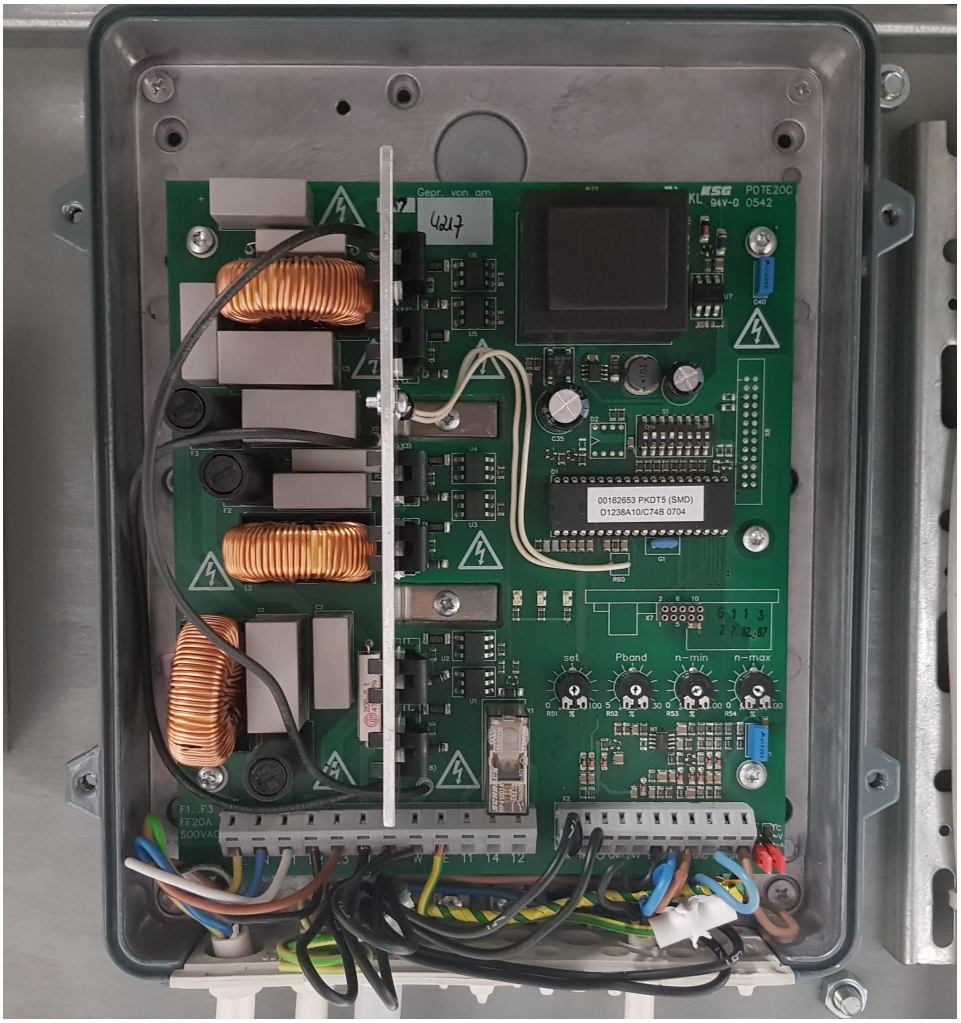
Figur E.1.: Vindtunnelen fra Vitenskapssenteret

F. Bilde av rekkeklemmer



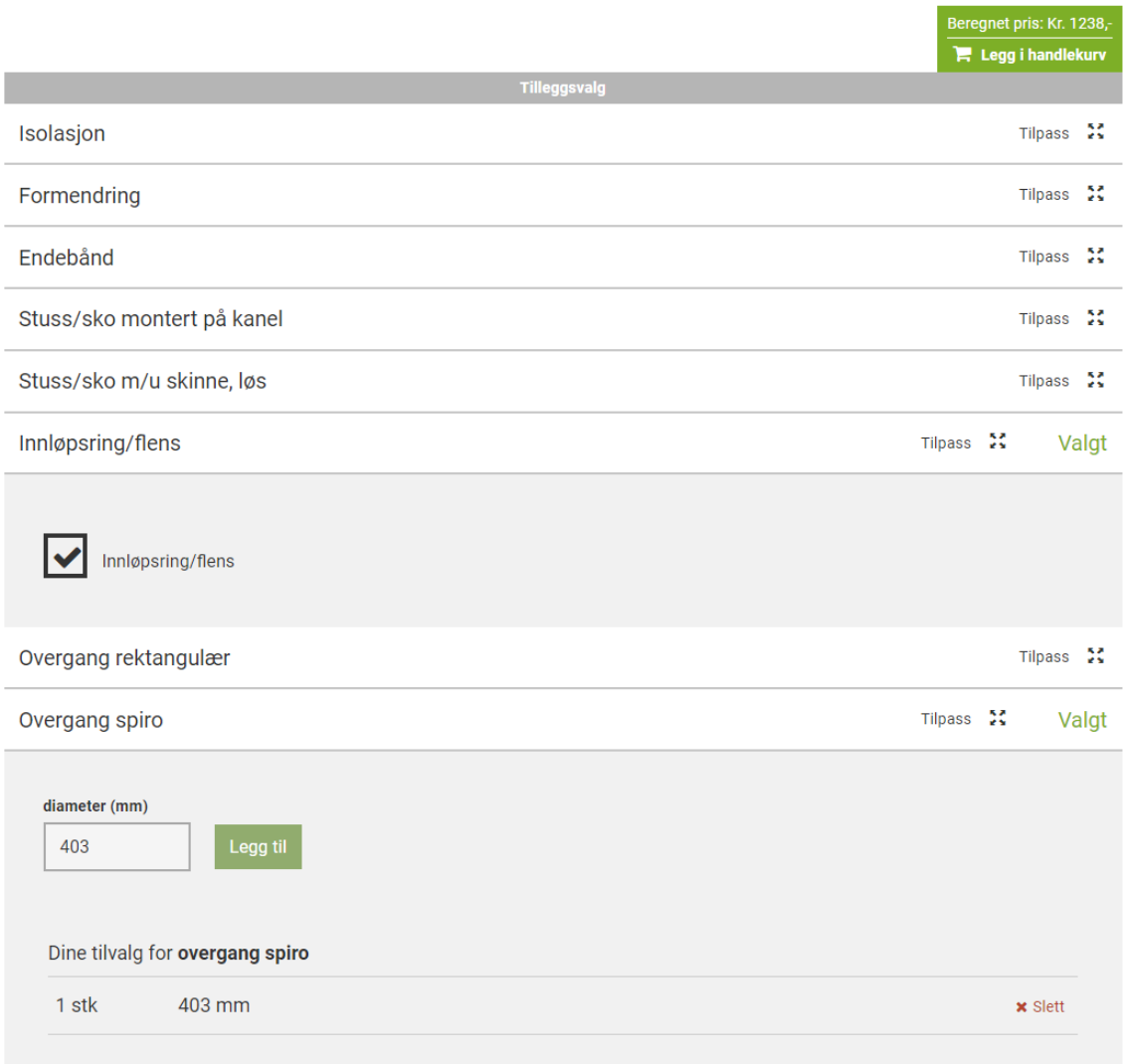
Figur F.1.: Rekkeklemmenene til det elektriske delsystemet

G. Bilde av motordriver kortet

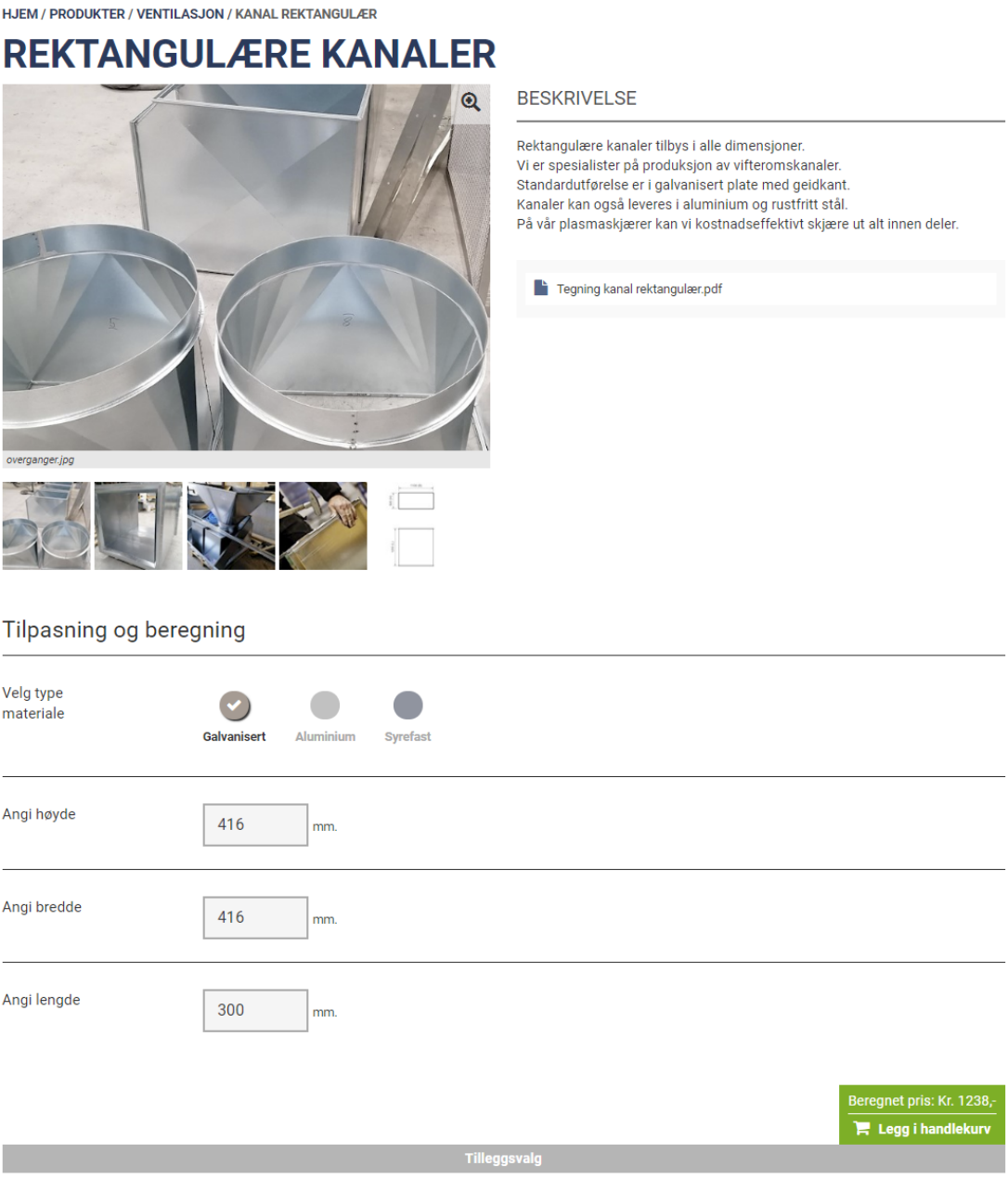


Figur G.1.: Motordriver kortet til det elektriske delsystemet

H. Priskalkulator for komponent til overgang av tverrsnitt



Figur H.1.: Skjermbilde 1 av priskalkulator.



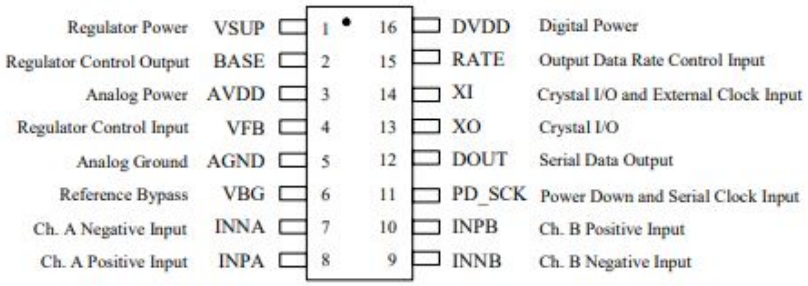
Figur H.2.: Skjermbilde 2 av priskalkulator.

I. Dht11 datablad

Parameters	Conditions	Minimum	Typical	Maximum
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			±1%RH	
Accuracy	25℃		±4%RH	
	0-50℃			±5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0℃	30%RH		90%RH
	25℃	20%RH		90%RH
	50℃	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25℃, 1m/s Air	6 S	10 S	15 S
Hysteresis			±1%RH	
Long-Term Stability	Typical		±1%RH/year	
Temperature				
Resolution		1℃	1℃	1℃
		8 Bit	8 Bit	8 Bit
Repeatability			±1℃	
Accuracy		±1℃		±2℃
Measurement Range		0℃		50℃
Response Time (Seconds)	1/e(63%)	6 S		30 S

Figur I.1.: Utdrag fra databladet til DHT11 sensoren [23].

J. Lastcelle blad



SOP-16L Package

Pin #	Name	Function	Description
1	VSUP	Power	Regulator supply: 2.7 ~ 5.5V
2	BASE	Analog Output	Regulator control output（NC when not used）
3	AVDD	Power	Analog supply: 2.6 ~ 5.5V
4	VFB	Analog Input	Regulator control input（connect to AGND when not used）
5	AGND	Ground	Analog Ground
6	VBG	Analog Output	Reference bypass output
7	INA-	Analog Input	Channel A negative input
8	INA+	Analog Input	Channel A positive input
9	INB-	Analog Input	Channel B negative input
10	INB+	Analog Input	Channel B positive input
11	PD_SCK	Digital Input	Power down control (high active) and serial clock input
12	DOUT	Digital Output	Serial data output
13	XO	Digital I/O	Crystal I/O (NC when not used）
14	XI	Digital Input	Crystal I/O or external clock input, 0: use on-chip oscillator
15	RATE	Digital Input	Output data rate control, 0: 10Hz; 1: 80Hz
16	DVDD	Power	Digital supply: 2.6 ~ 5.5V

Figur J.1.: PIN-beskrivelse for HX711 kortet [29].

Parameter	Notes	MIN	TYP	MAX	UNIT
Full scale differential input range	V(inp)-V(inn)		±0.5(AVDD/GAIN)		V
Common mode input		AGND+1.2		AVDD-1.3	V
Output data rate	Internal Oscillator, RATE = 0		10		Hz
	Internal Oscillator, RATE = DVDD		80		
	Crystal or external clock, RATE = 0		f_{clk}/1,105,920		
	Crystal or external clock, RATE = DVDD		f_{clk}/138,240		
Output data coding	2's complement	800000		7FFFFF	HEX
Output settling time (1)	RATE = 0		400		ms
	RATE = DVDD		50		
Input offset drift	Gain = 128		0.2		mV
	Gain = 64		0.4		
Input noise	Gain = 128， RATE = 0		50		nV(rms)
	Gain = 128， RATE = DVDD		90		
Temperature drift	Input offset（Gain = 128）		±6		nV/℃
	Gain（Gain = 128）		±5		ppm/℃
Input common mode rejection	Gain = 128， RATE = 0		100		dB
Power supply rejection	Gain = 128， RATE = 0		100		dB
Reference bypass （V_{BG}）			1.25		V
Crystal or external clock frequency		1	11.0592	20	MHz
Power supply voltage	DVDD	2.6		5.5	V
	AVDD，VSUP	2.6		5.5	
Analog supply current (including regulator)	Normal		1400		μA
	Power down		0.3		
Digital supply current	Normal		100		μA
	Power down		0.2		

Figur J.2.: Operative egenskaper til HX711 kortet [29].

K. LM358 Elektriske Egenskaper

PARAMETER		TEST CONDITIONS	LM358			LM2904			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	
Input Offset Voltage		See (2), T_A = 25°C		2	7		2	7	mV
Input Bias Current		$I_{IN(+)}$ or $I_{IN(-)}$, T_A = 25°C, V_{CM} = 0 V, See (3)		45	250		45	250	nA
Input Offset Current		$I_{IN(+)} - I_{IN(-)}$, V_{CM} = 0 V, T_A = 25°C		5	50		5	50	nA
Input Common-Mode Voltage Range		V^+ = 30 V, See (4) (LM2904, V^+ = 26 V), T_A = 25°C	0		V^+−1.5	0		V^+−1.5	V
Supply Current		Over Full Temperature Range							
		R_L = ∞ on All Op Amps							
		V^+ = 30 V (LM2904 V^+ = 26 V)		1	2		1	2	mA
		V^+ = 5 V		0.5	1.2		0.5	1.2	mA
Large Signal Voltage		V^+ = 15V, T_A = 25°C,							
Gain		R_L ≥ 2 kΩ, (For V_O = 1 V to 11 V)	25	100		25	100		V/mV
Common-Mode Rejection Ratio		T_A = 25°C,	65	85		50	70		dB
		V_{CM} = 0 V to V^+−1.5 V							
Power Supply Rejection Ratio		V^+ = 5 V to 30 V	65	100		50	100		dB
		(LM2904, V^+ = 5 V to 26 V), T_A = 25°C							
Amplifier-to-Amplifier Coupling		f = 1 kHz to 20 kHz, T_A = 25°C (Input Referred), See (5)		−120			−120		dB
Output Current	Source	V_{IN}^+ = 1 V,	20	40		20	40		mA
		V_{IN}^- = 0 V,							
		V^+ = 15 V,							
		V_O = 2 V, T_A = 25°C							
	Sink	V_{IN}^- = 1 V, V_{IN}^+ = 0 V	10	20		10	20		mA
		V^+ = 15V, T_A = 25°C,							
		V_O = 2 V							
		V_{IN}^- = 1 V,	12	50		12	50		µA
		V_{IN}^+ = 0 V							
		T_A = 25°C, V_O = 200 mV,							
		V^+ = 15 V							
Short Circuit to Ground		T_A = 25°C, See (6), V^+ = 15 V		40	60		40	60	mA
Input Offset Voltage		See (2)			9			10	mV
Input Offset Voltage Drift		R_S = 0 Ω		7			7		µV/°C
Input Offset Current		$I_{IN(+)} - I_{IN(-)}$			150		45	200	nA
Input Offset Current Drift		R_S = 0 Ω		10			10		pA/°C
Input Bias Current		$I_{IN(+)}$ or $I_{IN(-)}$		40	500		40	500	nA

Figur K.1.: Elektriske egenskaper for LM358 [14].

L. Datablad for vindtunnelvifte

DE

Movement by Perfection

The Royal League in ventilation, control and drive technology

Product documentation

Type
FC040-4DQ.2F.A7

Article number
102984

ZIEHL-ABEGG

1. Product specification

Technical data

Article number	102984
Type	FC040-4DQ.2F.A7
Designation	Axial fan with diecast blades
Rated values	3~230/400V+16/-10% D/Y 50Hz P_1 310W 1.07/ 0.62A 1350/min COSY 0,77 3~230/400V+16/-10% D/Y 60Hz P_1 430W 1550/min COSY 0,83
Electrical connection	Supply cable lateral-diagonal 9x 0,5 mm2, 145 cm
Min. operating temperature °C	-40
Cable quality	Li4G4G-J
Type of protection	IP54
Thermal class	THCL155
Connection diagram	1360-106XB
Rating plate	1x fixed
Fitting position	H/Vu/Vo
Motor protection	thermal contact
Impregnation	Moisture and hot climate protection
Quality of bearings	ball bearing with long-time lubrication
Material Rotor	Aluminium
Painting rotor	unpainted
Material blades	Aluminium
Painting impeller	unpainted
Field of application	Standard application
Painting housing	Bell mouth unpainted
Painting mot.suspens	Motor suspension powder-coated consistency class 2
colour suspension	RAL 9005 (jet black)
Weight kg	9,30
ErP Data	Efficiency η_{statA}: 31,7 % Efficiency grade: N_{actual} = 41,7 / N_{target} = 40* *ErP 2015

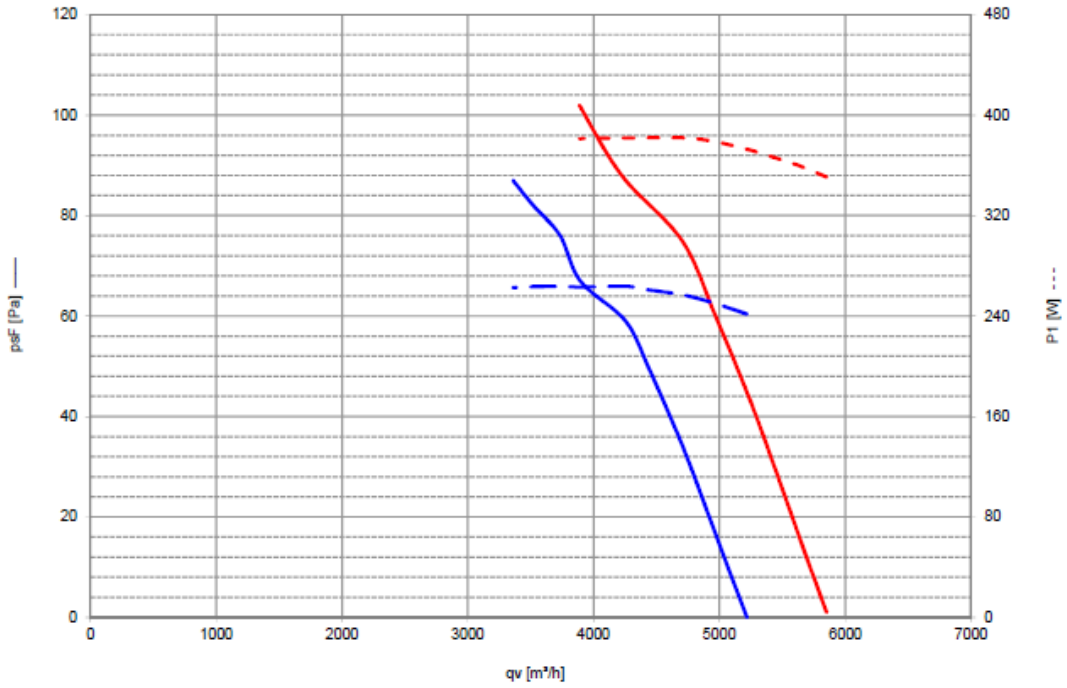
2. Characteristic curve

Beschreibung / Description
Typ: FC040-4DQ.2F.A7
3~ 230/400V +16/-10% D/Y 50Hz P1 310W
1,07/0,62A 1350/MIN COSY 0,77
3~ 230/400V +27% D/Y 60Hz P1 430W
1,3/0,75A 1550/MIN COSY 0,83
IP54 THCL 155

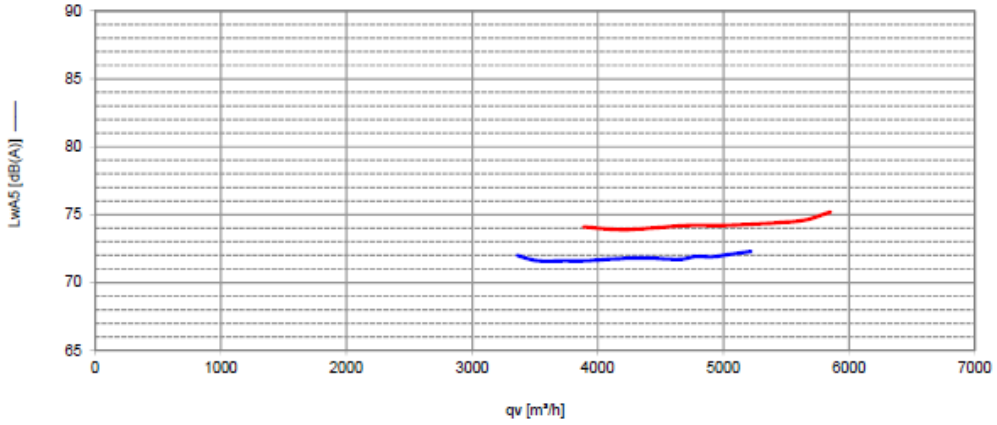
Messaufbau / Assembling:
Ventilator montiert in Volldüse ohne Berührschutzgitter.
Fan measured in full bell mouth without guard grille.

Legende / Legend
A) 3~ 230/400V 50Hz D/Y [ID 97314]
C) 3~ 230/400V 60Hz D/Y [ID 84723]
Gemessen mit üblichen Toleranzen / Measured with normal tolerances

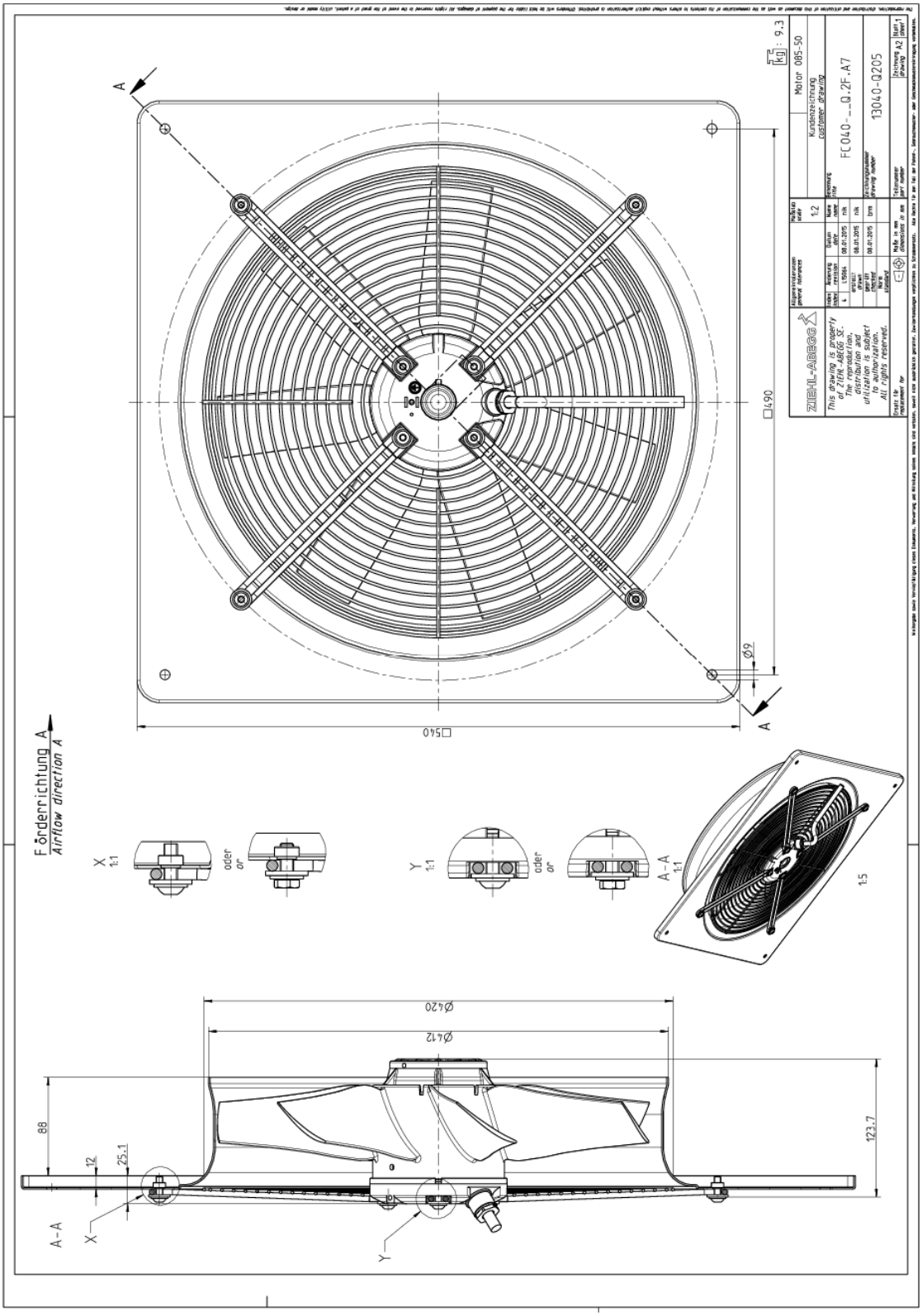
1. Diagramm / Chart : Volumenstrom - Druckerhöhung - elektr. Leistungsaufnahme / Airflow - Pressure - Electr. Power Input



2. Diagramm / Chart : Volumenstrom - Akustik / Airflow - Acoustics



3. Drawing



Dimensions in mm
Shown drawing is just to show the dimensions of the fan.

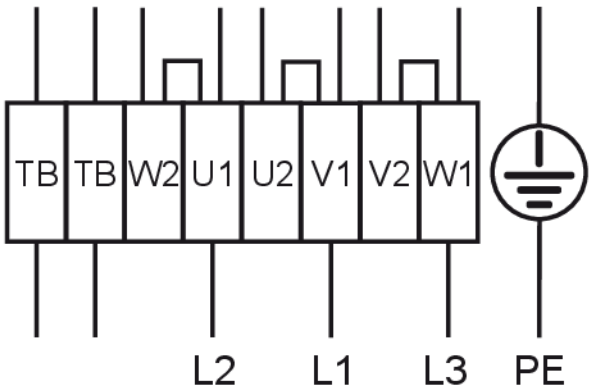
4. Connection diagram

1360-106XB

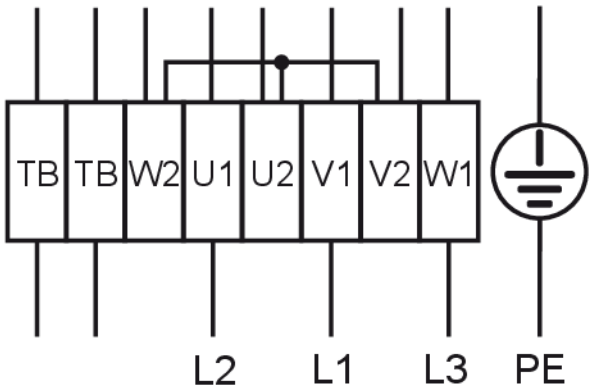
3~ motor with one speed and thermostatic switch (if built in).

U1	brown
V1	blue
W1	black
U2	red
V2	grey
W2	orange
TB	white

Δ-connection



Y-connection



5. EU-Declaration of conformity

EC-Declaration of conformity

- Translation -
(english)

ZA75-GB 1616 Index 011
00285644-GB

Manufacturer: **ZIEHL-ABEGG SE**
Heinz-Ziehl-Straße
74653 Künzelsau
Germany

The manufacturer is solely responsible for issuance of the declaration of conformity.

The products:

- External rotor motor MK.., MW..
- Axial fan FA.., FB.., FC.., FE.., FF.., FS.., FT.., FH.., FL.., FN.., FV.., DN.., VR.., VN.., ZC.., ZF.., ZN..
- Centrifugal fan RA.., RD.., RE.., RF.., RG.., RH.., RK.., RM.., RR.., RZ.., GR.., ER..
- Cross-flow fan QK.., QR.., QT.., QG..

The engine type:

- Asynchronous internal or external rotor motor
- Asynchronous internal or external rotor motor with integrated frequency inverter, except EDP system
- Electronically commutated (EC) internal or external rotor motor
- Electronically commutated internal or external rotor motor with integrated EC Controller, except EDP system

These products comply with the following EU directives:

- EMC Directive 2014/30/EU
- Low voltage directive 2014/35/EU
- ErP Directive 2009/125/EC, in conjunction with Regulation (EU) no. 327/2011

The following harmonized standards are in use:

EN 60034-1:2010 + Cor.:2010
EN 60204-1:2006 + A1:2009 + AC:2010
EN 60529:1991 + A1:2000 + A2:2013
EN 61000-6-3:2007 + A1:2011 + AC:2012
EN 61000-6-2:2005 + AC:2005

Compliance with the ErP Directive 2009/125/EC does not refer to external rotor motors MK.., MW..

All ErP-relevant information comprises measurements which are determined using a standardised measurement set-up. More details can be obtained from the manufacturer.

Compliance with the EMC Directive 2014/30 / EU refers only to those products when they are connected by mounting / operating instructions . If these products are integrated into a system or supplemented with other components (eg. sensing controls) and operated , the manufacturer or operator is responsible of the overall system for compliance with the EMC Directive 2014/30 / EU .

Künzelsau, 20.04.2016
(Location, date of issue)

ZIEHL-ABEGG SE
Dr. W. Angelis
Technical Director Air Movement Division
(Name, Function)

i.V. W. Angelis

(Signature)

ZIEHL-ABEGG

M. Gruppekontrakt

Gruppekontrakt for Bachelorgruppe 2020-09

Kristoffer Andersen Kristian Auestad Marius Balsvik
Steffen Barskrind Håvard Gaska Joachim Haug

24. mai 2020

1 Introduksjon

Dette er kontrakten som vi gruppe-meldemmer kommer til og forholde oss til under bacheloroppgaven. Ved å ha en kontrakt setter vi grunnregler for gruppen samt forholde oss til et regelverk dersom problemer eller konflikter mellom gruppemedlemmer skulle oppstå.

2 Gruppe etikette

1. Dersom et gruppemedlem er syk eller forsinket til møte, skal vedkommende melde fra i god tid slik at resten ikke venter på vedkommende.
2. Personen som har kommet for sent etter 20 minutter samt uten grunn, straffen skal gå på å kjøpe noe til min 15 og maks 25 kr pr pers eller noe samlet.
3. Alle skal få si sine meninger i gruppen.
4. Om et medlem har problemer i gruppen kan det bli tatt opp med gruppeleder eller i plenum.
5. Problemer eller andre tanker/konflikter skal gå igjennom gruppeleder om det ikke ønskes å bli tatt i plenum.
6. Alle gruppemedlemmer skal holde seg i en god hygienisk stand.
7. Gruppemedlem som er mer enn 15 minutter for sen uten god grunn eller forvarsel skal spandere snacks på gruppa.
8. Om reglene i kontrakten blir brutt, skal gruppa bestemme straff om ikke annet er oppført i kontrakten.
9. Gruppa skal ha det gøy i løpet av prosjektet i og utenfor arbeidet.
10. Innleveringsfrister skal bli holdt innen avtalt tid, og gruppemedlemmer har selv ansvar om å melde fra hvis de ikke rekker fristen eller eventuelt trenger hjelp.

Figur M.1.: Gruppekontrakt

Bibliografi

[1] Arifuzzaman og Mohammad Mashud. *Design Construction and Performance Test of a Low Cost Subsonic Wind Tunnel*.

[2] Jewel B. Barlow, Jr William H. Rae og Alan Pope. *Low–speed wind tunnel testing*. 3rd. John Wiley & Sons, Incorporated, 1999.

[3] Brett Beauregard. *PID Library*.

[4] Metha R. D Bell J. H. *CONTRACTION DESIGN FOR SMALL LOW-SPEED WIND TUNNELS*. Apr. 1988.

[5] Bogde. *Loadcell Library*.

[6] William Bolton. *Mechatronics. Electronic Control Systems in Mechanical and Electrical Engineering*. 6th. Pearson, 2015.

[7] Thomas L. Floyd. *Electronic Devices. Conventional Current Version*. 10th. Pearson, 2018.

[8] Python Software Foundation. *Python documentation*. Versjon 3.7.7. URL: `https://docs.python.org/3.7/`.

[9] Gene F. Franklin, J. David Powell og Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. 7th. Pearson, 2014.

[10] Bjørn Gjevik. *INNFØRING I FLUIDMEKANIKK Forelesninger og øvelser i MPTF-2200*. HBV, Kongsberg, 2017.

[11] Joshua Hrisko. *Arduino Pitot Tube Wind Speed and Airspeed Indicator - Theory and Experiments*.

[12] HT Sensor Technology CO., Limited. *TAL220. Parallel Beam Load Cell*.

[13] Ihsan Y. Hussain mfl. *DESIGN, CONSTRUCTION AND TESTING OF LOW SPEED WIND TUNNEL WITH ITS MEASUREMENT AND INSPECTION DEVICES.*

[14] Texas Instruments. *LMx58-N. Low-Power, Dual-Operational Amplifiers*. Des. 2014.

[15] Vinayak Kulkarni, Niranjan Sahoo og Sandip D. Chavan. «Simulation of honeycomb–screen combinations for turbulence management in a subsonic wind tunnel». I: *Journal of Wind Engineering and Industrial Aerodynamics* 99.1 (jan. 2011), s. 37–45. DOI: 10.1016/j.jweia.2010.10.006.

[16] Cesareo de La Rosa Siqueira. *Vortex Shedding GIF*.

[17] Chris Liechti. *pySerial*.

[18] Makuna. *Arduino Library for RTC, Ds1302, Ds1307, Ds3231, & Ds3234 with deep support*.

[19] MathWorks. *Step response*.

[20] Stefano Mauro mfl. «Small-Scale Open-Circuit Wind Tunnel: Design Criteria, Construction and Calibration». I: *International Journal of Applied Engineering Research* 12 (des. 2017), s. 13649–13662.

[21] R. D. Mehta og P. Bradshaw. «Design rules for small low speed wind tunnels». I: *The Aeronautical Journal of the Royal Aeronautical Society* 83 (827 nov. 1979), s. 443–453. DOI: 10.1017/S0001924000031985.

[22] R.D. Mehta og P.H. Hoffmann. «Boundary layer two-dimensionality in wind tunnels». I: *Experiments in fluids* (5 sep. 1987), s. 358–360.

[23] Mouser. *DHT11. Humidity & Temperature Sensor*.

[24] Nettvett.no. *Eksempel på risikoanalyse for bedrift*. 2019.

[25] Justin D. Pereira. *Wind Tunnels: Aerodynamics, Models and Experiments (Engineering Tools, Techniques and Tables)*. Nova Science Publishers, Inc., 2013. ISBN: 9781612092041.

[26] James Scheiman. «Considerations for the Installation of Honeycomb and Screens To Reduce Wind-Tunnel Turbulence». I: *NASA Technical Memorandum 81868* (1981).

[27] Ken Schwaber og Jeff Sutherland. *The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game*. 2017.

[28] Scrum.org. *The Scrum Framework Poster*. 2016.

[29] AVIA Semiconductor. *HX711. 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales*. 2.0.

[30] NXP Semiconductors. *MPXV7002. Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated*. Rev. 4. Mar. 2017.

[31] Mahmood Shafiee og Fateme Dinmohammadi. *An FMEA-Based Risk Assessment Approach for Wind Turbine Systems: A Comparative Study of Onshore and Offshore*. 2014.

[32] NASA on The Commons.

[33] Itziar Bueno Tintoré. *DESIGN OF A THREE-AXIS WIND TUNNEL FORCE BALANCE*. 2018.

[34] Engineering Toolbox. *Hydraulic Diameter*. 2003.

[35] Dara Trent. *Strain Gauge Load Cell Basics*.

Ordforklaring

AC	Vekselstrøm
ADC	Analog til Digital konverter
Anemometer	Sensor for vindmåling
Bode-diagram	Diagram som brukes i frekvensanalyse
C	Programmeringsspråk
CFD	Computational Fluid Dynamics
DAC	Digital til Analog konverter
FEM	Finite Element Method
GUI	Graphical User Interface
IDM	Innebygd datamaskin
Mikrokontroller	Liten datamaskin som kontrollerer et system
OP-amp	Operasjonsforsterker
PCB	Printed Circuit Board
PID	Proporsjonal-Integrasjon-Derivasjon algoritme
PWM	Puls-bredde modulasjon

SBC	Single-board computer
Termistor	Elektrisk komponent som endrer motstand ved temperaturforandring
Testmodell	Den fysiske modellen som plasseres i testkammeret til vindtunnelen som det skal gjøres eksperimenter på
Transferfunksjon	Matematisk funksjon for analyse av kontrollsystemer
USN	Universitetet i Sør-Øst Norge

Figurer

Tabeller