UNIVERSITETET I AGDER

Utvikling av Banestyring for Hydraulisk Offshorekran

Tor-André Rettedal Hetland

Konrad Johan Jensen

Veiledere:
David Alireza Anisi, UiA
Petter Semb, NOVN

Masteroppgaven er gjennomført som ledd i utdanningen ved Universitetet i Agder og er godkjent som del av denne utdanningen. Denne godkjenningen innebærer ikke at universitetet innestår for de metoder som er anvendt og de konklusjoner som er trukket.

Universitetet i Agder
Fakultetet for teknologi og realfag
Institutt for ingeniørvitenskap
19. mai 2015

Sammendrag

I dag styres NOVN sine kraner med manuelt pådrag til hver aktuator, eller med aksestyring som styrer krantuppen i rette linjer i rommet. Begge disse metodene krever en erfaren kranoperatør for å effektivt utføre løfteoperasjoner. Motivasjonen til denne oppgaven var å effektivisere disse løfteoperasjonene. Håndtering av rør er for det meste en repetitiv oppgave som med fordel kan automatiseres. Det er strenge krav om sikkerhet og presisjon samtidig som man ønsker at operasjonen utføres raskest mulig. Løsningen som presenteres i denne rapporten tar steget videre mot automatisering av løfteoperasjoner, nemlig banestyring. Banestyring vil utføre prosessen effektivt og forutsigbart, dette vil lette arbeidet til kranoperatøren betydelig. Kranoperatørens oppgave blir å overse prosessen og ta over på de delene av prosessen hvor det ikke er hensiktsmessig å bruke banestyring.

Hovedmålet i denne oppgaven var å utvikle banestyring for NOVN sine rørhåndteringskraner og deretter implementere dette på en PLS. Banestyringen som ble utviklet i denne oppgaven gjør det mulig for en kran å følge en bane basert på en liste med punkter i rommet. Systemet ble laget for å enkelt kunne implementeres i en mer omfattende systemarkitektur som tillater mer kompliserte oppgaver. Systemet ble implementert på en Siemens PLS og hovedsaklig testet i en HiL-simulering. HiL-simuleringen viser at systemet tilfredsstiller kravene for presisjon som er satt i NOVN sin teststandard. Det ble i tillegg utført en test på en ny rørhåndteringskran på NOVN sitt verksted i Søgne. Den fysiske testen viste at banestyringen virker i praksis. Kranen fulgte banen tett under både HiL-simulering og fysisk test.

Banestyringen som har blitt utviklet i denne oppgaven baserer seg på rette linjer i aktuatorlengder og vil finne raskere vei mellom to punkter. Dette vil gjøre kjøringen mer effektiv og stabil. Alle krav og ønsker satt av problemeier ble tilfredstilt. Banestyringen har blitt implementert i dagens system og kan brukes som et alternativ til manuell styring og aksestyring.

Forord

Denne rapporten er den avsluttende delen av masterutdanningen i Mekatronikk MSc ved Universitetet i Agder, med fagkode MAS-500. Oppgaven er tildelt Tor-André Rettedal Hetland og Konrad Johan Jensen, av National Oilwell Varcco Norway (NOVN). Oppgaven har tittelen "Utvikling av Banestyring for Hydraulisk Offshorekran", og går ut på å videreutvikle banestyring for en kran samt komme med en konsept som kan testes for videre implementering. Arbeidet ble for det meste gjort i mekatronikklabben på Universitet i Agder - Grimstad, men fysisk testing har blitt gjort ved NOVN sine lokaler i Kristiansand og Søgne.

Vi ønsker å takke NOVN for den spennende og lærerik oppgaven samt å ha stilt med nødvendig utstyr. En takk går til Per Øyvind Strandmyr-Ødegaard, Petter Semb og Kjell Arne Øvland som har vært våre kontaktpersoner på NOVN og har hjulpet med administrative og tekniske spørsmål.

Førsteamanuensis David Alireza Anisi har vært vår veileder og vi vil takke han for god rettledning og tilbakemelding underveis i prosjektet. Vi vil også takke resten av foreleserne på Mekatronikkmasteren for deres hjelp.

Tor-André Rettedal Hetland og Konrad Johan Jensen
Mai 2015

Innhold

1. Introduksjon

Dette kapittelet vil introdusere problemstillingen og gi bakgrunnsinformasjon og motivasjon. Det vil virke som et oppslagsverk for resten av oppgaven da løsningen knyttes opp mot problemstilling. Kapittelet anbefales for lesere som ikke er kjent med oppgaven fra før eller ikke er kjent med terminologien brukt om rørhåndteringskraner.

1.1 Bakgrunn

Kontinuerlig forbedring av sikkerhet og prosesser er to viktige mål i offshoreindustrien. I perioder med lav investeringsvilje vil smarte løsninger kunne gjøre store utslag for å få en kontrakt eller ikke. Hovedformålet med denne oppgaven er å videreutvikle banestyring til NOVN sine rørhåndteringskraner. Dagens kranoperatør kan styre kranen på to forskjellige måter, manuell styring og aksestyring. Ved manuell styring blir hver aktuator styrt separat mens aksestyring gir operatøren mulighet til å styre krantuppen etter et *globalt xyz*-koordinatsystem definert med origo midt i pidestallen. Store deler av arbeidet en rørhåndteringskran utfører er i stor grad repetitive og kan med fordel automatiseres. Kranførerens jobb er krevende og skiftene er korte for å opprettholde konsentrasjonen som er nødvendig. Ved å implementere banestyring vil operatøren overse prosessen istedenfor å utføre den. Banestyring vil si at kranen beveger seg etter en gitt bane uten hjelp av operatøren. Kranføreren vil overta styringen på deler av prosessen som ikke er hensiktsmessig å kjøre ved hjelp av banestyring. Denne rapporten vil bygge videre på tidligere arbeid gjort av gruppe masterstudenter ved UiA [1]. Det foreligger 3D-modell og MATLAB/Siumlink-modell for test av banestyring. De utviklet et simuleringsverktøy for testing av forskjellige kontrollstrategier. Implementering av banestyring i PLS-kode og testing av konsept ved simulering vil være hovedfokuset i denne rapporten.

1.2 Problemdefinisjon

Hovedmålet for oppgaven er å implementere banestyring for en rørhåndteringskran på en Programmerbar Logisk Styring (PLS). Krantuppen skal følge en gitt bane i rommet. Per dags dato utføres all styring av rørhåndteringskraner manuelt eller med aksestyring av en operatør. NOVN har ønske om å videreføre det teoretiske arbeidet gjort innen banestyring og ta neste steg mot praktisk implementasjon.

Oppgaven vil basere seg på tidligere arbeid av masterstudenter ved UiA i samarbeid med NOVN [1]. Eksisterende materiell tilgjengelig for kandidatene er:

- Tidsserier logget i eksisterende kraner for å utvikle transferfunksjon mellom ventilpådrag og kranbevegelser
- 3D-modeller av NOVN sine kraner
- Mekaniske og hydrauliske modeller modellert i SimulationX

- Programkode for kontroll av kranen, i form av en mal for en rørhåndteringskran
- Reguleringsstrategier for banestyring
- Kinematiske modeller

Resultatet vil være en komplett programkode i Siemens Step7 som kan benyttes på en fysisk kran. Delmålene er som følger:

- Implementere kranens kinematikk på en PLS skrevet i Structured Control Language (SCL)
- Implementere en kaskaderegulator skrevet i SCL
- Lage en banegenerator basert på en liste med punkter
- Innregulere kontrolleren for å følge en predefinert bane

De overnevnte delmålene vil være hovedfokus i denne rapporten da de er nødvendige for å få banestyring til å fungere. Disse delmålene kan tenkes som krav fra NOVN. NOVN har også ytret ønske om å undersøke et ytterligere tema. Følgende delmål vil bli gjort dersom tiden tillater det.

- Analysere hvordan sensorstøy og ikke-ideelle vinkelsensorer påvirker systemet og deretter utbedre dette ved hjelp av filterteknologi.
- Lage en avansert og dynamisk banegenerator som tar høyde for kranens hastighetsbegrensninger
- Utvikle en algoritme eller metode for optimal innregulering av kontroller.

Disse delmålene vil hjelpe videreutviklingen av banestyringen utover det som er fastsatt i denne masteroppgaven.

1.3 Motivasjon

Ny teknologi og nye funksjoner er viktig for å forbedre NOVN sine eksisterende produkter. Ved å utvikle banestyring for rørhåndteringskraner vil man ta neste steg mot en automatisk prosess i oljebransjen. Ved repeterende oppgaver vil en automatisering gi store tidsbesparelser, som er etterspurt av NOVN sine kunder. Automatisering vil gjøre arbeidet til operatørene som styrer kranene lettere og tryggere. Operatøren vil ivareta sikkerheten ved å overse prosessen med muligheten til å manuelt overta kontrollen. Hvis løsningen fra denne oppgaven skal ha en reel verdi for NOVN må den kunne implementeres på flere krantyper. Muligheten til å bruke løsningen på flere krantyper vil gjøre det lettere å implementere den som standard. I tillegg til skalering mellom krantyper bør løsningen være portabel til andre plattformer. Ved å skrive koden i Structured Control Language (SCL) vil selve kjernen av programmet være kompakt og fremtidsrettet for videre implementering i en mer avansert system arkitektur.

1.4 Problemløsning

Kinematikken til kranen er gitt i denne oppgaven, men den er skrevet med matrise og vektornotasjon, dette støttes ikke i SCL. For å få kinematikken inn på PLS må den skrives om fra matriseform til algebraiske ligninger.

Det er også gitt forslag til kontrollstrategier fra tidligere masteroppgave, men dette vil ikke danne hovedgrunnlaget for løsningen som presenteres, da disse benytter tilbakemelding på den deriverte vinkelmålingen som er påvirket av mye støy. Dersom delmålet om å rette problemet

med støy på sensormålinger ikke blir nådd, vil fremdeles løsningen som presenteres fungere da den ikke bruker vinkelhastighet som tilbakekobling. Strategien utarbeidet i denne oppgaven bruker ventilposisjon til å estimere sylinderfart. Dette vil bli dekket i mer detalj i kapittel 3. SCL kode til PLSen vil basere seg på blokkdiagram av utviklet kontrollstrategi.

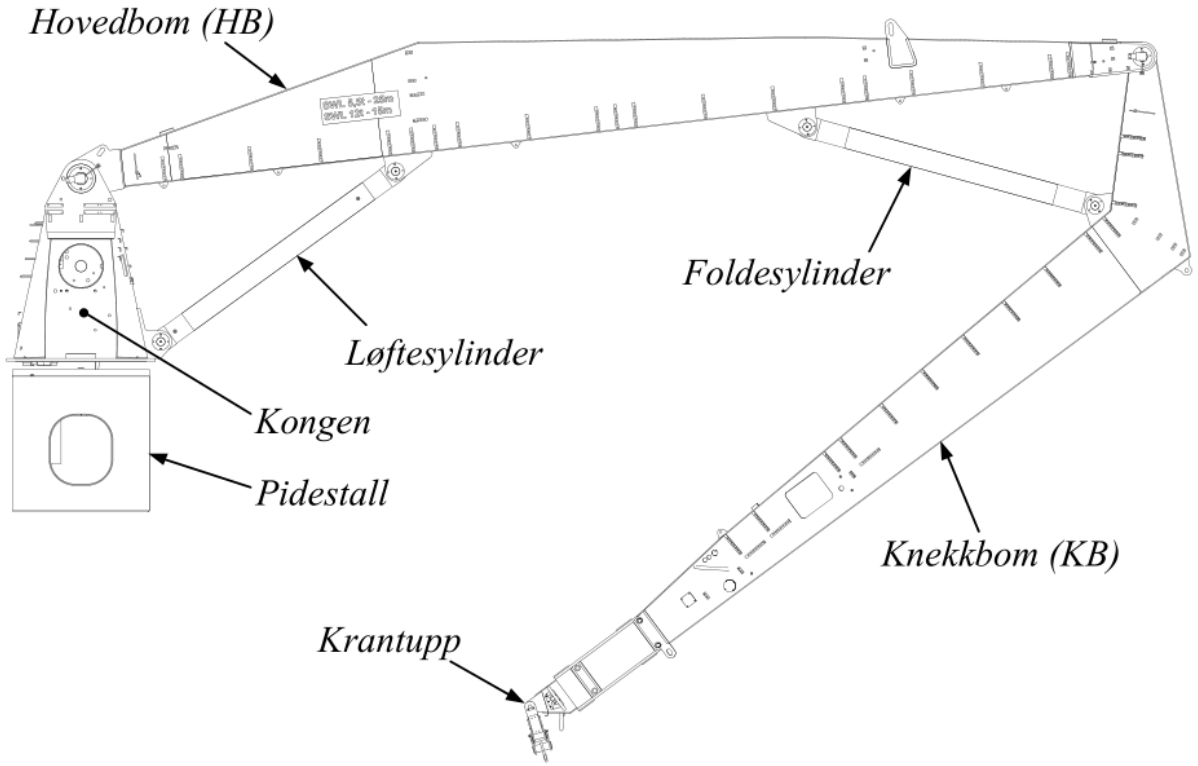
Det vil settes opp en Hardware-In-The-Loop (HiL) simulering for å teste PLS systemet. Da vil LABVIEW Real-Time Target brukes til å kommunisere mellom modellen og PLSen. Selv om en dynamisk hydromekanisk modell er gitt i denne oppgaven må den bearbeides før den kan brukes i HiL. Modellen tok lang tid å simulere, dette gjør den uegnet til å kjøre i HiL da den må kunne kjøre i sanntid. Den eksisterende modellen er dog et veldig godt utgangspunkt for modellen som kan eksporteres og brukes som Real-time target i HiL-simuleringen.

Når alt er implementert i PLSen og dynamisk modell fungerer må systemet innreguleres. Det vil bruke flere forskjellige metoder for innregulering, hovedsakelig i MATLAB. Hvis tiden tillater det vil det bli brukt optimaliseringsalgoritmer til å innregulere kontrollparameterne. MATLAB sine innebygde optimaliseringsalgoritmer vil benyttes. Hver forsterkningsfaktor i kaskadekontrolleren vil settes som en designvariabel.

MATLAB/Simulink, SimulationX, Simatic Step7 og LabVIEW RealTime vil bli brukt for å løse de problemene som er definert. MATLAB/Simulink har mulighet for å teste, analysere og optimere både instrumentering, programkode og kontrollsystem som kommer til å bli utviklet. LabVIEW RealTime vil bli benyttet for å kjøre en Hardware-in-the-Loop (HIL) simulering i sanntid. Denne løsningen ble brukt fordi det finnes et sterkt faglig miljø for bruk av disse programmene på UiA. I denne simuleringen vil det inngå en modell som er laget i SimulationX. PLSen vil bli programmert med Simatic Step7, og vil også inngå i HiL-simuleringen.

1.5 Rørhåndteringkranens oppbygging

Dette delkapittelet er til for å gi en oversikt kranens geometri og terminologi rundt dens komponenter, dette ble allerede dekket i masteroppgaven fra 2013 [1] så mesteparten av innholdet i dette kapittelet er hentet fra deres rapport. Rørhåndteringskranen som brukes i denne oppgaven er en knekkbomkran med utforming som vist i figur 1.1. Denne utformingen og geometrien er populær fordi den tillater kranoperatøren å føre krantuppen tett mot lasten som gjør det enkelt å bruke spesialverktøy for håndtering av rør og *risere*. I tillegg kan kranen benyttes for ordinære løfteoperasjoner med vaier. Muligheten for å kjøre krantuppen tett opp mot lasten medfører minimal pendelbevegelse av hengende last og dermed mindre risiko for personell og utstyr.



Figur 1.1: Terminologi for kranens mekaniske komponenter

NOVN har en flere forskjellige knekkbomkraner, alle disse er konstruert på lignende måte, det er bare forskjellige geometriske forhold. Så lenge geometrien er ligner vil det være mulig å utvikle banestyring som passer alle knekkbomkraner. Selve kranen står på en pidestall som er fastmontert på en plattform eller et skipsdekk. Kongen er festet i pidestallen med en stor svingkrans. Kongen, samt hele kranen svinger ved hjelp av hydrauliske motorer som sitter montert i kongen og virker mot svingkransen i pidestallen. Videre er hovedbommen (HB) festet til kongen med et rotasjonsledd, og blir aktuert av en hydraulisk løftesylinder. Knekkbommen (KB) er så festet i enden av HB med et nytt rotasjonsledd. Denne er aktuert av en hydraulisk foldesylinder. På enden av KB er krantuppen, hvor verktøy for rørhåndtering kan monteres. Knekkbomkranen i denne oppgaven har tre vinkelmålere i tillegg til tilbakemelding på de tre ventilposisjonene. En vinkelmåler er montert i svingkransen til kongen for å gi tilbakemelding på kranens svingvinkel. En vinkelmåler er montert i leddet mellom kongen og HB, for å gi tilbakemelding på løftvinkelen av HB. Den siste vinkelmåleren er montert i leddet mellom HB og KB, for å gi tilbakemelding på foldevinkelen.

1.6 Rapportstruktur

Kapittel 2 vil gi det teoretiske grunnlaget for resten av rapporten. Teorien bak arbeidet som er gjort i dette prosjektet vil være dekket i dette kapittelet. Innholdet vil være rettet mot teori som er relevant for oppgaven, men ikke inneholde spesifikke detaljer.

Kapittel 3 vil inneholde spesifikke detaljer rundt metode og den faktiske løsningen. Dette kapittelet vil bygge videre på teorien fra kapittel 2, men vil være direkte i hvordan det har blitt brukt i oppgaven.

Kapittel 4 er diskusjonskapittelet hvor resultatene blir diskutert. Implikasjonene av resultatene

vil også bli diskutert samt forslag til videre utvikling. Hensikten med dette kapittelet er å vurdere problemstillingen og krav opp mot løsningen.

Kapittel 5 er konklusjonen, i dette kapittelet vil det forklares om løsningen faktisk løste problemet. En kort beskrivelse problemstillingen, løsningen, resultatene og forslag til videre utvikling vil inkluderes i dette kapittelet.

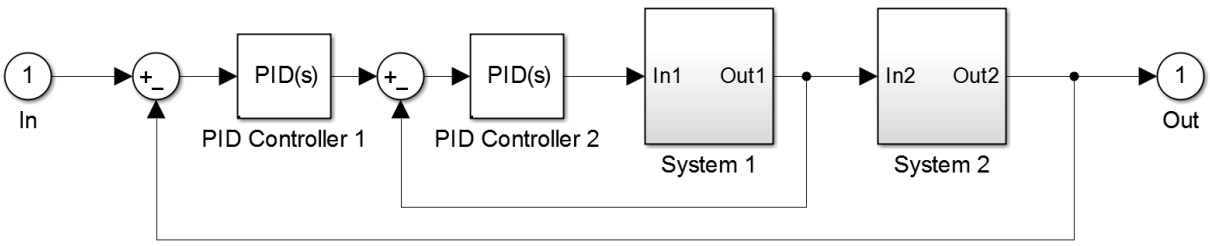
2. Teoretisk Bakgrunn

Dette kapittelet skal gi en teoretisk innføring i de emnene som er blitt benyttet i dette prosjektet. Emnene i kapittelet vil holdes generelt, men rettet mot de løsningene som har blitt brukt i kapittel 3. Terminologi som brukes videre i Resultater, Diskusjon og Konklusjon vil introduseres her. Kapittelet anbefales for lesere som ikke er godt kjent med modellbasert design eller ønsker detaljer rundt løsningen som presenteres i kapittel 3.

2.1 Kontrollere

I dag er det vanlig å bruke PID-kontrollere til å regulere systemer og prosesser i alle typer industrier. En stor andel av disse PID-kontrollerne bruker kun P og I for å regulere prosessen. Dette er fordi D-leddet er lett påvirket av støy. Det vil også benyttes en PI-kontroller i dette systemet for å unngå problemer med støy. En PI-kontroller vil gi null stasjonært avvik, men ytelsen kan bli noe redusert i forhold til en PID-kontroller. Innreguleringen vil dog bli betydelig enklere da det kun er to parametere som skal settes.

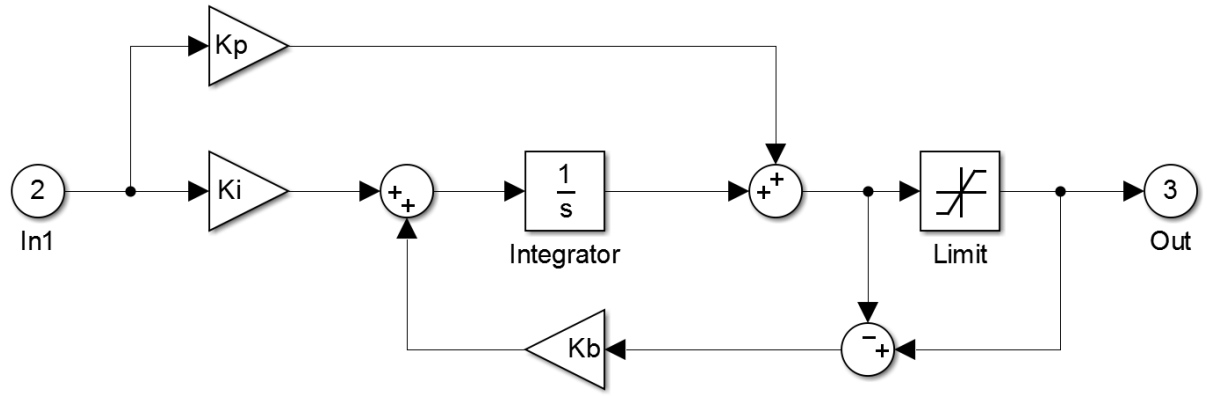
I prosesser hvor flere målesignaler er tilgjengelig, er det også vanlig å bruke en kaskadestruktur med en indre og ytre sløyfe. Dette er spesielt gunstig i prosesser hvor den indre sløyfen reagerer raskere på kontrollsignalet enn den ytre sløyfen. Et eksempel på en kaskadestruktur er vist i figur 2.1.



Figur 2.1: Eksempel på en kaskadestruktur

Målet med å bruke en slik kaskadestruktur er å få en rask indre sløyfe som har tilstrekkelig båndbredde slik at den kan kompensere for eventuelle forstyrelser som oppstår [2, s. 276].

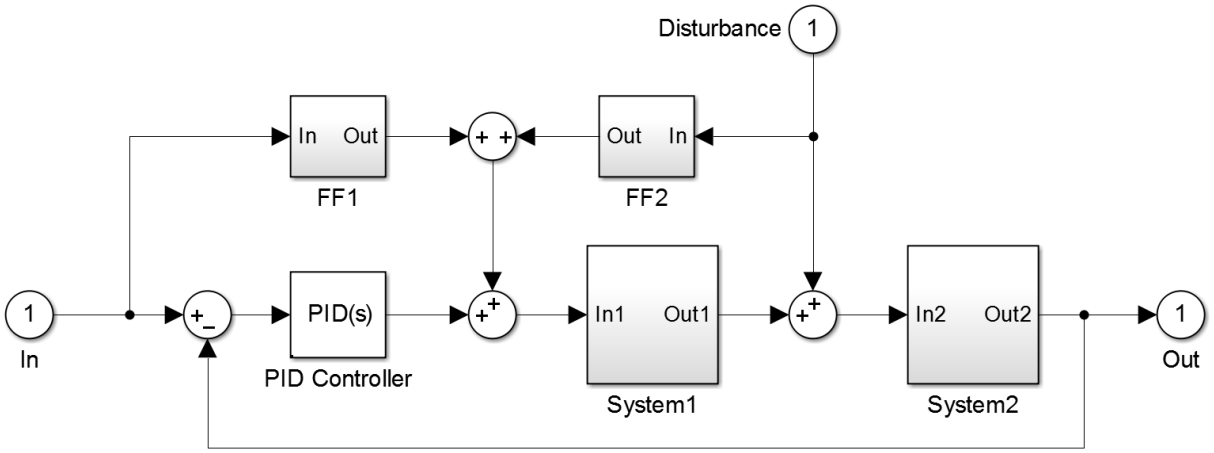
I prosesser hvor aktuatorene kan gå i metning, møter man en ny utfordring med PID-kontrollere. Når aktuatoren går i metning så vil integral-leddet fortsette å integrere feilsignalet. Det bygger seg da opp en stor verdi i integratoren. Dette problemet blir kalt *integrator windup*. Metoder som begrenser eller eliminerer dette problemet blir kalt for *anti windup*. I denne oppgaven brukes en metode som kalles *back calculation anti windup*, vist under i figur 2.2.



Figur 2.2: Eksempel på PI-kontroller med anti windup

Denne metoden måler differansen i signalet før og etter metningen, og legger dette til før integreringen. Dette vil begrense integralet i kontrolleren, og man unngår da problemet med *integrator windup*. Forsterkningsfaktoren K_b blir som regel satt til 1 [2, s. 83].

En annen metode som kan benyttes til å forbedre kontrolleren, er å implementere en fremoverkobling. Det finnes to måter å implementere fremoverkoblingen på. Den ene måten er å måle forstyrrelser som påvirker systemet og gi et signal basert på denne målingen. Dette vil redusere påvirkningen av forstyrrelsene på systemet. Den andre måten er å bruke en fremoverkobling på settpunktet til kontrolleren. Dette vil blant annet gjøre systemet raskere når settpunktet endrer seg. Fordelen med fremoverkoblingen er lik i begge situasjoner. Når man kun benytter tilbakekobling i kontrolleren så er man avhengig av at det har oppstått en feil før man kan rette den opp. Ved å bruke fremoverkobling kan man eliminere feilen før den har oppstått [2, s. 286]. Fremoverkoblingen kan både være en transferfunksjon eller en forsterkning. Et eksempel på fremoverkobling vises i figur 2.3.

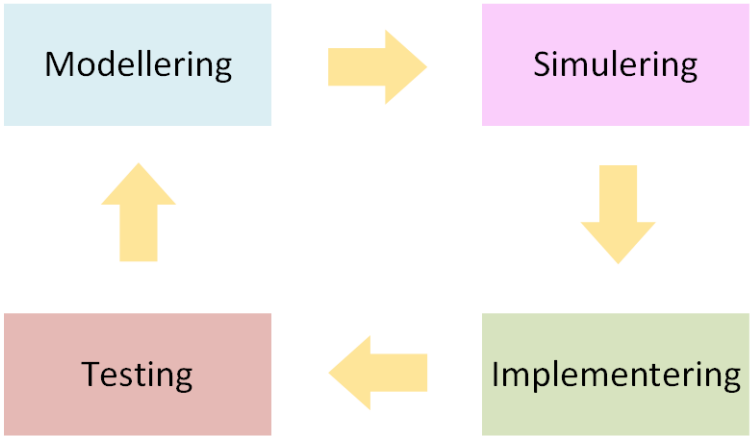


Figur 2.3: Eksempel på en Fremoverkobling

I dette eksempelet benyttes både fremoverkobling på settpunktet og på de målbare forstyrrelsene i systemet.

2.2 Modellbasert design

Modellbasert design er en arbeidsmetode for utvikling og design av industrielle applikasjoner, gjerne fokusert mot reguleringsteknikk, signalbehandling og programvareutvikling. Mye av fordelen går ut på at man bruker en modell av det fysiske systemet, og kan kjøre tester uten tilgang til det fysiske systemet. Figur 2.4 viser et eksempel på hvordan denne prosessen fungerer.



Figur 2.4: Eksempel på modellbasert design

Over kan man se at dette blir en lukket sløyfe som kontinuerlig oppdateres med nye iterasjoner. Siden man kun bruker en modell av systemet så kan testingen og videreutvikling av programvaren foregå mye hurtigere enn ved andre metoder. Dette gjør at selv komplekse kontrollalgoritmer raskt kan bli implementert på fastvareplattformer. Denne metoden fokuserer også mye på simulering, og mange av fordelene kommer herfra. Ved å kunne kjøre simuleringer tidlig i designprosessen, så kan eventuelle feil bli fanget opp og fjernet tidlig [3].

2.2.1 Model-in-the-loop, MiL

MiL-simulering er en metode for testing og simulering som benyttes i en tidlig fase av designprosessen, hvor man kun har en enkel modell av det fysiske systemet og kontrolleren man har designet. Både modellen og kontrolleren vil ofte være i samme dataprogram, for eksempel i samme Simulink modell. I dette stadiet kan man raskt teste overordnet funksjonalitet og gjøre endringer på strukturen [4].

2.2.2 Software-in-the-loop, SiL

SiL-simulering tar et steg videre i forhold til MiL. I dette stadiet har man programmert en mer reell kontroller og kan teste programvarefunksjonaliteten mer omfattende. Ofte vil programvaren være automatisk generert til C/C++ for å kjøre på en innbakt kontroller. Mange kontrollere har en emulert versjon av kontrolleren som kan brukes i forbindelse med SiL, for eksempel har Siemens *PLCSIM* som er en emulert versjon av deres PLSer. Dersom en god emulert versjon av kontrolleren er tilgjenglig kan en stor deler av designprosessen foregå i SiL. Ved å gå over fra MiL til SiL og SiL til HiL går designprosessen tregere da flere komponenter er involvert [4].

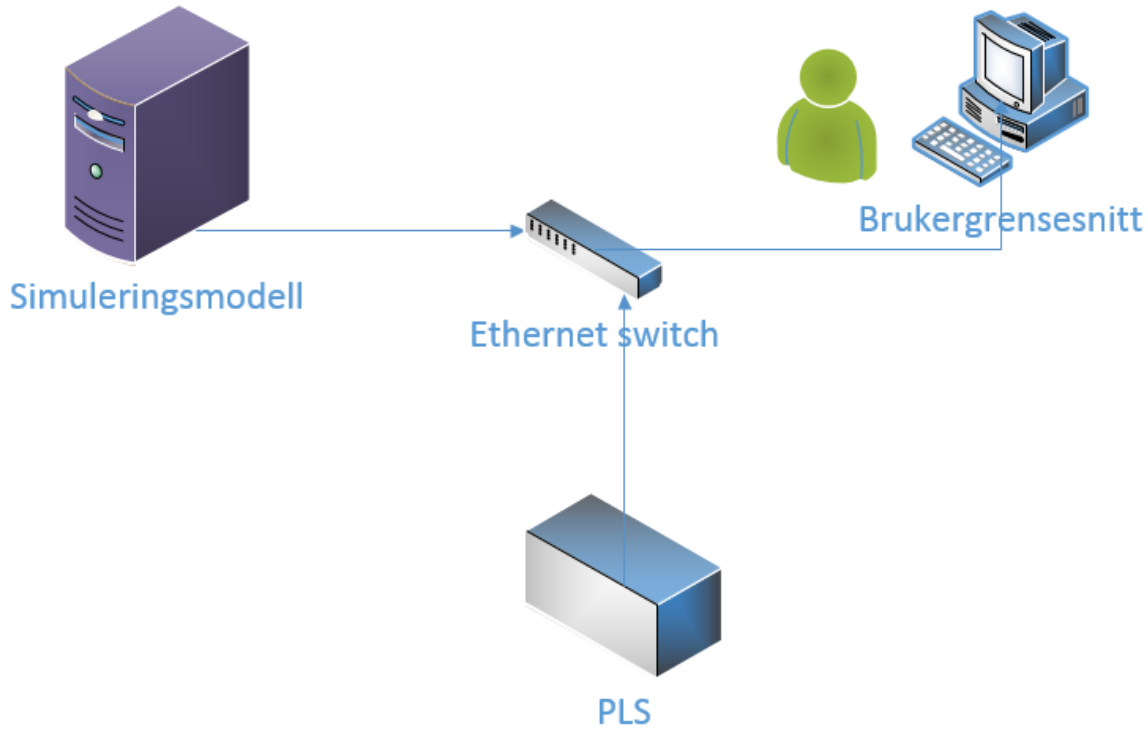
2.2.3 Hardware-in-the-loop, HiL

I en HiL-simulering så har man implementert kontrolleren på et fullverdig kontrollsystem, som skal benyttes på det ferdige produktet. Kontrollsystemet blir koblet til simuleringsmodellen til modellen via inngangene og utgangene som benyttes på det fysiske systemet. Kontrolleren lures til å tro at den mottar signal fra et fysisk system. Simuleringsmodellen kjøres i sanntid, ofte

på en dedikert datamaskin. Forskjellen fra en HiL-simulering og det faktiske systemet, at HiL simuleringen benytter en matematisk modell som oftest ikke fanger all kompleksiteten til et system. Ved å ha en god simuleringsmodell vil simuleringen bli mer reell [4]. For å kjøre en HiL-simulering trenger man følgende:

- Fysisk kontroller
- Simuleringsmodell
- Grensesnitt for kommunikasjon
- Brukergrensesnitt

I figur 2.5 vises det hvordan en HiL-simulering kan settes opp med de aktuelle komponentene.



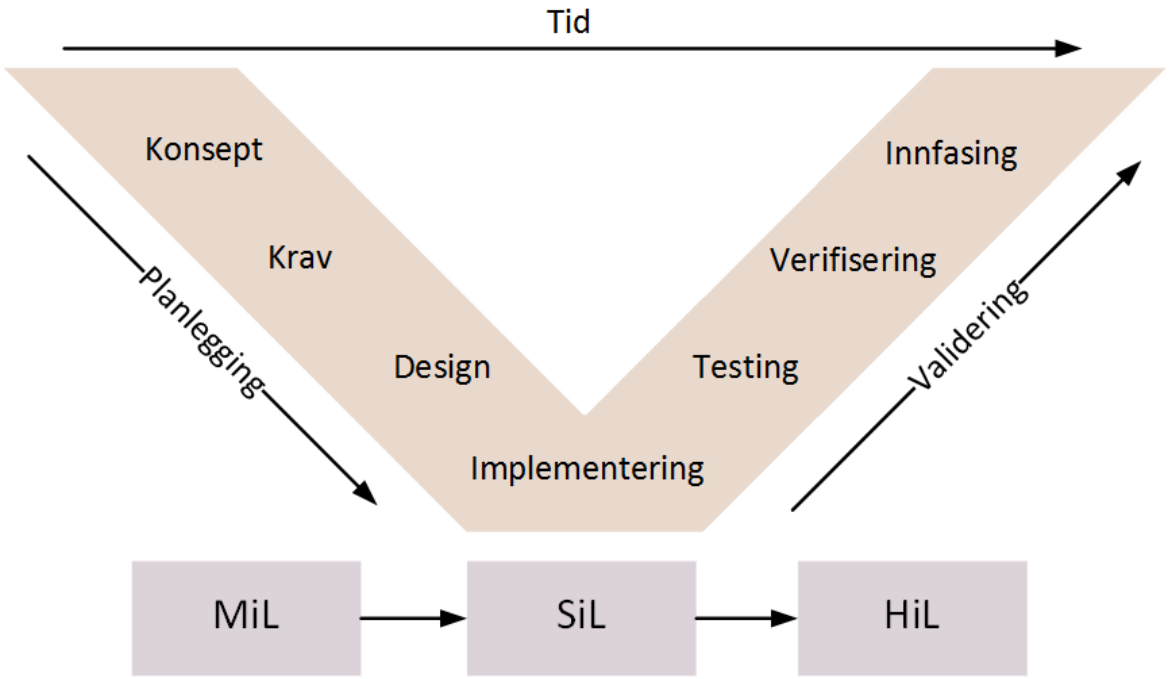
Figur 2.5: Eksempel på oppsett av en HiL-simulering

Dette oppsettet har med de fire nødvendige punktene. En fysisk PLS, ofte med innebygget switch for TCP/IP kommunikasjon, en simuleringsmodell og et brukergrensesnitt som alle tre er tilkoblet via Ethernet TCP/IP. I teorien kan simuleringsmodellen kjøre på samme datamaskin som brukergrensesnittet, men dette er som regel ikke tilfelle da modellen ofte er kompleks og trenger dedikert prosessorkraft for å kunne kjøre i sanntid.

2.2.4 Overordnet designprosess

En del av designprosessen vil være å benytte MiL, SiL og HiL til å teste systemet. Disse metodene benyttes på hver sin del av designprosessen. Man kan også se på V-modellen for å vise i hvilke faser av prosessen som MiL, SiL og HiL benyttes. V-modellen viser et mer generelt og overordnet

bilde av designprosessen. Figur 2.6 viser et eksempel på hvor man benytter MiL, SiL og HiL, sett i forhold til V-modellen.



Figur 2.6: Eksempel på V-modell med Mil, SiL og HiL

V-modellen viser også tidsperspektivet til designprosessen. Hovedprosessen til modellbasert design kan befinne seg mellom design, implementering og testing, hvor man kjører en iterativ prosess. Tabell 2.1 viser hvilke(n) del av systemet som er modellert og hvilke(n) del er reelle.

Tabell 2.1: Modellbasert designprosess

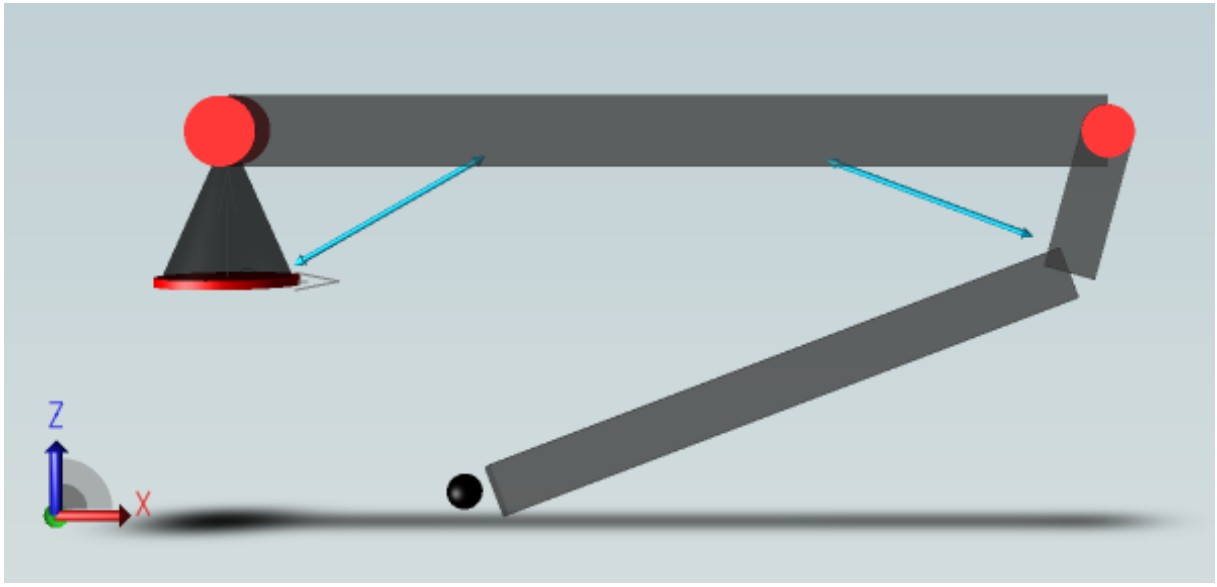
Designmetode	Kontrollalgoritme	Kontroller	Prosess
MiL	Modell	Modell	Modell
SiL	Reell	Modell	Modell
HiL	Reell	Reell	Modell

2.3 Programvare

Dette kapittelet vil forklare hvilken programvare som har blitt brukt. Bruksområde til programvaren vil forklares på generell basis, detaljer rundt løsningen vil diskuteres i kapittel 3.

2.3.1 SimulationX

SimulationX er et program for å simulere dynamiske systemer som er utviklet og solgt av ITI GmbH. Programmet inneholder biblioteker av komponenter til forskjellige disipliner som mekanikk, hydraulikk og elektronikk. Grensesnittet tillater enkel avlesning av relevante verdier som for eksempel volumstrøm, trykk, temperatur, spenning eller forflytning. Dersom modellen er tredimensjonal genereres en 3D-modell av systemet. Ved å definere geometri og relativ plassering av legemer vil SimulationX lage mekanisk modell, dette er vist i figur 2.7.



Figur 2.7: 3D modell av knekkbomkran generert i SimulationX

SimulationX-modellen kan eksporteres og brukes som en simuleringsmodell i en rekke programmer. Det som er relevant for denne oppgaven er muligheten til å eksportere modellen i C-kode som en dll-fil til LabVIEW RealTime, samt som en S-funksjon til Simulink. Ved eksportering velges inngangssignal og utgangssignal. Modellen er en samling av algebraiske og differensialligninger [5].

2.3.2 LabVIEW

Laboratory Virtual Instrumentation Engineering Workbench, LabVIEW, er et verktøy for å programmere applikasjoner som samhandler med virkelig målt data. Labview bruker et programmeringsspråk som heter *G* [6], det er grafisk og skal være intuitivt å bruke. LabVIEW støtter TCP/IP kommunikasjon og kan enkelt ta i bruk dll-filer som simuleringsmodeller. LabVIEW har en tilleggsfunksjon som gjør det mulig å kjøre en modell på en dedikert datamaskin dersom prosessorkraft på datamaskinen ikke er tilstrekkelig. Dette er praktisk når modellen er så komplisert at den må kjøre på veldig lave tidssteg, noe som krever mye prosessorkraft. Denne funksjonaliteten kalles LabVIEW RealTime.

LabVIEW har to hovedvindu; et blokkdiagram og et frontpanel. Frontpanelet er brukergrensesnittet brukeren vil se, mens blokkdiagrammet er programmet som kjører i bakgrunnen. Dette gjør det enklere for personen som designer programmet å endre funksjonalitet og fremvisning underveis i designprosessen.

2.3.3 Siemens SIMATIC STEP 7

Siemens SIMATIC STEP 7 er et programmeringsrammeverk for å programmere Siemens sine PLSer. Programstrukturen deles inn i blokker, hver av blokkene anvendes til forskjellige operasjoner. Blokkene som er tilgjengelig i SIMATIC S7 er.

Data block : Datablokker (DB) brukes til å lagre verdier. Datablokker kan være globale og brukes av alle andre blokker, eller en lokal *instance data block* for kun en funksjonsblokk.

Function Call : En funksjon (FC) behandler inngangssignalet og sender ut utgangssignalet.

Function Block : En funksjonsblokk (FB) gjør det samme som en FC, men har en datablokk (DB) assosiert med seg. Dette er nyttig dersom verdiene må lagres for behandling senere.

Organization Block : Organisasjonsblokken (OB) kaller opp de andre blokkene ved behov. Noen OB blokker kjører på en bestemt klokkefrekvens for å ha kontroll på hvor ofte en operasjon skjer, som for eksempel TCP/IP kommunikasjon.

User Defined Datatype : UDT gjør det mulig for brukeren å lage egne datatyper etter behov.

Det er fire programmeringsspråk fra IEC standard tilgjengelig i SIMATIC S7. Disse vil bli diskutert individuelt, men mest fokus blir lagt på SCL da den er mest relevant for denne oppgaven.

Structured Control Language: SCL er et høynivå optimalisert for å programmere logiske kontrollere og inneholder elementer fra programmeringsspråket PASCAL, samt typiske PLS-elementer som inngang/utgangssignal, klokker og tellere [7]. Dette gjør SCL spesielt egnet for programmering av komplekse algoritmer, matematiske funksjoner, databehandling og prosessoptimalisering. SCL støtter logiske tester og erklæringer som IF, CASE, FOR, WHILE og REPEAT. SCL har dog ikke støtte for matrisenotasjon eller vektorer.

Function Block Diagram: FBD er et grafisk programmeringsspråk. Det blir ofte brukt i OB blokkene for enkel oversikt over FC og FB som blir kallet opp. FBD gjør det enkelt å konvertere filtyper eller utføre enkle matematiske operasjoner.

Ladder Logic: LAD blir brukt i PLSer for kontrollapplikasjoner. Navnet kommer fra utseende til koden, som ligner på stiger (ladders). Det er spesielt egnet for å initiere sekvenser. Det er hovedsakelig brukt til binære prosesser da aritmiske operasjoner er tungvinte å utføre i LAD.

Statement List: STL er et programmeringsspråk for PLSer. Det er et lavnivåspråk som minner om Assembly. Programmet utføres ved *jump* instruksjoner og funksjonsoppkallinger. STL er lite brukt i denne oppgaven utover standard MAL gitt av NOVN.

2.3.4 MATLAB/Simulink

MATLAB er et høynivå programmeringsspråk med en rekke funksjonaliteter som gjør det veldig utbredt over hele verden. Blant annet brukes det til å visualisere og analysere data. MATLAB støtter vektor og matrisekalkulasjoner som er sentralt i ingeniørfag, derav kommer også navnet MATLAB, Matrix Laboratory [8]. Det er ofte hensiktsmessig å bruke blokkdiagrammer for å representere et fysisk system, da kan Simulink brukes. Simulink gir en grafisk fremvisning av et system, det har innebygget et bibliotek med konfigurerbare blokker. Simulink gjør det enkelt å bytte løsertype fra for eksempel kontinuerlig tid til diskret tid. Simulink bygger på MATLAB, en simulering kan sende data til MATLAB for prosessering, og man kan også sende en funksjon fra MATLAB til Simulink.

2.4 Innregulering og optimalisering

For å få et system med god ytelse kommer flere metoder for innregulering til å bli testet og evaluert. I komplekse systemer med mange parametere kan det være ønskelig å unngå manuell innregulering, både fordi man kan ende opp med å bruke mye tid på innreguleringen, men også for å sikre at man benytter et sett med parametere som gir den beste løsningen for systemet. Hovedfokuset vil være å benytte to forskjellige typer for innregulering, som igjen består av forskjellige metoder. Disse er vist under.

- Automatisk innregulering i MATLAB/Simulink
    - PID Tuner

- Minimeringsalgoritmer i MATLAB/Simulink
  - Fmincon
  - Fminsearch
  - Pattern Search
  - Genetic Algorithm

Ved å bruke automatisk innregulering i MATLAB/Simulink kan man definere hvilken kontroller man vil benytte, samt gi visse krav eller mål som systemet skal kunne oppfylle. Dette kan for eksempel være båndbredde eller fasemargin. Ved å bruke disse metodene vil man forsikre seg om at kontrollerparameterne gjør at systemet både er raskt og stabilt, sålenge kravene er mulige å oppfylle.

PID Tuner Dette er en applikasjon i control system toolbox i MATLAB/Simulink hvor man kan innregulere kontrollere med et grafisk grensesnitt. Den benytter en algoritme for å finne kontrollerparametere basert på en systemmodell som skal reguleres. Flere typer kontrollere er tilgjengelige, for eksempel P, PI og PID. Algoritmen vektlegger stabilitet, systemytelse og robusthet. I applikasjonen kan man selv justere vektleggingen på kontrollerkriteriene, for eksempel at man tillater mye oversving så lenge man får hurtig systemrespons. Applikasjonen kan også vise figurer over systemrespons i tidsdomenet og frekvensdomenet, for åpen og lukket sløyfe. Dette gjør PID Tuner til en nyttig verktøy for innregulering av kontrollere [9].

Minimeringsalgoritmene er i en helt annen kategori, og kan brukes på veldig mange forskjellige problemer hvor målet er å finne et sett med parametere som minimerer en gitt funksjon, også kalt en målfunksjon. Alle algoritmene som beskrives i dette delkapittelet kan løse ulineære problemer. I konteksten av denne oppgaven brukes disse algoritmene til å minimere feilsignalet mellom referansen og den målte verdien på utgangen til systemet. Systemet vil da optimaliseres uavhengig av ønsket båndbredde eller lignende, og løsningen vil generelt sett bli så rask og stabil som mulig. Ulempen med denne metoden er at man ikke kan spesifisere samme kriterier som med PID tuner som for eksempel båndbredde og fasemargin. Man mister med andre ord noe kontroll over designprosessen med denne metoden.

Fmincon Dette er en algoritme for å finne minimum på en ulineær multivariabel målfunksjon hvor man i tillegg kan definere visse begrensninger på parameterne. Et eksempel på en begresning kan være at parameterne skal ligge i intervallet [1 10]. Fmincon er en gradient-basert metode som ser på gradientene i et n-dimensjonalt rom for å finne en retning hvor målfunksjonen avtar. Den krever at man definerer et startpunkt for første iterasjon. Neste iterasjon vil bevege seg i retningen med lavest gradient [10].

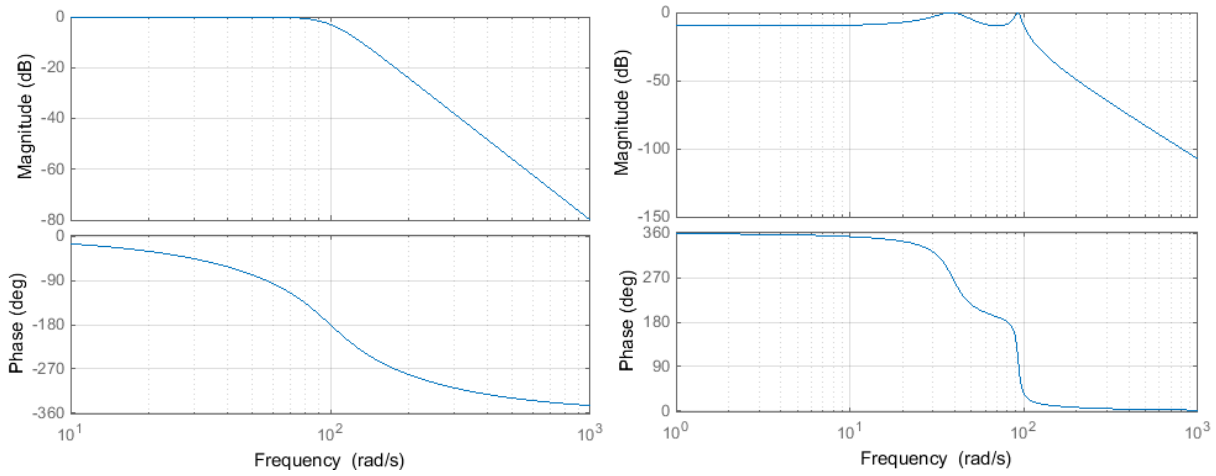
Fminsearch Denne algoritmen bruker en direkte metode uten gradienter for å minimere målfunksjonen. Den trenger også en startverdi til første iterasjon. Man kan ikke ha noen begrensninger med denne algoritmen [11].

Pattern Search Dette er også en direkte metode som sammenligner verdien til målfunksjonen til et sett punkter rundt nåværende punkt. Med Pattern Search kan man også definere begrensninger på parameterne, slik som med Fmincon. Den krever også et startpunkt [12].

Genetic Algorithm Denne algoritmen er en metode for å løse problemer med og uten begrensninger på parametere. Den genererer automatisk en populasjon for hver generasjon. Populasjonen består av individer, som er et sett med unike løsninger på målfunksjonen. Neste generasjon med individer blir laget basert på både mutasjon, overkrysning og seleksjon. I tillegg kan man implementere elitisme, som vil si at de beste genene automatisk blir med over til neste generasjon for å forhindre eventuell mutasjon. Nye generasjoner vil da bevege seg mot en optimal løsning [13].

2.5 Digital filterdesign

De filtrene som designes i denne rapporten vil være digitale filtere, og skal implementeres på en PLS. Det ble testet to filtertyper; Buterworthfilter og Elliptisk filter. Butterworthfilteret ble valgt over det andre alternativet fordi det har en forsterkning på 0 desibel i passbåndet. Med tanke på optimaliseringsalgoritmer vil butterworth filteret være grei å jobbe med, hvis man spesifiserer orden trenger man kun å justere en parameter; avkutningsfrekvensen. For et elliptisk filter må man i tillegg definere tillatt rippel i passbåndet, og minimum dempning i stoppbåndet. En sammenligning mellom et butterworthfilter og et elliptisk filter vises under i figur 2.8.



(a) Butterworthfilter

(b) Elliptisk filter

Figur 2.8: Bodeplot av et 4. ordens butterworthfilter og elliptisk filter

Filterene er designet i MATLAB med funksjonene *butter* og *ellip*, og har en avkutningsfrekvens på 100 rad/s. For det elliptiske filteret var rippel i passbåndet satt til 10dB og minimum dempning i stoppbåndet til 300dB. Som vist i figur 2.8a er forsterkningen til butterworthfilteret helt flat i passbåndet.

I MATLAB kan man med funksjonen *butter* få transferfunksjonen til butterworthfilteret ved å definere orden og avkutningsfrekvens. Dette kan også gjøres manuelt ved å bruke en tabell. Man kan da hente ut normaliserte polynomer for nevneren i transferfunksjonen ved en avkutningsfrekvens på 1 rad/s. Transferfunksjonen er da på formen $G(s) = \frac{1}{a(s)}$. Disse er vist i tabell 2.2, hentet fra [14].

Tabell 2.2: Normaliserte butterworth-polynomer

Orden	Polynom $a(s)$
1	$(1 + s)$
2	$(1 + 1.414 \cdot s + s^2)$
3	$(1 + s) \cdot (1 + s + s^2)$
4	$(1 + 0.765 \cdot s + s^2) \cdot (1 + 1.848 \cdot s + s^2)$
5	$(1 + s) \cdot (1 + 0.618 \cdot s + s^2) \cdot (1 + 1.618 \cdot s + s^2)$
6	$(1 + 0.518 \cdot s + s^2) \cdot (1 + 1.414 \cdot s + s^2) \cdot (1 + 1.932 \cdot s + s^2)$

For å få disse polynomene på generell form som funksjon av avkutningsfrekvensen ω, kan s byttes ut med $\frac{s}{\omega}$.

Siden filterene skal implementeres på en digital kontroller, så må man ha de på en form som kan benyttes på PLSen. En diskret transferfunksjon i z-domenet vil kunne brukes til å finne en differanseligning som kan programmeres inn i PLSen. For å transformere en transferfunksjon fra s-domenet til z-domenet vil en bilineær transformasjon benyttes, også kalt Tustins metode. Denne metoden opprettholder blant annet frekvensresponsen til systemet, og er mye brukt for digitale filtere. Utledningen er vist i formel 2.1 til 2.5, hentet fra [15].

$$z = e^{sT} \tag{2.1}$$

$$s = \frac{1}{T}\ln(z) \tag{2.2}$$

$$s = \frac{2}{T}\left[\frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \frac{1}{5}\left(\frac{z-1}{z+1}\right)^5 + \ldots\right] \tag{2.3}$$

$$s \approx \frac{2}{T}\frac{z-1}{z+1} \tag{2.4}$$

$$s = \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \tag{2.5}$$

hvor;

T = tidsteget til systemet

Denne transformasjonen kan brukes til å transformere et butterworthfilter til z-domenet og videre til en differanseligning. Transformasjonen til et 2. ordens butterworthfilter er vist i formel 2.6 til 2.25.

Hjelpe variabelen c introduseres.

$$c = \omega \cdot \frac{T}{2} \tag{2.6}$$

Løser med hensyn på ω

$$\omega = c \cdot \frac{2}{T} \tag{2.7}$$

Kontinuerlig 2. orden transferfunksjon som av butterworth filter som vist i tabell 2.2

$$G(s) = \frac{\omega^2}{s^2 + \sqrt{2}\cdot\omega\cdot s + \omega^2} \tag{2.8}$$

Diskretiserer transfer funksjonen og substituerer inn hjelpevariabelen c.

$$G(z) = \frac{c^2 \cdot \left(\frac{2}{T}\right)^2}{\left(\frac{2}{T}\right)^2 \cdot \left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \left(\frac{2}{T}\right)^2 \cdot \sqrt{2}\cdot c \cdot \left(\frac{1-z^{-1}}{1+z^{-1}}\right) + c^2 \cdot \left(\frac{2}{T}\right)^2} \tag{2.9}$$

Forenkler uttrykket, stryker $\left(\frac{2}{T}\right)^2$

$$G(z) = \frac{c^2}{\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \sqrt{2}\cdot c \cdot \left(\frac{1-z^{-1}}{1+z^{-1}}\right) + c^2} \tag{2.10}$$

Multipliserer ut parentesene.

$$G(z) = \frac{c^2\cdot z^2 + 2\cdot c^2\cdot z + c^2}{(c^2 + \sqrt{2}\cdot c + 1)\cdot z^2 + (2\cdot c^2 - 2)\cdot z + (c^2 - \sqrt{2}\cdot c + 1)} \tag{2.11}$$

Multipliserer med $\frac{z^{-2}}{z^{-2}}$

$$G(z) = \frac{c^2 + 2\cdot c^2\cdot z^{-1} + c^2\cdot z^{-2}}{(c^2 + \sqrt{2}\cdot c + 1) + (2\cdot c^2 - 2)\cdot z^{-1} + (c^2 - \sqrt{2}\cdot c + 1)\cdot z^{-2}} \tag{2.12}$$

Sorterer utrykket slik koeffisientene til variablene kommer frem

$$G(z) = \frac{\frac{c^2}{c^2+\sqrt{2}\cdot c+1} + \frac{2\cdot c^2}{c^2+\sqrt{2}\cdot c+1}\cdot z^{-1} + \frac{c^2}{c^2+\sqrt{2}\cdot c+1}\cdot z^{-2}}{1 + \frac{2\cdot c^2-2}{c^2+\sqrt{2}\cdot c+1}\cdot z^{-1} + \frac{c^2-\sqrt{2}\cdot c+1}{c^2+\sqrt{2}\cdot c+1}\cdot z^{-2}} \tag{2.13}$$

Substituerer inn hjelpekonstanter for koeffisientene til variablene

$$G(z) = \frac{b_0 + b_1\cdot z^{-1} + b_2\cdot z^{-2}}{a_0 + a_1\cdot z^{-1} + a_2\cdot z^{-2}} \tag{2.14}$$

Forklaring av hjelpekonstantene

$$b_0 = \frac{c^2}{c^2+\sqrt{2}\cdot c+1} \tag{2.15}$$

$$b_1 = 2\cdot b_0 \tag{2.16}$$

$$b_2 = b_0 \tag{2.17}$$

$$a_0 = 1 \tag{2.18}$$

$$a_1 = \frac{2\cdot c^2-2}{c^2+\sqrt{2}\cdot c+1} \tag{2.19}$$

$$a_2 = \frac{c^2-\sqrt{2}\cdot c+1}{c^2+\sqrt{2}\cdot c+1}. \tag{2.20}$$

Deler transferfunksjonen opp utgangssignal delt på inngangssignal.

$$\frac{Y_k}{X_k} = \frac{b_0 + b_1\cdot z^{-1} + b_2\cdot z^{-2}}{a_0 + a_1\cdot z^{-1} + a_2\cdot z^{-2}} \tag{2.21}$$

Kryssmultipliserer for å fjerne brøken.

$$Y_k\cdot(a_0 + a_1\cdot z^{-1} + a_2\cdot z^{-2}) = X_k\cdot(b_0 + b_1\cdot z^{-1} + b_2\cdot z^{-2}) \tag{2.22}$$

Multipliserer ut parentesene.

$$Y_k\cdot a_0 + Y_{k-1}\cdot a_1 + Y_{k-2}\cdot a_2 = X_k\cdot b_0 + X_{k-1}\cdot b_1 + X_{k-2}\cdot b_2 \tag{2.23}$$

Løser med hensyn på Y_k

$$Y_k = \frac{X_k\cdot b_0 + X_{k-1}\cdot b_1 + X_{k-2}\cdot b_2 - Y_{k-1}\cdot a_1 - Y_{k-2}\cdot a_2}{a_0} \tag{2.24}$$

Siden $a_0 = 1$ kan denne fjernes.

$$Y_k = X_k\cdot b_0 + X_{k-1}\cdot b_1 + X_{k-2}\cdot b_2 - Y_{k-1}\cdot a_1 - Y_{k-2}\cdot a_2 \tag{2.25}$$

Hvor;

T	=	tidsteget til systemet
c	=	konstant for mellomregning
b_n	=	konstanter i teller
a_n	=	konstanter i nevner
Y_n	=	utgangssignal
X_n	=	inngangssignal

Som ligning 2.25 viser får man en differanseligning som benytter foregående verdier til både inngangssignalet og utgangssignalet. Samme metode kan også benyttes på filtere av høyere orden.

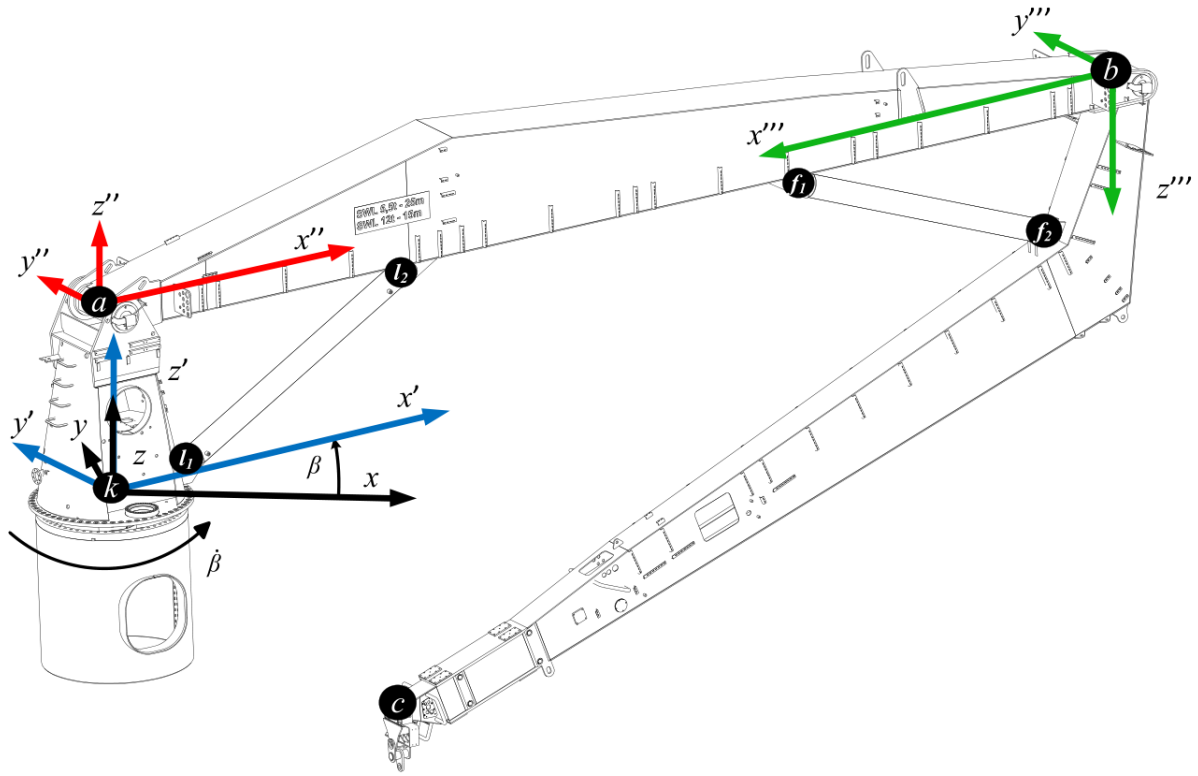
2.6 Kinematikk for pipehandlerkran

Kinematikken for denne kranen var utarbeidet av tidligere mastergruppe [1]. Det har kun blitt gjort små endringer fra deres kapittel 2. Dette kapittelet er en sitering for å enklere kunne lese denne rapporten isolert.

Kinematikken som er utviklet for rørhåndteringskranen bruker fire koordinatsystem.

- *Globalt* koordinatsystem xyz
- Kranens koordinatsystem $x'y'z'$
- Hovedbommens koordinatsystem $x''y''z''$
- Knekkbommens koordinatsystem $x'''y'''z'''$

I tillegg er åtte punkter definert for å kunne utlede ligningene for krankinematikken. Disse åtte punktene og de fire koordinatsystemene er vist på figur 2.9.



Figur 2.9: Oversikt over kranens koordinatsystemer og definerte punkter

Origo for xyz og $x'y'z'$ er definert av punktet k, plassert i senter av svinglageret mellom kongen og pidestallen. Punktet a er definert i leddet mellom kongen og hovedbommen, og er origo for $x''y''z''$. Punktet b er definert i leddet mellom hovedbommen og knekkbommen, og er origo for $x'''y'''z'''$. Punktene l_1, l_2, f_1 og f_2 er festepunktene til henholdsvis løftesylinder og foldesylinder. Punktet c er definert i krantuppen. Posisjonen til krantuppen vil definere kranens arbeidsradius, R, som er avstanden fra globalt origo til punktet c sin projeksjon i xy-planet, vist i formel 2.26.

$$R = \sqrt{x_c^2 + y_c^2} \tag{2.26}$$

Vinkelen til denne projeksjonen om z-aksen vil være lik vinkelen β mellom xyz og $x'y'z'$, gitt i formel 2.27

$$\beta = \arctan\left(\frac{y_c}{x_c}\right) \tag{2.27}$$

I kranens koordinatsystem $x'y'z'$ vil punktet c alltid befinne seg i $x'z'$-planet, og kan beskrives med vektoren s'_{kc} vist i formel 2.28.

$$s'_{kc} = \begin{bmatrix} x'_c \\ y'_c \\ z'_c \end{bmatrix} = \begin{bmatrix} R \\ 0 \\ z_c \end{bmatrix} \tag{2.28}$$

En transformasjonsmatrise $\mathbf{A}$ defineres for overføring mellom xyz og $x'y'z'$ ut ifra vinkelen β, vist i formel 3.29.

$$\mathbf{A} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{A}^{-1} = \mathbf{A}^T = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.29}$$

Den globale hastighetsvektoren til punktet c kan også defineres. Ofte vil denne vektoren være en referanse som benyttes av kontrolleren. Vektoren v_c er vist i formel 2.30.

$$v_c = \begin{bmatrix} v_{x_c} \\ v_{y_c} \\ v_{z_c} \end{bmatrix} \tag{2.30}$$

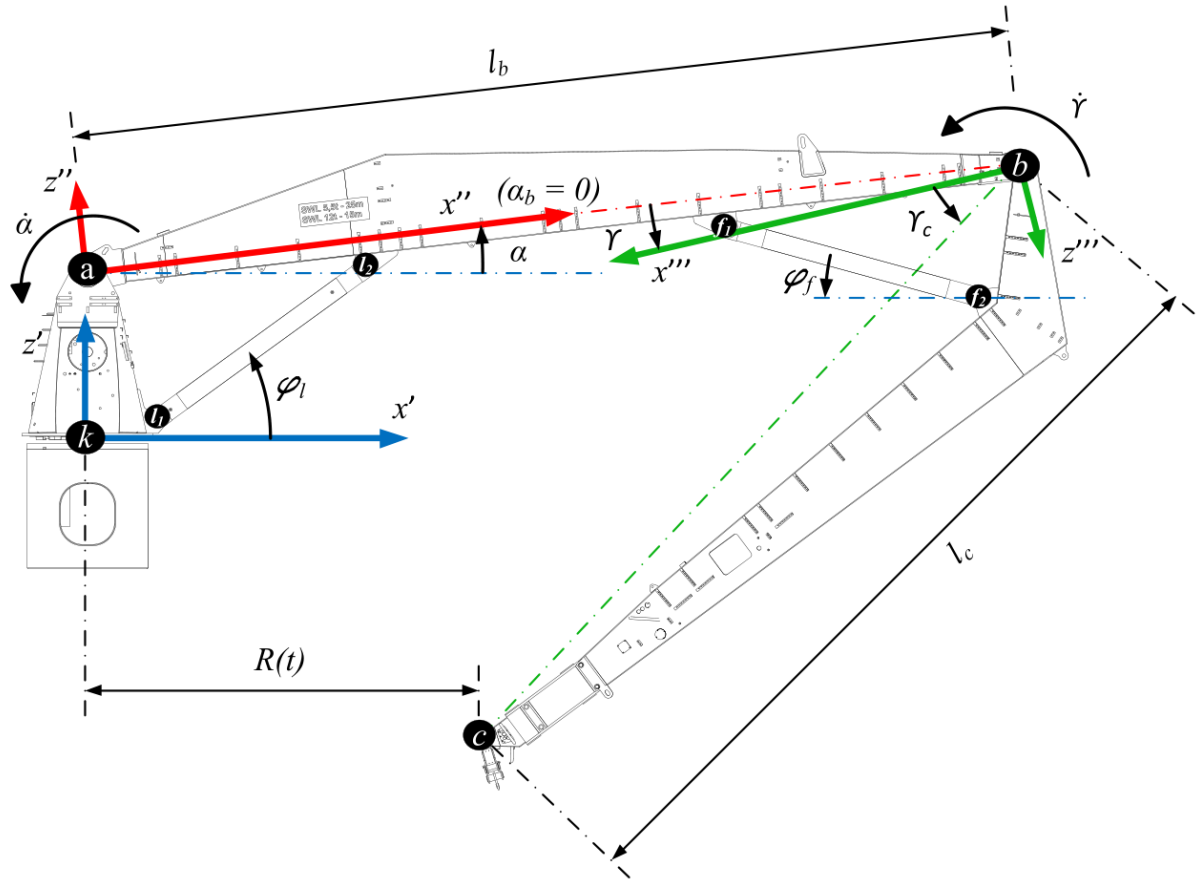
Denne globale hastighetsvektoren kan så transformeres til kranens koordinatsystem ved å bruke den transponerte av $\mathbf{A}$. Ved å transformere både posisjonen og hastigheten til c over i kranens koordinatsystem vil mange av videre utregninger kunne foregå i to dimensjoner fremfor tre dimensjoner, da alle de 8 definerte punktene ligger i $x'z'$-planet. Transformert hastighetsvektor er vist i formel 2.31.

$$v'_c = \begin{bmatrix} v'_{x_c} \\ v'_{y_c} \\ v'_{z_c} \end{bmatrix} = \mathbf{A}^T \cdot v_c = \begin{bmatrix} v_x \cdot \cos\beta + v_y \cdot \sin\beta \\ -v_x \cdot \sin\beta + v_y \cdot \cos\beta \\ v_z \end{bmatrix} \tag{2.31}$$

Hastigheten v'_{y_c} kan brukes sammen med arbeidsradiusen for å regne ut vinkelhastigheten til svingkransen, $\dot{\beta}$, vist i formel 2.32.

$$\dot{\beta} = \frac{v'_{y_c}}{R} \tag{2.32}$$

Fremtidige utregninger vil i all hovedsak foregå i $x'z'$-planet. Figur 2.10 viser kranen sett fra siden.



Figur 2.10: Oversikt over kranens koordinatsystemer og definerte punkter, sett fra siden

Hovedbommen sitt innfestningspunkt a kan beskrives med vektoren s'_{ka}.

$$s'_{ka} = \begin{bmatrix} x'_a \\ z'_a \end{bmatrix} \tag{2.33}$$

Løftesylinderen sitt innfestningspunkt på kongen, l_1 kan beskrives med vektoren s'_{kl_1}, vist i formel 2.34.

$$s'_{kl_1} = \begin{bmatrix} x'_{l_1} \\ z'_{l_1} \end{bmatrix} \tag{2.34}$$

Både formel 2.33 og formel 2.34 er definert i kranens koordinatsystem, og disse vektorene vil være konstante uavhengig av kranens positur.

En transformasjonsmatrise $\mathbf{A}'$ brukes for å transformere fra kranens koordinatsystem $x'y'z'$ til hovedbommens koordinatsystem $x''y''z''$ basert på hovedbomvinkelen α.

$$\mathbf{A}' = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \tag{2.35}$$

Løftesylinderen sitt innfestningspunkt på hovedbommen, l_2, er definert i hovedbommens koordinatsystem med origo gitt i a, vist i formel 2.36. Denne vektoren vil også være konstant uavhengig av positur.

$$s''_{al_2} = \begin{bmatrix} x''_{l_2} \\ z''_{l_2} \end{bmatrix} \tag{2.36}$$

Denne vektoren kan transformeres til kranens koordinatsystem, vist i formel 2.37.

$$s'_{kl_2} = \begin{bmatrix} x'_{l_2} \\ z'_{l_2} \end{bmatrix} = s'_{ka} + \mathbf{A}' \cdot s''_{al_2} = \begin{bmatrix} x'_a + x''_{l_2} \cdot \cos\alpha - z''_{l_2} \cdot \sin\alpha \\ z'_a + x''_{l_2} \cdot \sin\alpha + z''_{l_2} \cdot \cos\alpha \end{bmatrix} \tag{2.37}$$

Foldesylinderen sitt innfestningspunkt på hovedbommen, f_1 kan beskrives med vektoren s''_{af_1}.

$$s''_{af_1} = \begin{bmatrix} x''_{f_1} \\ z''_{f_1} \end{bmatrix} \tag{2.38}$$

Denne vektoren kan så transformeres til kranens koordinatsystem.

$$s'_{kf_1} = \begin{bmatrix} x'_{f_1} \\ z'_{f_1} \end{bmatrix} = s'_{ka} + \mathbf{A}' \cdot s''_{af_1} = \begin{bmatrix} x'_a + x''_{f_1} \cdot \cos\alpha - z''_{f_1} \cdot \sin\alpha \\ z'_a + x''_{f_1} \cdot \sin\alpha + z''_{f_1} \cdot \cos\alpha \end{bmatrix} \tag{2.39}$$

Knekkbommen sitt innfestningspunkt b kan beskrives med vektoren s''_{ab}, vist i formel 2.40.

$$s''_{ab} = \begin{bmatrix} x''_b \\ z''_b \end{bmatrix} \tag{2.40}$$

For å transformere denne vektoren til krankoordinater benyttes en geometrisk betraktning med lengden og vinkelen mellom punkt a og b. Lengden beregnes som vist i formel 2.41.

$$l_b = \mid s''_{ab} \mid = \sqrt{(x''_b)^2 + (z''_b)^2} \tag{2.41}$$

Vinkelen beskrives med orienteringen til vektoren s''_{ab} i hovedbomkoordinatsystemet som vist i formel 2.42. Denne vinkelen er konstant, og er 0° så lenge x'' -aksen går gjennom både punkt a og punkt b.

$$\alpha_b = \arctan\left(\frac{z''_b}{x''_b}\right) \tag{2.42}$$

Lengden l_b og vinkelen α_b brukes sammen med løftevinkel α, til å beskrive punktet b i krankoordinatsystemet. Dette er vist i formel 2.43.

$$s'_{kb} = \begin{bmatrix} x'_b \\ z'_b \end{bmatrix} = \begin{bmatrix} x'_a + l_b \cdot \cos(\alpha + \alpha_b) \\ z'_a + l_b \cdot \sin(\alpha + \alpha_b) \end{bmatrix} \tag{2.43}$$

Videre defineres punktet l_2 som innfestingspunktet til løftesylinderen på hovedbommen. Dette punktet kan beskrives i hovedbomkoordinater med vektoren vist i formel 2.44

$$s''_{al_2} = \begin{bmatrix} x''_{l_2} \\ z''_{l_2} \end{bmatrix} \tag{2.44}$$

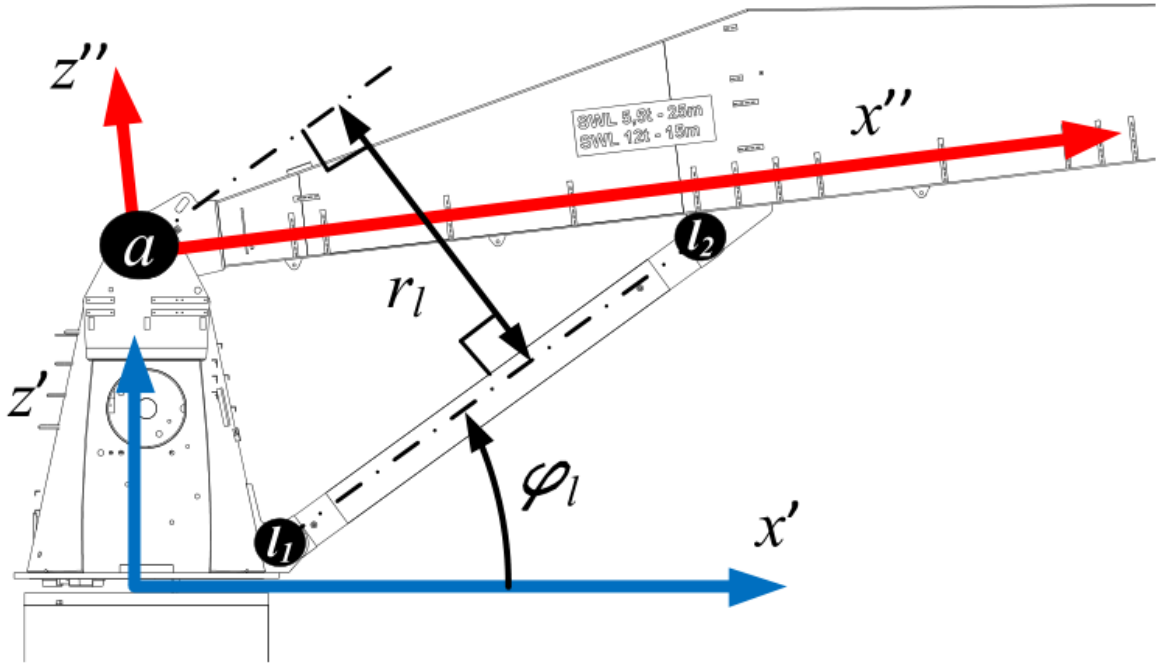
Vektoren i 2.44 transformeres til krankoordinatsystemet som vist i 2.45.

$$s'_{kl_2} = s'_{ka} + \mathbf{A}' \cdot s''_{al_2} = \begin{bmatrix} x'_{l_2} \\ z'_{l_2} \end{bmatrix} = \begin{bmatrix} x'_a + x''_{l_2} \cdot \cos\alpha - z''_{l_2} \cdot \sin\alpha \\ z'_a + x''_{l_2} \cdot \sin\alpha + z''_{l_2} \cdot \cos\alpha \end{bmatrix} \tag{2.45}$$

Ved hjelp av vektorene s'_{kl_1} og s'_{kl_2} kan løftesylinderens vinkel beregnes som vist i formel 2.46

$$\varphi_l = \arctan\left(\frac{z'_{l_2} - z'_{l_1}}{x'_{l_2} - x'_{l_1}}\right) \tag{2.46}$$

Denne vinkelen er relativ til x'-aksen som vist i figur 2.11.



Figur 2.11: Radiusen rl

Ved hjelp av denne vinkelen kan radiusen r_l beregnes, som vist i formel 2.47.

$$r_l = \frac{(x'_{l_1} - x'_a) \cdot \tan\varphi_l - (z'_{l_1} - z'_a)}{\sqrt{\tan^2\varphi_l + 1}} \tag{2.47}$$

Foldesylinderens innfestingspunkt f_1 på hovedbommen kan beskrives i lokale koordinater som vektoren i formel 2.48

$$s''_{af_1} = \begin{bmatrix} x''_{f_1} \\ z''_{f_1} \end{bmatrix} \tag{2.48}$$

Vektoren s''_{af_1} transformeres til krankoordinatsystemet som vist i formel 2.49

$$s'_{kf_1} = s'_{ka} + \mathbf{A}' \cdot s''_{af_1} = \begin{bmatrix} x'_{f_1} \\ z'_{f_1} \end{bmatrix} = \begin{bmatrix} x'_a + x''_{f_1} \cdot \cos\alpha - z''_{f_1} \cdot \sin\alpha \\ z'_a + x''_{f_1} \cdot \sin\alpha + z''_{f_1} \cdot \cos\alpha \end{bmatrix} \tag{2.49}$$

Et nytt koordinatsystem $x'''y'''z'''$ opprettes med origo i b som vist i figur 2.9. Dette koordinatsystemet vil følge knekkbommens bevegelse. Transformasjonsmatrisen fra $x'''z'''$ til $x''z''$ er gitt i formel 2.50.

$$\mathbf{A}'' = \begin{bmatrix} \cos(\gamma+\pi) & -\sin(\gamma+\pi) \\ \sin(\gamma+\pi) & \cos(\gamma+\pi) \end{bmatrix} = \begin{bmatrix} -\cos\gamma & \sin\gamma \\ -\sin\gamma & -\cos\gamma \end{bmatrix} \tag{2.50}$$

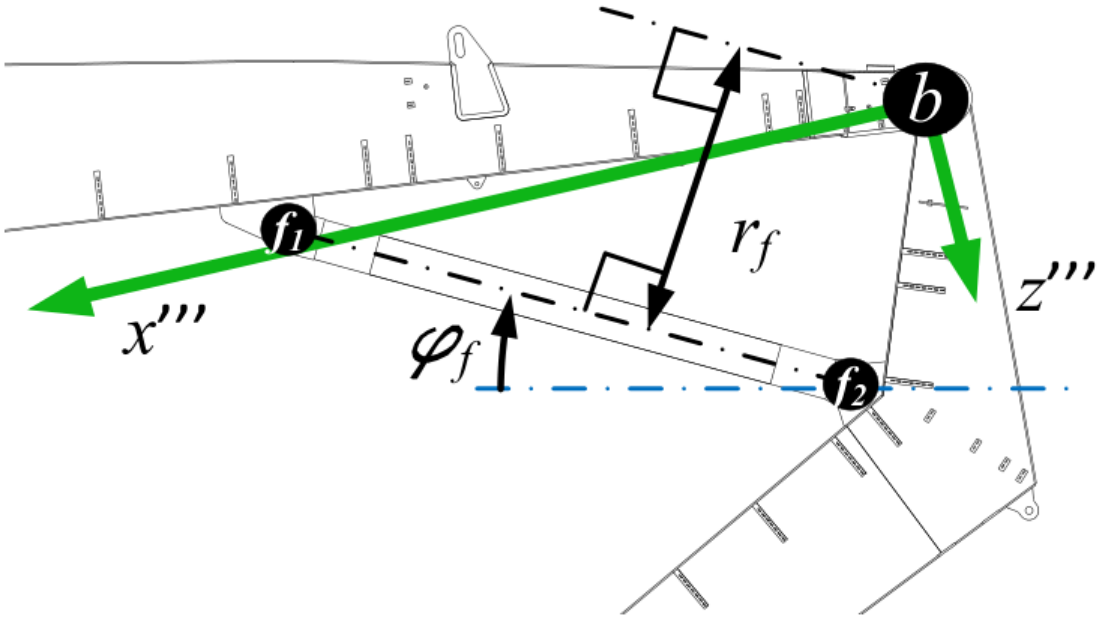
Vinkelen γ er kranarmens foldevinkel. Siden $x'''y'''z'''$ er definert med x'''-aksen mot venstre, må man legge til π radianer i vinkelen i $\mathbf{A}''$. Punktet f_2 defineres som foldesylinderens innfestningspunkt på knekkbommen. I knekkbommen sitt koordinatsystem kan vektoren fra origo til f_2 beskrives med formel 2.51

$$s'''_{bf_2} = \begin{bmatrix} x'''_{f_2} \\ z'''_{f_2} \end{bmatrix} \tag{2.51}$$

s'''_{bf_2} transformeres til $x'z'$-planet, som vist i formel 2.52

$$s'_{kf_2} = s'_{kb} + \mathbf{A}' \cdot \mathbf{A}'' \cdot s'''_{bf_2} = \begin{bmatrix} x'_{f_2} \\ z'_{f_2} \end{bmatrix} = \begin{bmatrix} x'_b - x'''_{f_2} \cdot \cos(\alpha+\gamma) + z'''_{f_2} \cdot \sin(\alpha+\gamma) \\ z'_b - x'''_{f_2} \cdot \sin(\alpha+\gamma) - z'''_{f_2} \cdot \cos(\alpha+\gamma) \end{bmatrix} \tag{2.52}$$

Ved hjelp av s'_{kf_1} og s'_{kf_2} kan foldesylinderens vinkel regnes uyt. Vinkelen er vist figur 2.12.



Figur 2.12: Radiusen rf

Beregning av vinkelen er vist i formel 2.53.

$$\varphi_f = \arctan\left(\frac{z'_{f_2} - z'_{f_1}}{x'_{f_2} - x'_{f_1}}\right) \tag{2.53}$$

Ved hjelp av denne vinkelen kan radiusen r_f beregnes, vist i formel 2.54

$$r_f = \frac{(x'_{f_1} - x'_b) \cdot \tan\varphi_l - (z'_{f_1} - z'_b)}{\sqrt{\tan^2\varphi_f + 1}} \tag{2.54}$$

Det som gjenstår å utlede er vinkelhastighetene $\dot{\alpha}$ og $\dot{\gamma}$ som kan finnes da r_l og r_f kjente. Lokal hastigheten for punkt b i $x''z''$-planet defineres som v''_b . Denne hastigheten kan beskrives ved hjelp av vinkelhastigheten $\dot{\alpha}$ og beregnes opp som vist i formel 2.55

$$v''_b = \begin{bmatrix} v''_{bx} \\ v''_{bz} \end{bmatrix} = \begin{bmatrix} -\dot{\alpha} \cdot z''_b \\ \dot{\alpha} \cdot x''_b \end{bmatrix} \tag{2.55}$$

Den lokale hastigheten v''_b blir transformert til kranens koordinatsystem ved hjelp av $\mathbf{A}'$ som vist i formel 2.56. Uttrykket forenkles ved å introdusere variablene ξ_a og ξ_b.

$$v'_b = \mathbf{A}' \cdot v''_b = \mathbf{A}' \cdot \begin{bmatrix} -\dot{\alpha} \cdot z''_b \\ \dot{\alpha} \cdot x''_b \end{bmatrix} = \begin{bmatrix} \dot{\alpha} \cdot (-z''_b \cdot \cos\alpha - x''_b \sin\alpha) \\ \dot{\alpha} \cdot (-z''_b \cdot \sin\alpha + x''_b \cos\alpha) \end{bmatrix} = \dot{\alpha} \cdot \begin{bmatrix} \xi_a \\ \xi_b \end{bmatrix} \tag{2.56}$$

Lokal hastighet for punktet c i $x'''z'''$- planet, defineres som v'''_{cb}. Den kan beskrives som vist i formel 2.57. Summen $(\dot{\alpha} + \dot{\gamma})$ er vinkelhastigheten til knekkbommen relativ til det globale koordinatsystemet.

$$v_b''' = \begin{bmatrix} v_{cbx}''' \\ v_{cbz}''' \end{bmatrix} = \begin{bmatrix} -(\dot{\alpha}+\dot{\gamma}) \cdot z_c''' \\ (\dot{\alpha}+\dot{\gamma}) \cdot x_c''' \end{bmatrix} \tag{2.57}$$

Hastigheten v_{cb}''' transformeres til kranens koordinatsystem i formel 2.59 ved hjelp av transformasjonsmatrisene $\mathbf{A}'$ og $\mathbf{A}''$. Den globale vinkelhastigheten $(\dot{\alpha}+\dot{\gamma})$ faktoriseres ut og ξ_c og ξ_d benyttes for å samle uttrykkene.

$$v_{cb}' = \mathbf{A}' \cdot v_{cb}'' = \mathbf{A}' \cdot \mathbf{A}'' \cdot v_{cb}''' = \begin{bmatrix} (\dot{\alpha}+\dot{\gamma}) \cdot (z_c''' \cdot \cos(\alpha+\gamma) + x_c''' \sin(\alpha+\gamma)) \\ (\dot{\alpha}+\dot{\gamma}) \cdot (z_c''' \cdot \sin(\alpha+\gamma) - x_c''' \cos(\alpha+\gamma)) \end{bmatrix} = (\dot{\alpha}+\dot{\gamma}) \cdot \begin{bmatrix} \xi_c \\ \xi_d \end{bmatrix} \tag{2.58}$$

Ligningen som brukes for å beskrive samlet hastighet for punkt c er kombinert som vist i formel 2.59.

$$v_c' = \begin{bmatrix} v_{x_c}' \\ v_{z_c}' \end{bmatrix} = v_{cb}' + v_b' = \begin{bmatrix} \xi_a \cdot \dot{\alpha} + \xi_c \cdot (\dot{\alpha}+\dot{\gamma}) \\ \xi_b \cdot \dot{\alpha} + \xi_d \cdot (\dot{\alpha}+\dot{\gamma}) \end{bmatrix} = \begin{bmatrix} (\xi_a+\xi_c) & \xi_c \\ (\xi_b+\xi_d) & \xi_d \end{bmatrix} \cdot \begin{bmatrix} \dot{\alpha} \\ \dot{\gamma} \end{bmatrix} \tag{2.59}$$

Ligningen viser punktet c sin hastighet i $x'z'$-planet med hensyn på rotasjonshastighetene $\dot{\alpha}$ og $\dot{\gamma}$. Tidligere ble den generelle formelen for hastighet i kranens koordinatsystem utledet i formel 2.31. I formel 2.60 er x' og z' hastighetene fra denne ligningen satt inn for å beskrive forholdet mellom vinkelhastighetene $\dot{\alpha}$ og $\dot{\gamma}$ og globale hastigheter.

$$\begin{bmatrix} (\xi_a+\xi_c) & \xi_c \\ (\xi_b+\xi_d) & \xi_d \end{bmatrix} \cdot \begin{bmatrix} \dot{\alpha} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} v_{x_c}' \\ v_{z_c}' \end{bmatrix} = \begin{bmatrix} v_x \cdot \cos\beta + v_y \cdot \sin\beta \\ v_z \end{bmatrix} \tag{2.60}$$

Ved å snu denne ligningen og løse den med hensyn på $\dot{\alpha}$ og $\dot{\gamma}$, kan vinkelhastighetene løses ut til separate ligninger. Dette er vist i formel 2.61

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} (\xi_a+\xi_c) & \xi_c \\ (\xi_b+\xi_d) & \xi_d \end{bmatrix}^{-1} \cdot \begin{bmatrix} v_x \cdot \cos\beta + v_y \cdot \sin\beta \\ v_z \end{bmatrix} \tag{2.61}$$

Ligningene for rotasjonshastighet er etablert. Utstrekningshastigheten til løfte- og foldesylinderen kan regnes ut som vist i formel 2.62.

$$v_l = \dot{\alpha} \cdot r_l \quad og \quad v_f = \dot{\gamma} \cdot r_f \tag{2.62}$$

Punktet c sin plassering i knekkbomkoordinater kan beskrives med vektoren i formel 2.63.

$$s_{bc}''' = \begin{bmatrix} x_c''' \\ z_c''' \end{bmatrix} \tag{2.63}$$

For å transformere punktet til krankoordinater benyttes en geometrisk betraktning med lengden og vinkelen mellom punkt b og c. Lengden beregnes som vist i formel 2.64.

$$l_c = \mid s_{bc}''' \mid = \sqrt{(x_c''')^2 + (z_c''')^2} \tag{2.64}$$

Vinkelen beskrives med orienteringen til vektor s_{bc}''' i lokale knekkbomkoordinater som er vist i formel 2.65, denne vinkelen er konstant.

$$\gamma_c = \arctan\left(\frac{z_c'''}{x_c'''}\right) \tag{2.65}$$

Lengden l_c og vinkelen γ_c benyttes sammen med en gitt løftevinkel α og gitt foldevinkel γ, til å beskrive vektoren fra k til c i krankoordinatsystemet. Dette er vist i formel 2.66.

$$s'_{kc} = \begin{bmatrix} x'_c \\ z'_c \end{bmatrix} = \begin{bmatrix} x'_a + l_b \cdot \cos(\alpha + \alpha_b) + l_c \cdot \cos(\alpha + \gamma + \gamma_c + \pi) \\ z'_a + l_b \cdot \sin(\alpha + \alpha_b) + l_c \cdot \sin(\alpha + \gamma + \gamma_c + \pi) \end{bmatrix} \tag{2.66}$$

For å beskrive plasseringen av krantuppen globalt, transformeres vektoren fra k til c til globalt koordinatsystem som vist i formel 2.67

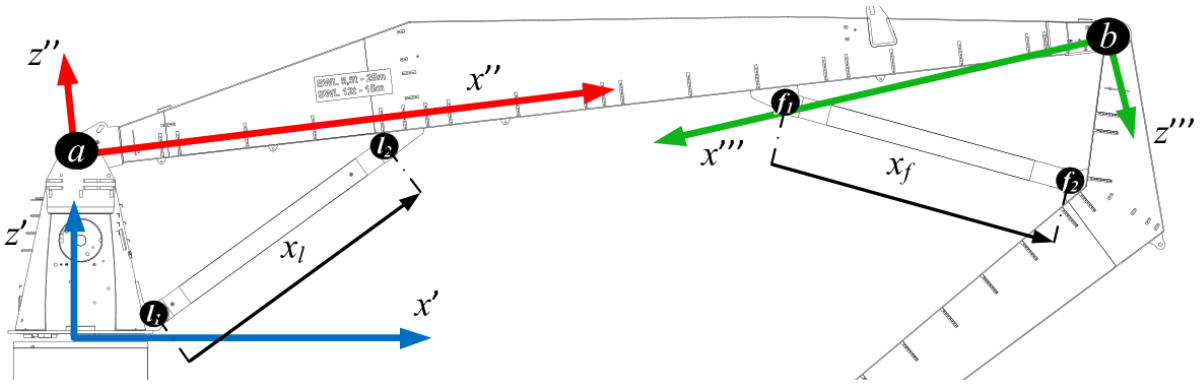
$$s_{kc} = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \mathbf{A} \cdot \begin{bmatrix} x'_c \\ 0 \\ z'_c \end{bmatrix} = \begin{bmatrix} \cos(\beta) \cdot x'_c \\ \sin(\beta) \cdot x'_c \\ z'_c \end{bmatrix} \tag{2.67}$$

For kjente x'_c og z'_c koordinater, kan korresponderende vinkler α og γ beregnes som vist i formel 2.68 og 2.69. Vinklene er løst ut av formel 2.66. Disse uttrykkene er tidligere utledet [16].

$$\alpha = \arccos\left(\frac{l_b^2 - l_c^2 + (x'_c - x'_a)^2 + (z'_c - z'_a)^2}{2 \cdot l_b \cdot \sqrt{(x'_c - x'_a)^2 + (z'_c - z'_a)^2}}\right) + \arctan\left(\frac{z'_c - z'_a}{x'_c - x'_a}\right) - \alpha_b \tag{2.68}$$

$$\gamma = \arccos\left(\frac{l_b^2 + l_c^2 - (x'_c - x'_a)^2 - (z'_c - z'_a)^2}{2 \cdot l_b \cdot l_c}\right) - \gamma_c + \alpha_b \tag{2.69}$$

Til slutt kan utstrekningen av sylinderne beskrives som lengden av vektoren mellom punkt l_1 og l_2, og f_1 og f_2. Disse lengdene kalles x_l og x_f og vises i figur 2.13.



Figur 2.13: Utstrekningene xl og xf

Utregningen av x_l vises i formel 2.70.

$$x_l = \mid s'_{l_1 l_2} \mid = \mid s'_{kl_2} - s'_{kl_1} \mid \tag{2.70}$$

Utregningen av x_f vises i formel 2.71.

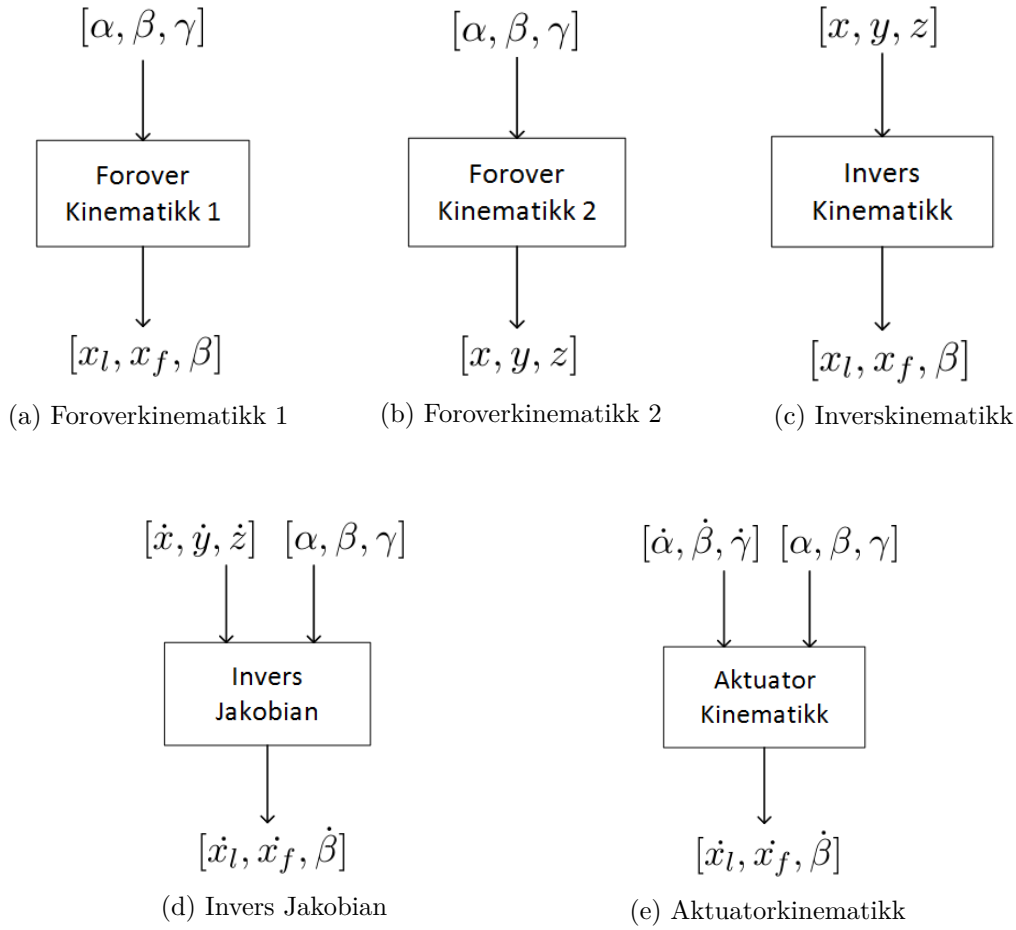
$$x_f = \mid s'_{f_1 f_2} \mid = \mid s'_{kf_2} - s'_{kf_1} \mid \tag{2.71}$$

3. Resultater

Dette kapittelet vil beskrive og forklare løsningen og resultatene til den banestyrte rørhåndteringskranen. Først vil kinematikken forklares og kategoriseres for enklere å forstå resten av system arkitekturen. De viktigste delene av løsningen vil gjennomgås i detalj ved hjelp av blokkdiagrammer og figurer. Dette kapittelet vil kun presentere løsningen, diskusjon og reflektering kommer i neste kapittel.

3.1 Kinematikk

Kinematiske beregninger har en sentral rolle i denne oppgaven. Utledning av kinematikken er beskrevet i kapittel 2.6. Ut i fra de formelene ble det laget kinematiske regneblokker som benyttes i systemet. Blokkene er delt inn som vist i figur 3.1.



(a) Foroverkinematikk 1 (b) Foroverkinematikk 2 (c) Inverskinematikk

(d) Invers Jakobian (e) Aktuatorkinematikk

Figur 3.1: Oversikt over kinematikkblokkene

Hver blokk har et sett med parametere som inngangssignal og som utgangssignal. Det som er

nytt i forhold til tidligere masteroppgave [1], er at det har blitt opprettet en blokk for aktuatorkinematikk. Denne blokken benyttes når man har vinkelhastighet som tilbakemelding fra kranen, i motsetning til ventilposisjon. Utgangssignalene er aktuatorhastighetene, og sendes videre til kontrolleren.

Alle ligninger som benyttes i kinematikkblokkene er på algebraisk form, siden PLSen ikke kan regne med matriser. Et eksempel vises i formel 3.1, som er en del av invers jakobian-blokken.

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} (\xi_a + \xi_c) & \xi_c \\ (\xi_b + \xi_d) & \xi_d \end{bmatrix}^{-1} \cdot \begin{bmatrix} v_x \cdot \cos\beta + v_y \cdot \sin\beta \\ v_z \end{bmatrix} \tag{3.1}$$

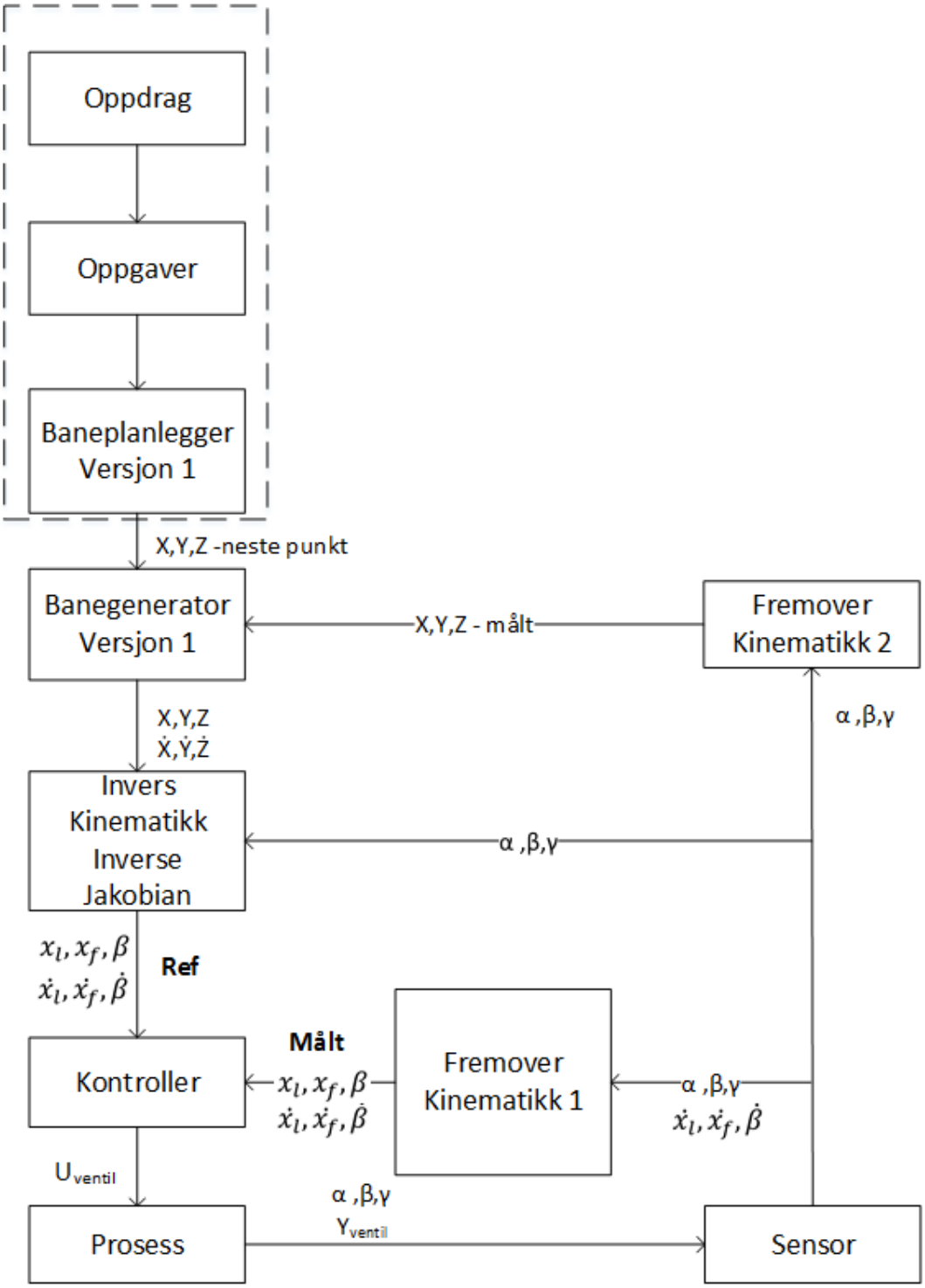
De algebraiske ligningene som beskriver denne regneoperasjonen er vist i formel 3.2 til 3.9.

$$v_{x\,MB} = v_x \cdot cos\beta + v_y \cdot sin\beta \tag{3.2}$$
$$v_{z\,MB} = v_z \tag{3.3}$$
$$j_{11} = \xi_a + \xi_c \tag{3.4}$$
$$j_{12} = \xi_c \tag{3.5}$$
$$j_{21} = \xi_b + \xi_d \tag{3.6}$$
$$j_{22} = \xi_d \tag{3.7}$$
$$\dot{\alpha} = \frac{j_{22} \cdot v_{x\,MB}}{j_{11} \cdot j_{22} - j_{12} \cdot j_{21}} - \frac{j_{12} \cdot v_{z\,MB}}{j_{11} \cdot j_{22} - j_{12} \cdot j_{21}} \tag{3.8}$$
$$\dot{\gamma} = \frac{j_{11} \cdot v_{z\,MB}}{j_{11} \cdot j_{22} - j_{12} \cdot j_{21}} - \frac{j_{21} \cdot v_{x\,MB}}{j_{11} \cdot j_{22} - j_{12} \cdot j_{21}} \tag{3.9}$$

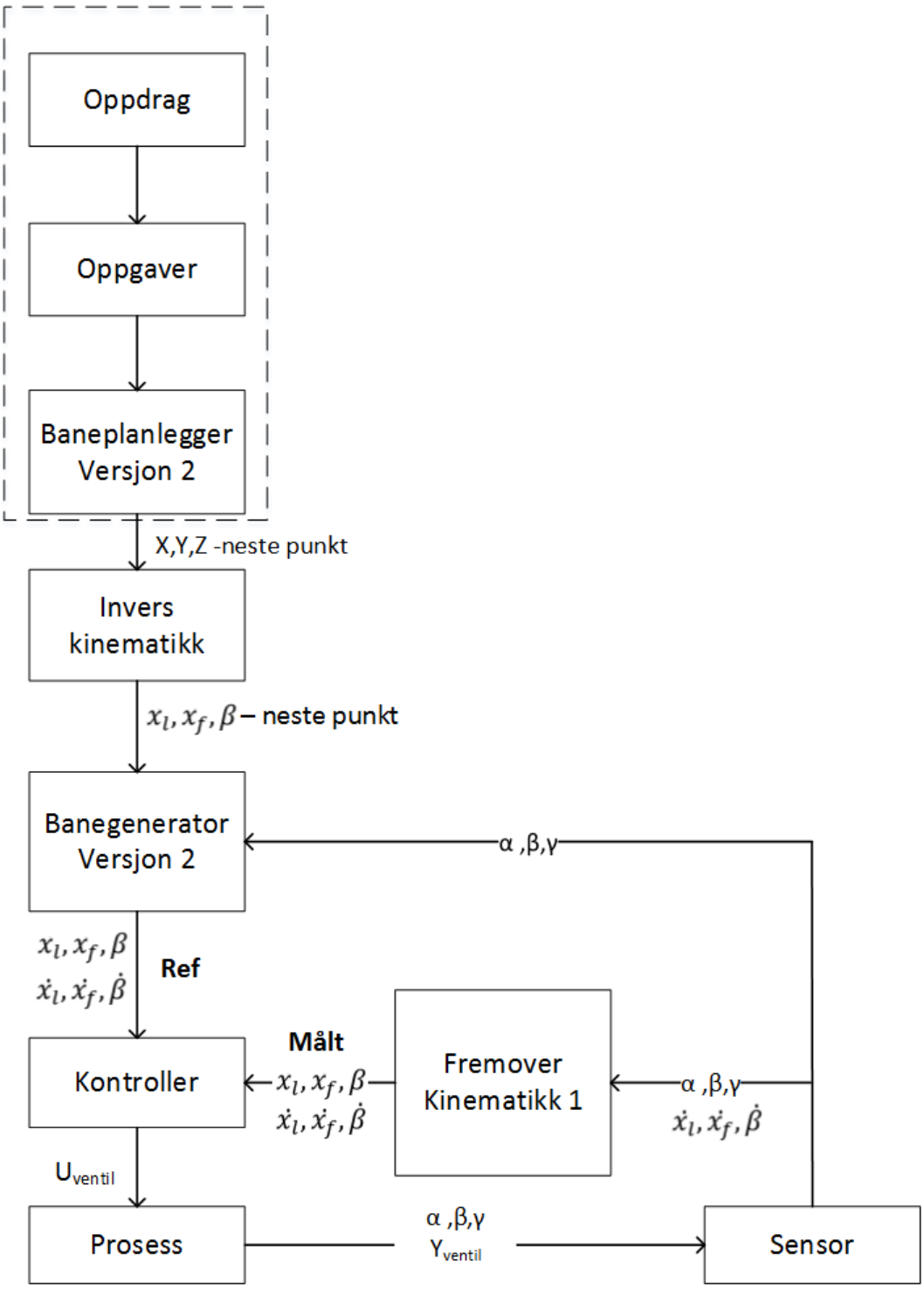
Ved å gjøre noen mellomregninger så blir formelene relativt ryddige. Alle de kinematiske ligningene er omgjort til algebraisk form i MATLAB, og ligger i vedlegg B.4.

3.2 Systemarkitektur

Systemarkitekturen beskriver egenskapene til et system. Hvordan systemarkitekturen ser ut er avhengig av interessent eller *stakeholder*. Her kan *The* $4+1$ *View Model of Software Architecture* [17] benyttes for å beskrive systemet, men i denne rapporten blir *Logical View* og *Process View* brukt sammen for å forklare hvordan brukeren vil oppleve systemet samt hvordan informasjonsflyten foregår. Den viser alt fra hvor operatøren gir sin kommando til hvordan PLSen og kranen kommuniserer. Systemarkitekturen i denne oppgaven presenteres som et blokkdiagram. Det har blitt utviklet to løsninger, med hver sin systemarkitektur. Banegeneratoren i system 1 jobber i xyz-rommet, og er en videreutvikling av banegeneratoren som ble laget av mastergruppen i 2013. Ved å bruke kinematikkblokker kan xyz-koordinatene transformeres til aktuatorposisjon, som sendes videre ned i systemet. System 2 inneholder en helt ny banegenerator, og jobber i aktuatorenes domene. Ved å jobbe i aktuatorenes domene vil man ha mer kontroll over systemet. Systemarkitekturen er designet slik at man har et modulært design. Figur 3.2 og 3.3 viser systemarkitekturen for system 1 og system 2, sett fra et *top-down* perspektiv. Stiplede linjer viser blokker som er utenfor denne oppgaven sitt omfang.



Figur 3.2: System 1



Figur 3.3: System 2

Over er forskjellen på de to systemene vist samt hvordan de forskjellige delsystemene er koblet sammen. System 1 er avhengig av flere tilbakemeldingssignaler enn system 2. Dette vil si at hvert delsystem i system 2 er mer selvstendig. Kontroller, Prosess og Sensor er lik i begge systemer, men rekkefølgen på blokkene i delsystemene er forskjellig. System 2 benytter en kinematikkblokk tidligere enn i system 1, som gjør at man jobber i aktuatorenes domene på et tidligere stadie.

For å beskrive hvert delsystem sin funksjonalitet, så er eksempel på et brukstilfelle laget for å gi en oversikt. I dette eksempelet er det kranfører som definerer kommandoen til "Oppdragblokken.
Oppdrag: Kranen skal flytte et rør fra posisjon x til posisjon y
Oppgaver: Lokaliser rør, kjør bort til x, plukk opp rør, kjør bort til y, legg ned rør, kjør tilbake til startposisjon
Baneplanlegger: Bestemmer hvor kranen må bevege seg for å inngå hindringer. Den lager et sett med punkter som kranen skal kjøre gjennom
Banegenerator: Genererer en bane i xyz eller aktuatorlengder som kranen skal følge basert på ønsket hastighetsprofil. Inngangssignalet til denne blokkene er neste ønsket punkt i xyz eller aktuatorlengde, utgangssignalet er posisjon og hastighets referanser som funksjon av tid for xyz eller aktuatorrommet.
Kontroller: Bruker banen som referanse og bestemmer pådraget til kranens aktuatorer. Inngangssignalet til denne blokken er aktuator posisjon og hastighets både referanse og målt verdi. Utgangssignalet er ventilpådrag.
Sensor: Måler vinkler og ventilåpning. I konteksten til denne oppgaven har denne blokken også en omregning fra ventilåpning til sylinderfart.

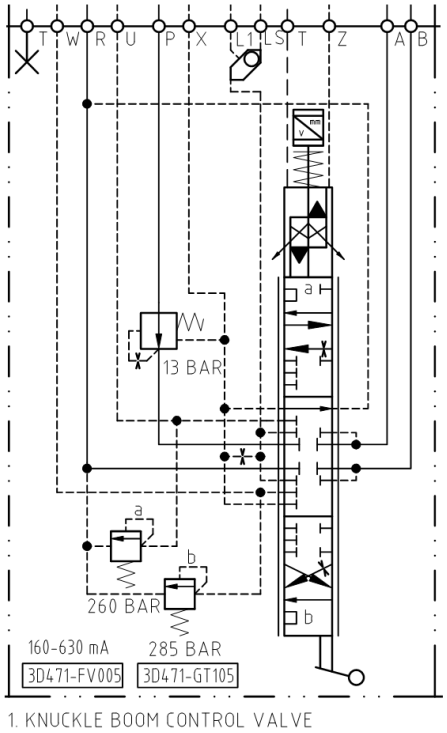
Deler av systemarkitekturen er utenfor arbeidsomfanget til denne masteroppgaven. Oppdrag og deloppgaver som er markert i figur 3.2 og 3.3 er ikke implementert i denne oppgaven. Løsningen som blir presentert i dette kapittelet baserer seg derfor på en liste med xyz-koordinater som kranen skal følge. Ved videre utvikling av systemarkitekturen vil operatøren kunne gi langt mer avanserte kommandoer til systemet, slik som eksempelet over. Selv om deler av systemarkitekturen ikke er utviklet har det blitt lagt stort vekt på å gjøre løsningen enkel å implementere videre. Fleksibel kode med mulighet for gjenbruk har vært i fokus.

3.3 Systemmodellering

En dynamisk modell av kranen laget i SimulationX var tilgjengelig ved prosjektstart. Det ble allikevel brukt en del tid på å modifisere modellen. Hensikten var både å optimalisere modellen for HiL-simulering, men også fordi tidligere modell manglet blant annet trykkompenserte servoventiler og lastholdeventiler. Siden trykkompensering på servoventilene er en viktig del av kontrollsystemet var det viktig å ha en modell som hadde denne funksjonaliteten.

3.3.1 Hydraulisk modellering og simulering

Det hydrauliske systemet er modellert etter hydraulikktegningene på en pipehandler med prosjektnummer G9619. Kranbevegelsene styres av en ventilblokk med fire trykkompenserte servoventiler. Teleskopbevegelsen til kranen er ikke en del av fokuset til oppgaven, på grunn av dette er det bare tre av ventilene som vil bli benyttet. I figur 3.4 vises servoventilen som styrer knekkbommen.



Figur 3.4: Trykkompensert servoventil som benyttes

Figuren viser at trykkompensatoren er stilt inn på 13 bar. Servoventilen har også to innebyggede sikkerhetsventiler for å unngå eventuelt overtrykk. Innstilt trykk på sikkerhetsventilene er vist i tabell 3.1.

Tabell 3.1: Innstilling på sikkerhetsventilene

	Knekkbom	Hovedbom	Sving
A-side	260 bar	280 bar	230 bar
B-side	285 bar	130 bar	230 bar

I tillegg er to av servoventilene usymmetriske, de gir forskjellig volumstrøm avhengig om de er åpnet mot A-siden eller B-siden. Dette vil motvirke arealforholdet på sylindrene, som vil si at sylindrene får lik hastighet i begge retninger. Volumstrømmen ved full ventilåpning og 13 bar trykkfall er vist under i tabell 3.2.

Tabell 3.2: Maksimal volumstrøm til servoventilene

	Knekkbom	Hovedbom	Sving
A-side	170 l/min	160 l/min	210 l/min
B-side	85 l/min	80 l/min	210 l/min

Denne oppførselen kan modelleres i SimulationX ved å velge *different edges* på ventilen. Man oppgir da et referansetrykkfall, og kan videre definere referansevolumstrøm på hver at de fire åpningene P-A, P-B, A-T og B-T. I SimulationX kan man også beskrive ventildynamikken med hensyn til ventilposisjon. Ofte er den beskrevet med en andregrads transferfunksjon. Databladet for servoventilene var ikke tilgjengelige så en båndbredde på 30 Hz, og en dempefaktor på 0.9 ble satt, noe som er typisk på en servoventil.

Selv om kranen kun har én pumpe for å levere volumstrøm til ventilene, har pumpen blitt modellert som tre trykkilder, en til hver ventil. Dette er en forenkling som er vanlig å gjøre, og sålenge man holder seg under maksimal volumstrøm som pumpen kan levere vil denne forenklingen være veldig lik faktisk system på kranen. Trykkilden styres av en funksjon som holder konstant trykk over hver ventil. Løftetrykket er vist i formel 3.10.

$$p_{lofte} = \begin{cases} p_A + p_{valve} & \text{når } y_{ventil} < 0 \\ p_B + p_{valve} & \text{når } y_{ventil} \geq 0 \end{cases} \tag{3.10}$$

Hvor;

p_{valve}	=	13 bar
p_A	=	trykket på A-siden
p_B	=	trykket på B-siden
y_{ventil}	=	ventilposisjonen

Det er også satt en begrensning på pumpetrykket. Nedre grense er 0 bar, og øvre grense er satt til 310 bar, likt som sikkerhetsventilen på pumpen på kranen.

Systemet inneholder også lastholdeventiler og oversenterventiler for å hindre kavitasjon, noe som kan forekomme ved negativ last. Lastholdeventilene er styrt med et ekstern pilottrykk, mens oversenterventilene benytter trykket på motsatt side av aktuatoren for å regulere ventilåpningen. På denne kranen er det brukt lastholdeventiler på A-siden på løftesylinder og A-siden på foldesylinder. Oversenterventiler er blitt brukt på A- og B-sidene til motoren, og B-siden på foldesylinder. B-siden på løftesylinder har ingen slik ventil fordi den ikke vil oppleve negativ last. På hver ventil var det oppgitt et innstilt mottrykk, p_{crack}. Pilotarealforholdet ρ har blitt estimert for å oppnå ønsket respons i modellen, og for å holde trykket i de utsatte delene på minst 20 bar for å forhindre kavitasjon. I formel 3.11 til 3.17 vises uttrykkene for ventilene når de er i statisk likevekt, og har 0 % åpning.

$$pilot_{lofte} = \begin{cases} 0 \text{ bar} & \text{når løftesylinder beveges inn} \\ 130 \text{ bar} & \text{når løftesylinder beveges ut} \end{cases} \tag{3.11}$$

$$pilot_{folde} = \begin{cases} 0 \text{ bar} & \text{når foldesylinder beveges inn} \\ 130 \text{ bar} & \text{når foldesylinder beveges ut} \end{cases} \tag{3.12}$$

$$motor_A + \rho_1 \cdot motor_B = p_{crack_1} \tag{3.13}$$

$$motor_B + \rho_2 \cdot motor_A = p_{crack_2} \tag{3.14}$$

$$loftesyl_A + \rho_3 \cdot pilot_{lofte} = p_{crack_3} \tag{3.15}$$

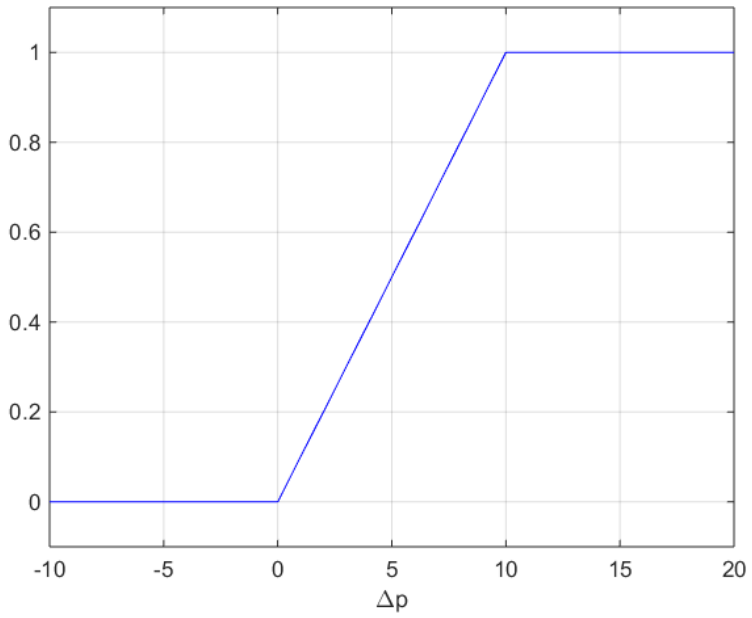
$$foldesyl_A + \rho_4 \cdot pilot_{folde} = p_{crack_4} \tag{3.16}$$

$$foldesyl_B + \rho_5 \cdot foldesyl_A = p_{crack_5} \tag{3.17}$$

Hvor;

$pilot_{lofte}$	=	Eksternt pilottrykk for løftesylinder
$pilot_{folde}$	=	Eksternt pilottrykk for foldesylinder
$motor_A$	=	Trykket på A-siden i motoren
$motor_B$	=	Trykket på B-siden i motoren
$loftesyl_A$	=	Trykket på A-siden i løftesylinderen
$loftesyl_B$	=	Trykket på B-siden i løftesylinderen
$foldesyl_A$	=	Trykket på A-siden i foldesylinderen
$foldesyl_B$	=	Trykket på B-siden i foldesylinderen
ρ_1	=	2
ρ_2	=	2
ρ_3	=	1.7
ρ_4	=	2.2
ρ_5	=	2
p_{crack_1}	=	230 bar
p_{crack_2}	=	230 bar
p_{crack_3}	=	320 bar
p_{crack_4}	=	310 bar
p_{crack_5}	=	310 bar

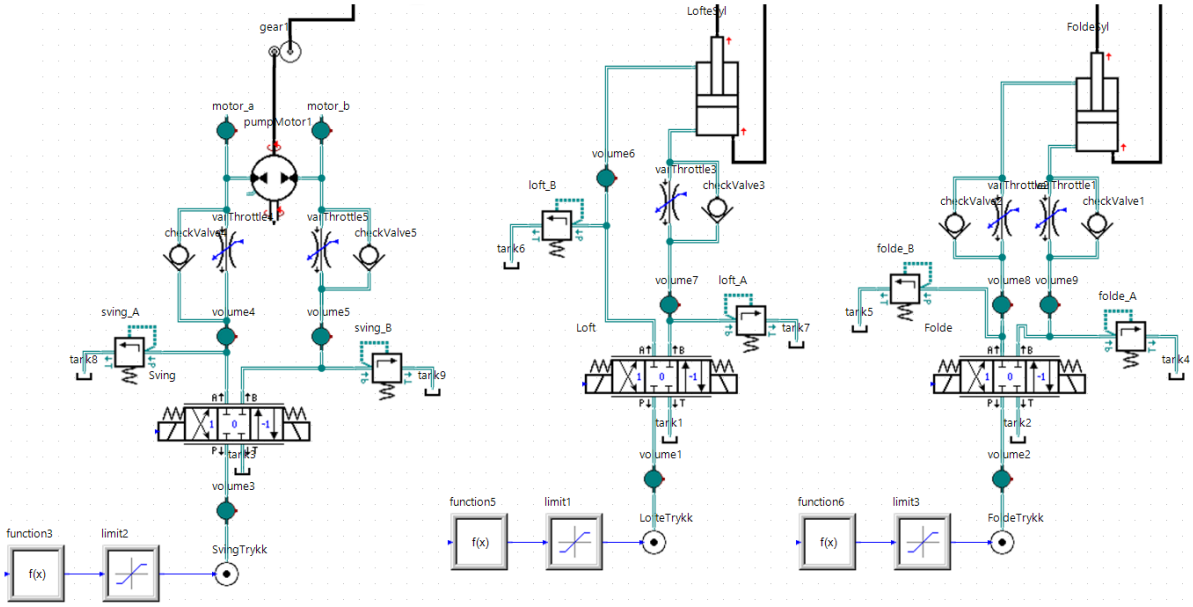
Ut i fra disse formlene kan man lage et normalisert signal som beskriver åpningen på ventilene. Ventilene begynner å åpne seg når aktuatortrykkene er større enn mottrykket p_{crack}, og det har blitt modellert slik at de er helt åpne når trykkdifferansen er 10 bar. Dette er vist i figur 3.5. 3.6.



Figur 3.5: Normalisert ventilåpning på lastholdeventiler og oversenterventiler

Ved å ha en gradvis ventilåpning så vil ventilene kunne strupe volumstrømmen ved å ha en liten åpning når systemet har negativ last.

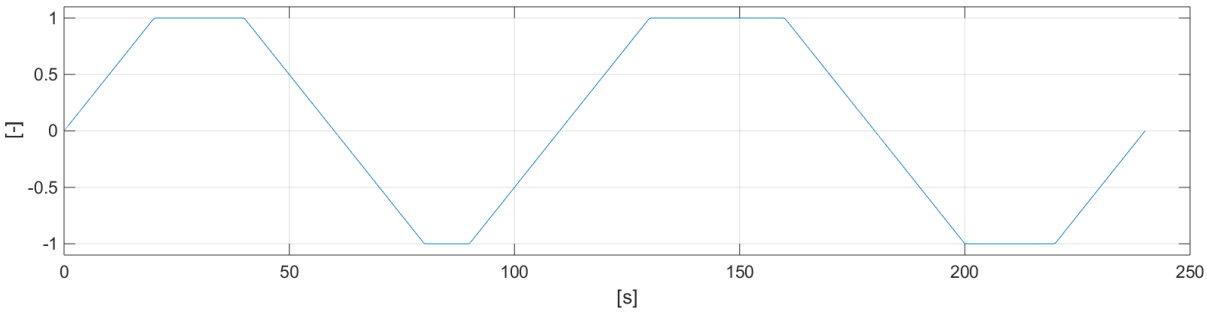
En oversikt over det hydrauliske systemet i SimulationX vises i figur 3.6.



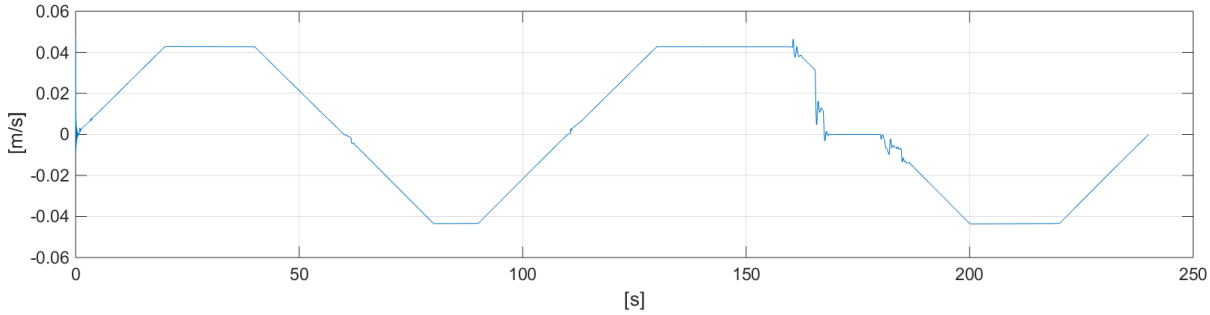
Figur 3.6: Hydraulikk modellert i SimulationX

Som vist over, så har systemet blitt modellert med tre trykkilder, tre servoventiler og fem lastholde- og oversenterventiler. Små volumer har blitt lagt inn for å modellere rør og slanger. De små volumene vil også forhindre skarpe trykkgradienter i systemet.

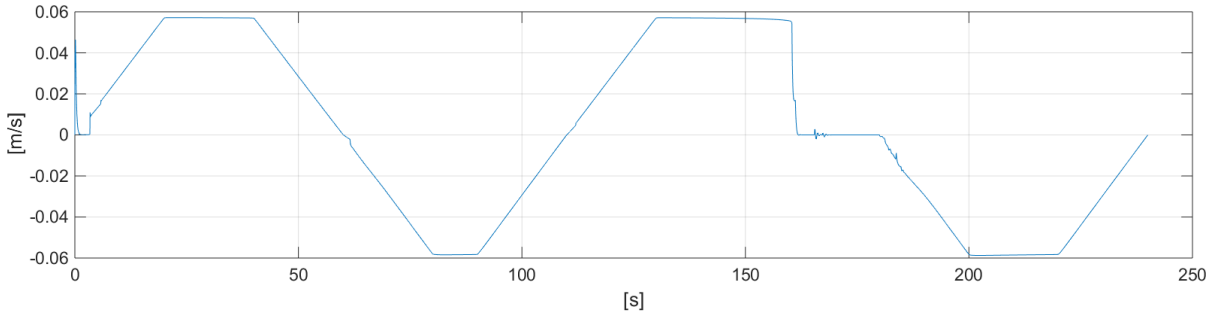
De viktigste funksjonene som skal testes er de trykkompenserte servoventilene og lastholde- og oversenterventilene. I tillegg må man forsikre seg om at alle trykkene i systemet er stabile, og at hverken trykket eller aktuatorene oscillerer. Et referansesignal til servoventilene har blitt laget for å kunne teste all funksjonalitet over et stort arbeidsområde. Foldesylinderen er spesielt viktig å teste, fordi den kan oppleve negativ last både når den beveges ut og inn. Figur 3.7 viser referansesignalet og aktuatornes hastighet.



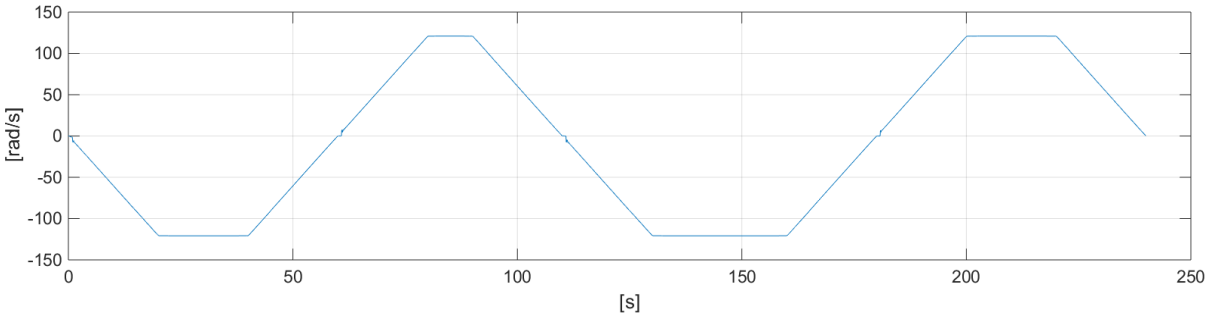
(a) Referansesignal



(b) Løftesylinderens hastighet



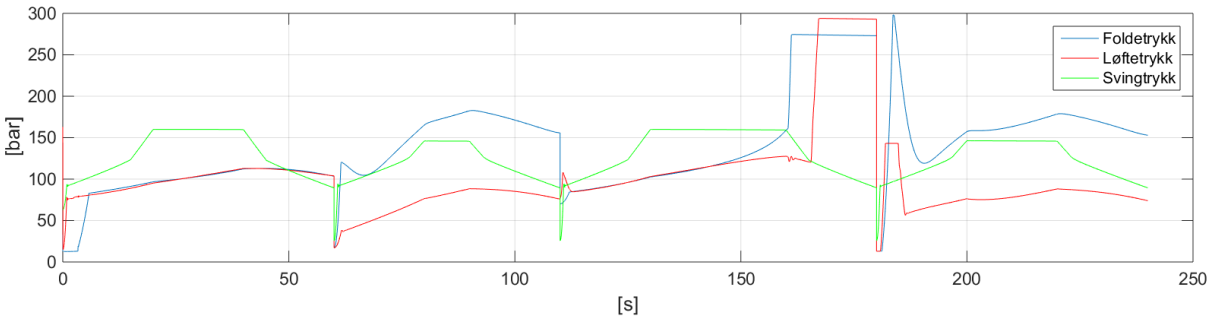
(c) Foldesylinderens hastighet



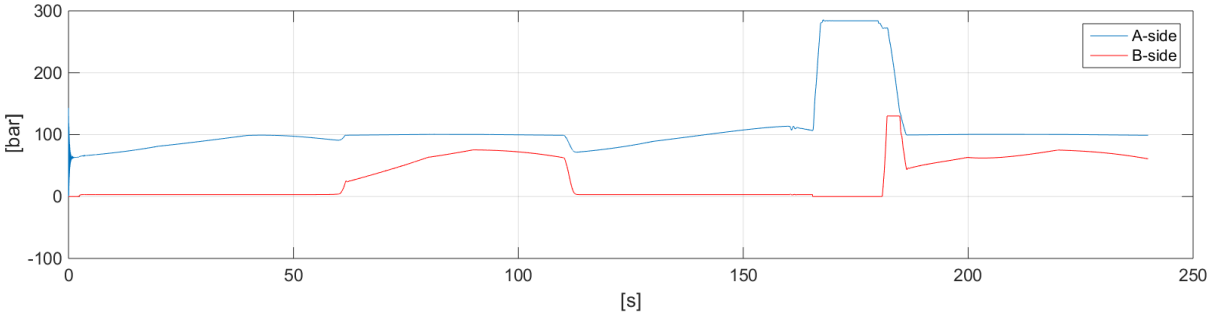
(d) Motorens hastighet

Figur 3.7: Referansesignal og aktuatorhastigheter

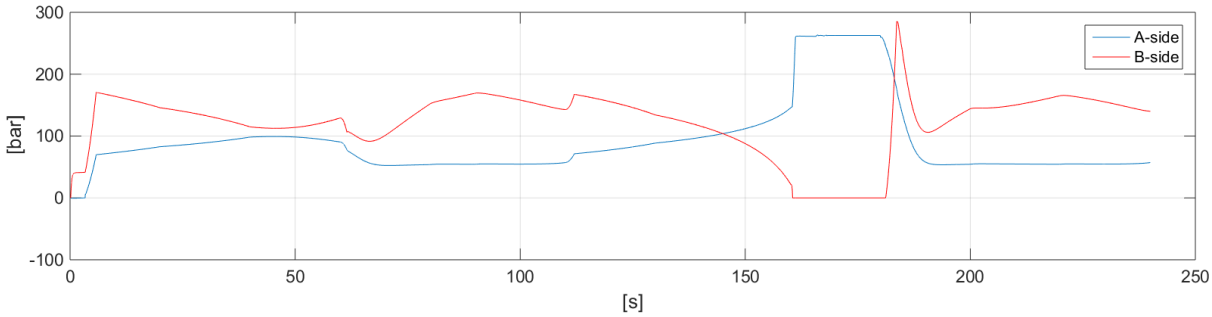
Som figuren over viser er aktuatornes hastighet proporsjonal med referansesignalet, som betyr at trykkompenseringen fungerer. Foldesylinderen og løftesylinderens hastighet går til 0 når de kommer til endestoppet. Når dette skjer vil sikkerhetsventilene åpne seg og slippe gjennom volumstrømmen. De forskjellige trykkene i systemet er vist i figur 3.16.



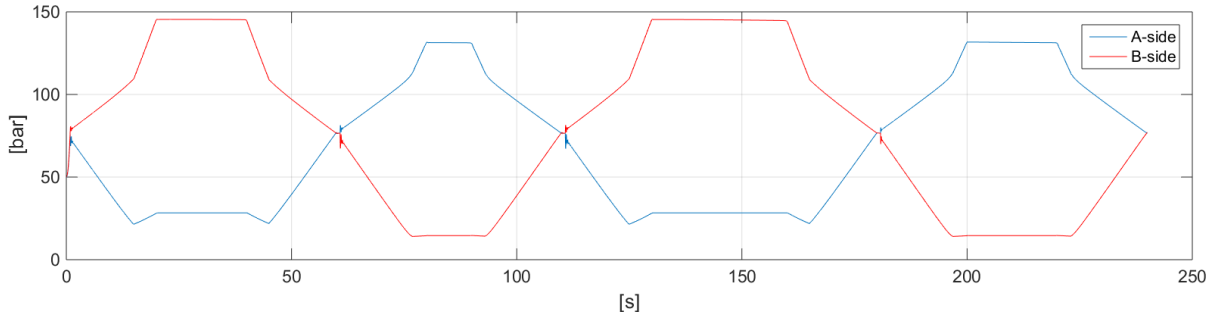
(a) Trykk i trykkildene



(b) Trykk i løftesylinderen



(c) Trykk i foldesylinderen



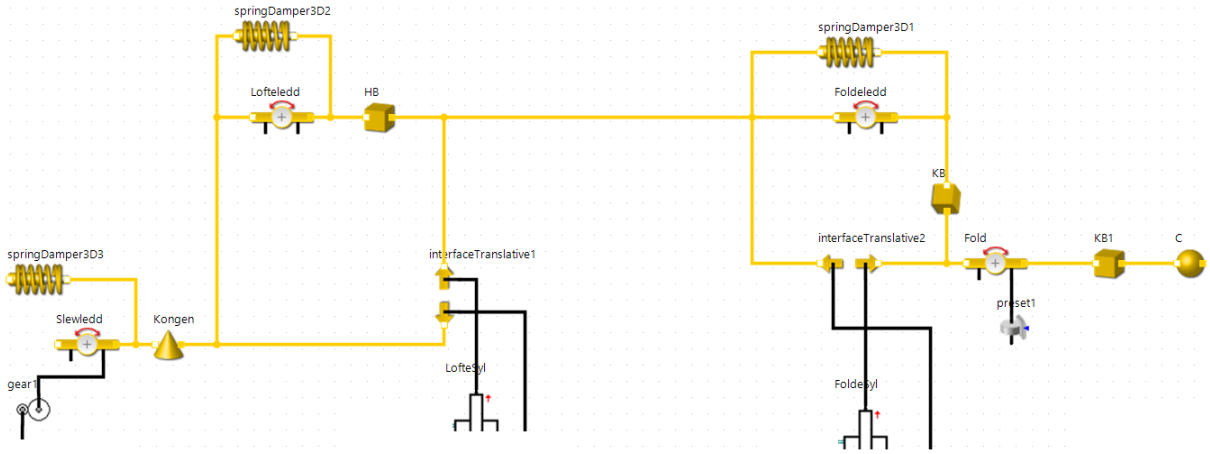
(d) Trykk i motoren

Figur 3.8: Trykk i systemmodellen

Som figuren viser er det ingen oscillasjoner i trykkene, som vil si at systemet er veldig stabilt. Trykket i trykkildene er alltid 13 bar over aktuatortrykket for å holde konstant trykk over servoventilen. Aktuatortrykkene går aldri under 20 bar ved negativ last. Når løftesylinderen og foldesylinderen kommer til endestoppet så øker trykket til den grensen som er satt i sikkerhetsventilene.

3.3.2 Mekanisk modellering og simulering

De mekaniske systemet har også blitt litt modifisert i forhold til tidligere masteroppgave [1]. Pidestallen og bommene har blitt modellert som stive legemer, hvor massen har blitt satt slik at den stemmer med massene på den fysiske kranen. Alle leddene mellom pidestall, hovedbom og knekkbom er modellert som rotasjonsledd. I tillegg så har en demper blitt festet til leddene for å modellere friksjon. Uten disse demperne så opplevde modellen mye svingninger, som blant annet gikk utover stabiliteten til det hydrauliske systemet. Dempekoeffisienten har blitt satt til $10^7 \frac{Nms}{rad}$ for å holde systemet stabilt. På kranen finnes det to lukkede mekaniske sløyfer, én til hver sylinder. Disse sløyfene inneholder både et rotasjonsledd og et translasjonsledd. Det ble brukt et *Bipolar Force Interface* for å lukke sløyfen og for å modellere den hydrauliske sylinderen sin tilkobling til det mekaniske systemet. Dette elementet er spesialdesignet for å kunne koble hydraulikksylindere til tredimensjonale mekaniske systemer. Pidestallen har også et gir mellom leddet og den hydrauliske motoren. Girforholdet har blitt estimert til 1900 for å stemme overens med loggede vinkelhastigheter. Figur 3.9 viser mekanikken i SimulationX



Figur 3.9: Mekanikk modellert i SimulationX

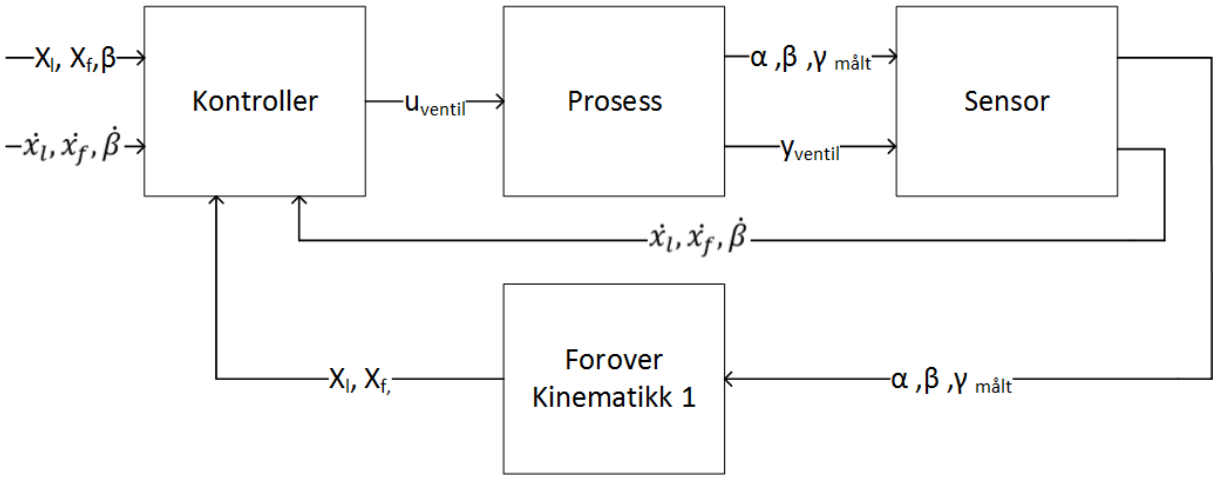
Over viser figuren at et lite element kalt C, har blitt lagt til. Dette er krantuppen, koordinatene til dette elementet brukes til å blant annet a teste kinematikken som er utviklet og for å observere krantuppens posisjon når systemet kjører en HiL-simulering. Et fiksert ledd har også blitt lagt inn for å lage avbøyningen på knekkbommen.

3.4 Kontroller

Kontrolleren mottar referansesignal fra banegeneratoren. Referansesignalene er aktuatornes posisjon og hastighet, dette gjør at kontroller kan brukes uavhengig om man benytter banegenerator versjon 1 eller versjon 2. Kontroller er siste ledd i systemarkitekturen, og sender utgangssignaler til kranens servoventiler.

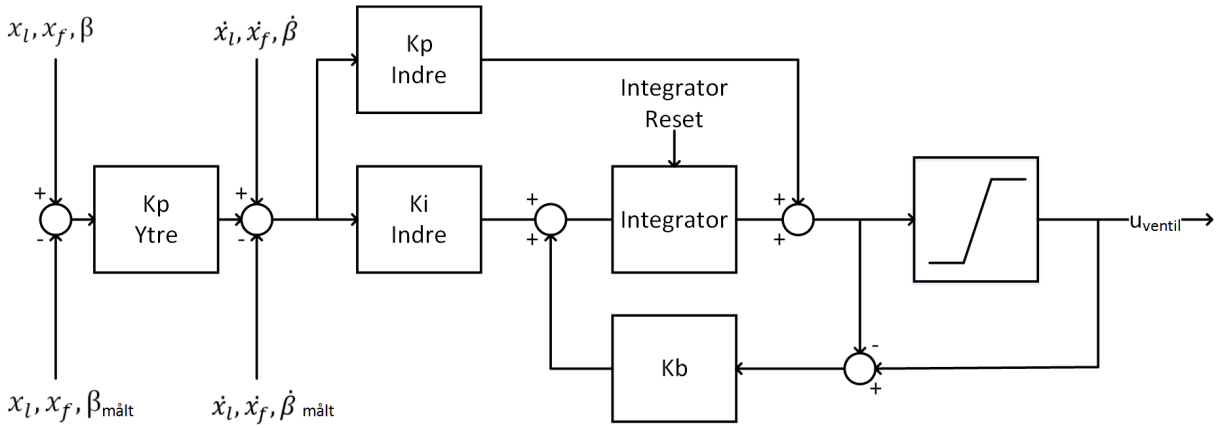
3.4.1 Kontrollarkitektur

Figur 3.10 viser den overordnede arkitekturen til kontroller.



Figur 3.10: Kontrollerens arkitektur

Som vist i figur 3.10 mottar kontrolleren også tilbakemelding på aktuatorenes posisjon og hastighet. Figur 3.11 viser et mer detaljert blokkdiagram av kontrolleren.

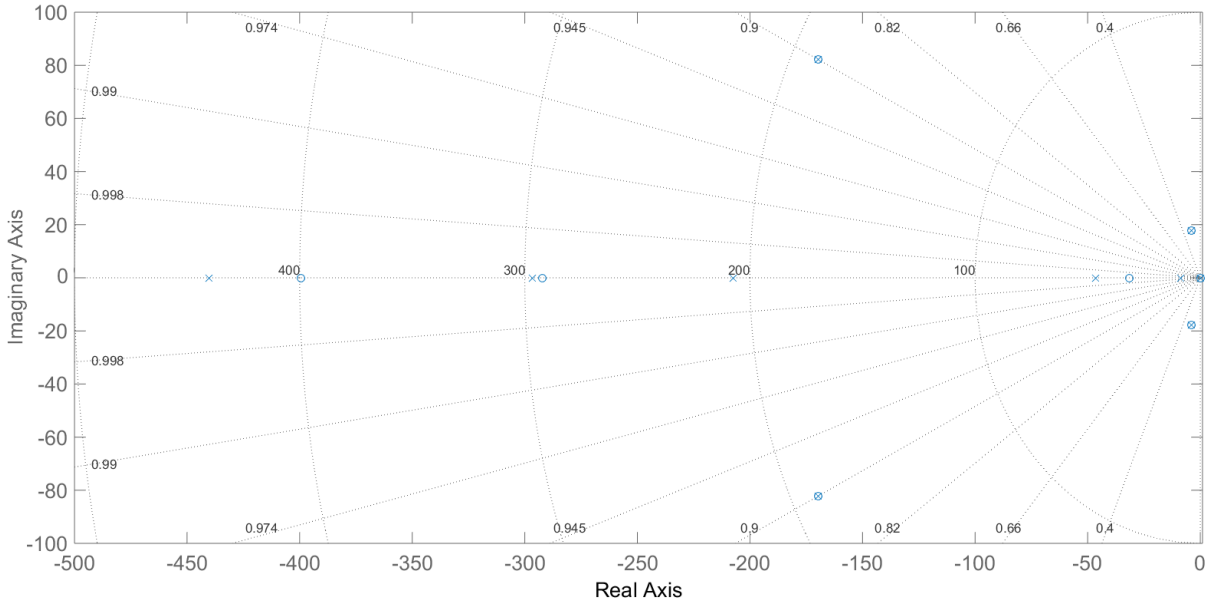


Figur 3.11: Blokkdiagram av kontrolleren

Kontrolleren er bygget opp som en kaskadekontroller med en indre hastighetssløyfe med PI-regulator og en ytre posisjonssløyfe med P-regulator. Den ytre sløyfen trenger ingen integrator, siden systemet er et type-1 system sett med hensyn til posisjonen. I tillegg har *anti windup* blitt implementert som forsikrer god ytelse når ventilene går i metning. En annen funksjon som har blitt laget, er en integrator reset. Denne funksjonen resetter integratoren til 0 når den aktiveres. Dette kan være fordelaktig når referansen endrer retning, som kan skje når kranen skal kjøre til et nytt punkt i rommet. Kontrolleren vil da reagere raskere siden integratoren ikke trenger å integrere fra for eksempel en positiv verdi til en negativ verdi.

3.4.2 Innregulering av kontrolleren

For å innregulere kontrolleren har transferfunksjon for hver aktuator blitt estimert. Disse transferfunksjonene kommer fra systemmodellen som har blitt laget i SimulationX. Modellen ble eksportert som en S-function, og lagt inn i Simulink. Linear Analysis Tool i Simulink ble brukt for å finne poler og nullpunkter til hver aktuator. Figur 3.12 viser poler og nullpunkter til løftesylinderen.



Figur 3.12: Poler og nullpunkter fra Simulink

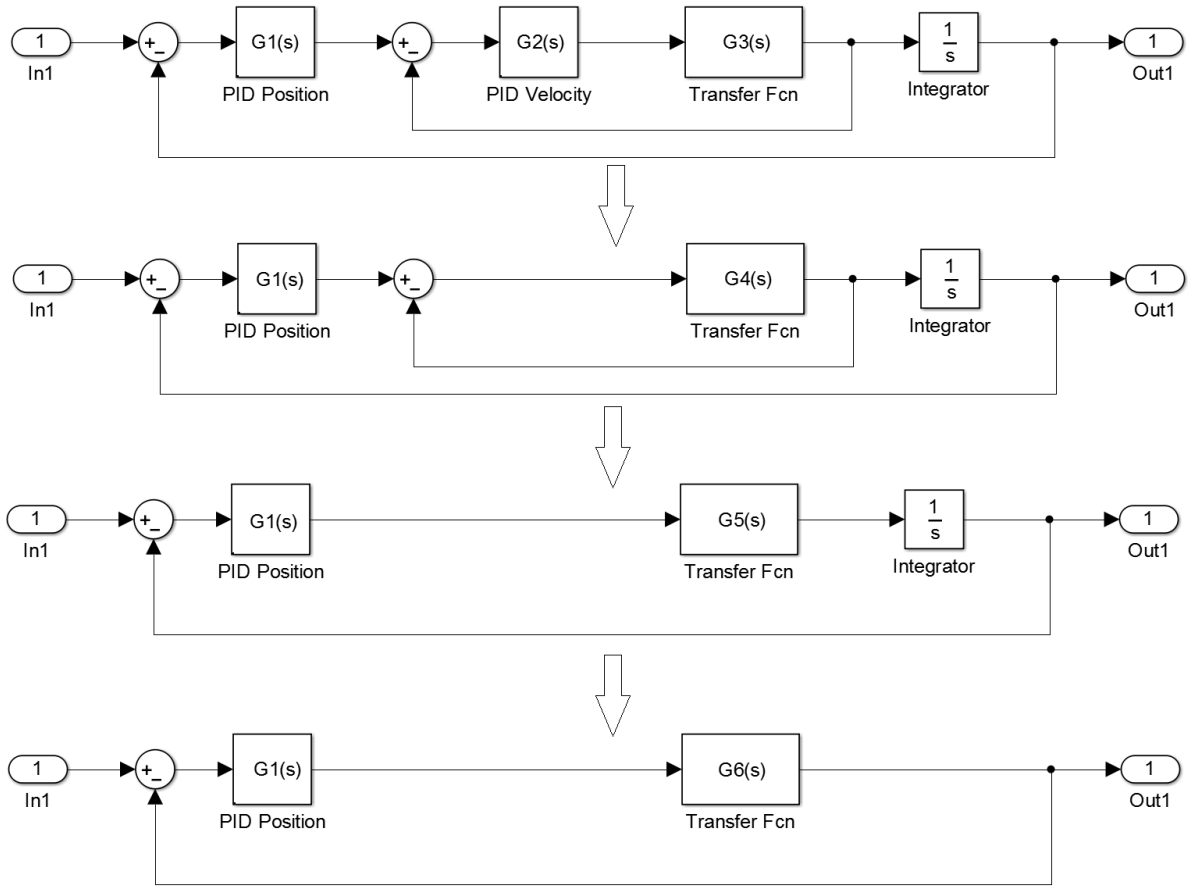
Linear Analysis Tool ga noen poler som ble antatt å være urealistiske, for eksempel poler med flere kHz båndbredde. I tillegg viste Linear Analysis Tool noen poler på samme sted som nullpunkter, disse ble også ignorert. Resultatet ble brukt til å lage en *zero pole gain*-modell av hver aktuator. Forsterkningen ble estimert slik at responsen stemmer overens med resultatet fra SimulationX. Transferfunksjonene vises i formel 3.18, 3.19 og 3.20

$$G_{sving} = \frac{7.5328e5}{(s^2 + 8.18s + 330)(s^2 + 340s + 3.566e4)} \tag{3.18}$$

$$G_{lofte} = \frac{1.4014e9(s + 399)(s + 292)(s + 31.6)}{(s + 208)(s + 297)(s + 440)(s + 46.4)(s + 8.69)(s^2 + 8.18s + 330)(s^2 + 340s + 3.566e4)} \tag{3.19}$$

$$G_{folde} = \frac{1.0578e9(s + 297)(s + 227)(s + 81.4)}{(s + 208)(s + 297)(s + 440)(s + 46.4)(s + 8.69)(s^2 + 8.18s + 330)(s^2 + 340s + 3.566e4)} \tag{3.20}$$

Disse transferfunksjonene beskriver aktuatorhastigheten som funksjon av inngangssignalet til servoventilen. Nå som transferfunksjonene er estimert kan MATLAB PID Tuner brukes til først å innregulere indre sløyfe, deretter ytre sløyfe. Figur 3.13 viser hvordan man kan lage en transferfunksjon som beskriver den innregulerte indre sløyfen sett fra den ytre sløyfen.



Figur 3.13: Innregulering i ytre og indre sløyfe

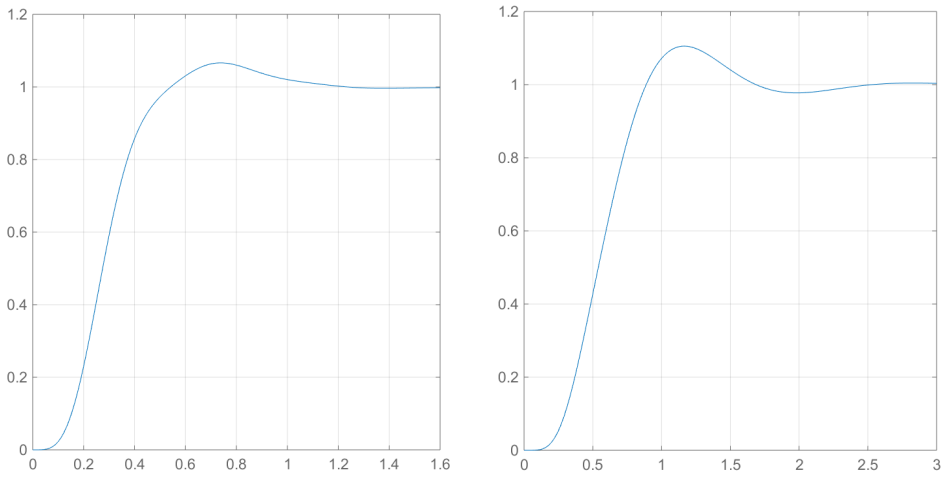
Formlene under viser hvordan dette gjøres matematisk.

$$G_4(s) = G_2(s) \cdot G_3(s) \tag{3.21}$$

$$G_5(s) = \frac{G_4(s)}{1 + G_4(s)} \tag{3.22}$$

$$G_6(s) = G_5(s) \cdot \frac{1}{s} \tag{3.23}$$

MATLAB PID Tuner kjøres to ganger per aktuator, for å innregulere $G_3(s)$ og $G_6(s)$. Figur 3.14 viser innregulert stegrespons på indre og ytre sløyfe for foldesylinderen.



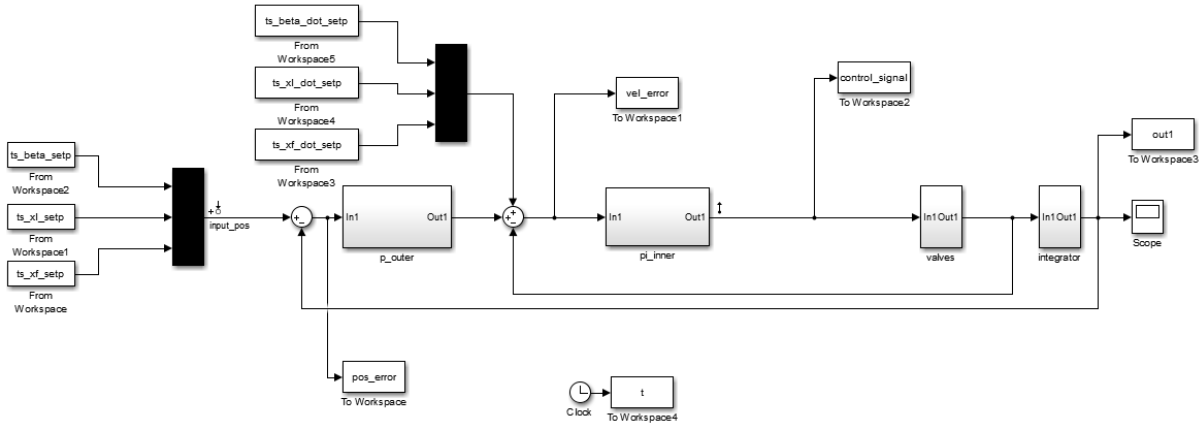
(a) Stegrespons indre sløyfe

(b) Stegrespons ytre sløyfe

Figur 3.14: Stegresponser på indre og ytre sløyfe for foldesylinderen

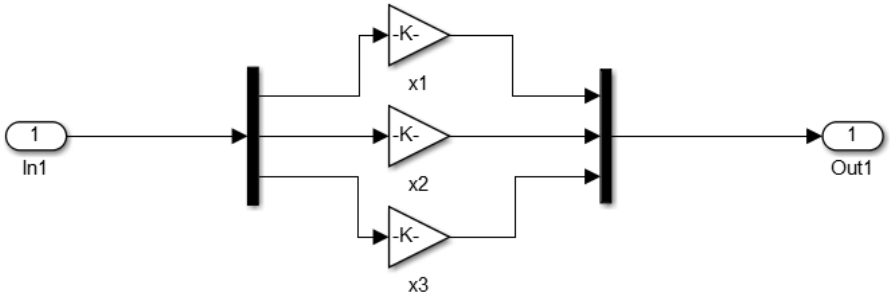
Det tillates litt oversving for å få en rask respons på systemet. Dette ble gjort på alle tre aktuatorene.

Optimaliseringsalgoritmene som er beskrevet i kapittel 2.4 ble også brukt til å innregulere systemet etter de estimerte transferfunksjonene. Kontrolleren og systemets transferfunksjon er lagt inn i en Simulink modell og brukes som målfunksjon til optimaliseringen. Oppgaven til algoritmen er å finne laveste sum av posisjonsfeilen til aktuatorene ved å endre parameterne på kaskadekontrolleren. Figur 3.15 viser hvordan optimaliseringen er satt opp i Simulink.

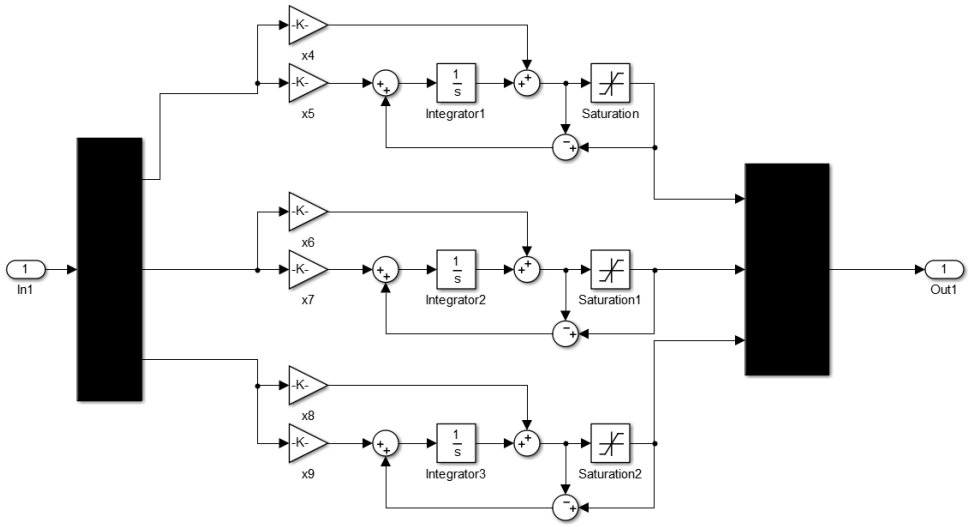


Figur 3.15: Simulinkmodell av kontrolleren

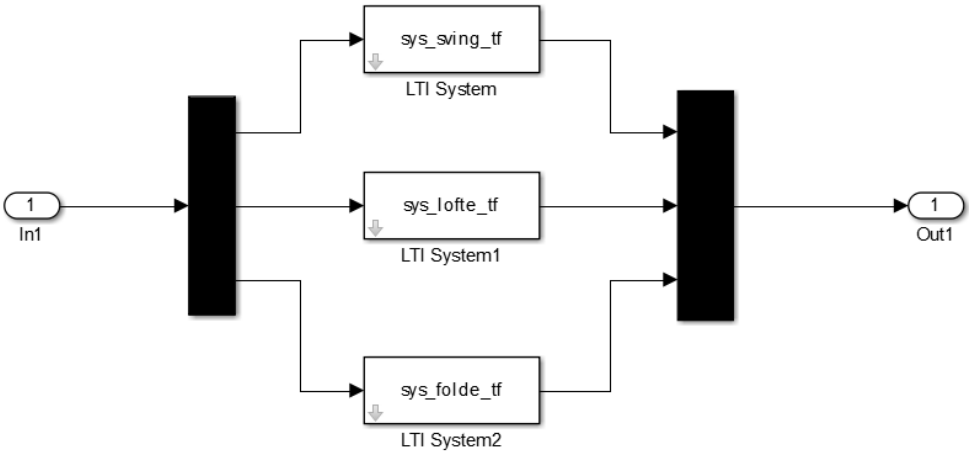
Kontrolleren får tidsserier for aktuatorposisjon og hastighet som referanse. Dette er ikke bare en *stepinput*, men en bane som kunne vært brukt i et reelt tilfelle. Målet med optimaliseringen er å finne de tre parameterne for kaskadekontrolleren for alle tre aktuatorer altså $x_1, .., x_9$ som vist i figur 3.16a og 3.16b. Figur 3.16c viser innholdet i *valves* blokken fra figur 3.15, dette er transferfunksjonene til ventilene som vist i ligning 3.18, 3.19 og 3.20



(a) P-kontroller for posisjon



(b) PI-kontroller for hastighet



(c) Estimerte transferfunksjoner for ventilene

Figur 3.16: Detaljert blokkdiagram av kontroller og estimert transferfunksjon

Alle fire algoritmene ble kjørt to ganger, modus A og modus B. Forskjellen er hvordan øvre og nedre grense på parameterene er satt. Tabell 3.16 viser hvordan grensene i de to modusene er satt.

Tabell 3.3: Alle metoder

Modus	Grense	K_{pytre}	K_{pindre}	K_{iindre}
Modus A	Nedre	1	10^{-4}	10^{-4}
	Øvre	10	10	1000
Modus B	Nedre	10^{-4}	10^{-4}	10^{-4}
	Øvre	1000	1000	1000

Grensene til kontrollparameterene er like for alle tre aktuatorer. Modus A har sine begresninger hovedsaklig fra PID tuner i MATLAB, men også manuell testing. PID tuner satte aldri K_{pytre} under 1 eller over 10. K_{pindre} og K_{iindre} henger tilsynelatende sammen, de øker og synker i verdi sammen. Dersom K_{iindre} var over 1000 ble integreringen alt for aggresiv og resultatne ble ikke bra. Dette skjedde når K_{pindre} nærmet seg 10, så derfor ble denne parameteren også begrenset. Nedre grense på indre sløyfe er satt for å ikke ha null i gain, som vil være det samme som å ikke utnytte hele kontrollstrukturen.
Modus B er en optimalisering mer eller mindre uten begresninger. Kravet er å holde seg til et positivt tall og under 1000 som manuell testing viste uansett ikke gav bra resultat. Tabell 3.4 viser parameterne til kontrolleren etter optimaliseringen, PID tuner og semimanuell tuning, referer med figur 3.16a og 3.16b for å se betydningen av $x_1, .., x_9$.

Tabell 3.4: Alle kontrollparametere

Metode	Modus	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
Fmincon	A	9.80	3.19	3.21	5.80	73.24	6.62	203.10	9.88	213.32
	B	4.37	8.77	10.11	7.87	108.2	12.38	109.12	11.68	85.15
Fminsearch		8.85	2.40	1.64	6.34	75.95	4.29	260.78	0.33	223.61
GA	A	9.63	3.88	3.61	5.77	75.34	9.97	198.59	9.98	198.76
	B	10.88	7.42	5.38	5.79	66.77	56.45	126.60	202.84	931.59
Patternsearch	A	10	3	3	5.80	72	10	255.68	10	218.56
	B	10.84	3.99	4.06	5.80	66.74	15.5	211	15.07	188.50
PID Tuner		2	2	2	1	108	1	109	1	85
Semimanuell		2.41	1.46	1.16	8.68	602.2	8.678	998.6	9.95	985.48

Startverdiene på de algortimene som krever det er vist i Tabell 3.5. Startverdiene er parameterene funnet via MATLAB PID tuner.

Tabell 3.5: Startverdier på kontrollparametere

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
2	2	2	1	108	1	109	1	85

Optimaliseringsalgoritmene kjøres med standardinnstillinger. Optimaliseringen kjører helt til verdien av målfunksjonen går under en toleransegrense eller den ikke finner en bedre løsning. Det er uklart hvilken metode som vil ha best ytelse før HiL-testing med de forskjellige parameterne har blitt gjort.

3.5 Baneplanlegger

Baneplanleggeren er et av de delsystemene som er høyest oppe i systemarkitekturen. Utviklingen av dette delsystemet er utenfor arbeidsomfanget til denne oppgaven. Den må allikevel tas hensyn til da de lavere nivåene av systemarkitekturen er avhengig av inngangssignaler fra dette delsystemet. Siden denne blokken skal motta kommandoer fra blokker som ikke er i denne oppgavens arbeidsomfang så er funksjonaliteten begrenset. I denne oppgaven er baneplanleggeren kun en liste med punkter som kranen skal innom. Disse punktene skrives direkte inn i PLS koden.

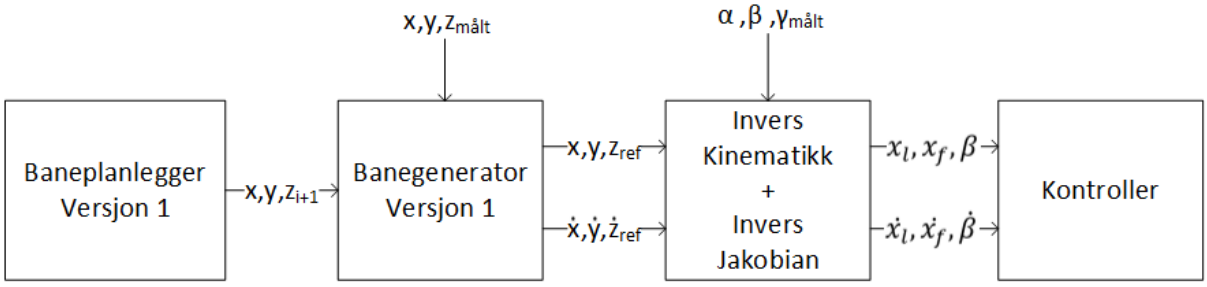
Ved fremtidig arbeid må denne blokken oppdateres for å kunne motta kommandoer fra delsystemet over. Baneplanleggeren skal få instrukser fra *oppgaver* blokken. Baneplanleggeren skal finne ut om det er noen faste hindringer eller bevegelige maskiner i arbeidsrommet før den lager en bane som den videresender til banegeneratoren.

3.6 Banegenerator

Banegeneratoren er programet som lager inngangsreferansen til kontrolleren. Den baserer seg på en liste med koordinater som den skal innom. Disse punktene representerer en operasjon som for eksempel er å hente en last ved et punkt og levere lasten ved et annet punkt. I denne oppgaven kommer koordinatene fra en liste i PLS koden, men ved videre utvikling av systemet og system arkitekturen vil operatøren som styrer kranen kunne velge banen eller oppdrag via HMI. Dette delkapittelet vil forklare forskjellen på de to banegeneratorene som har blitt utarbeidet i denne oppgaven.

3.6.1 Banegenerator versjon 1

Banegenerator versjon 1 gir referanser i $\mathbb{R}^3$, altså xyz og $\dot{x}\dot{y}\dot{z}$. Disse referansene regnes om til sylinderlengde og sylinderfart som igjen brukes som inngangssignal på kaskadekontrolleren. Denne banegeneratoren gjør at kranen følger en rett linje i $\mathbb{R}^3$, noe som kan være ønskelig i visse operasjoner. Denne banegeneratoren baserer seg på tidligere arbeid fra mastergruppen i 2013, [1]. Det antas at arbeidsrommet er fritt for hindringer i denne oppgaven. En ønsket hastighetsprofil blir laget mellom to punkter, og deretter integreres hastighetsreferansen for å lage en posisjonsreferanse. Videre i underkapittelet vil detaljene rundt hvordan banegeneratoren fungerer vises. Figur 3.17 viser et blokkdiagram av banegenerator versjon 1 og relevante blokker.



Figur 3.17: Blokkdiagram av Banegenerator versjon 1

Inn i baneplanleggeren kommer en liste med punkter kranen skal kjøre gjennom. Disse skrives

inn i en matrise som vist under i ligning 3.24.

$$L = \begin{bmatrix} x_{init} & y_{init} & z_{init} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \tag{3.24}$$

Første rad er startposisjonen til kranen og rad 2 til n er punktene som kranen må kjøre innom. Posisjonsvektoren mellom hvert punkt beregnes som vist i ligning 3.25.

$$\vec{s_i} = \begin{bmatrix} s_{i_x} \\ s_{i_y} \\ s_{i_z} \end{bmatrix} = \begin{bmatrix} x_i - x_{i-1} \\ y_i - y_{i-1} \\ z_i - z_{i-1} \end{bmatrix} \tag{3.25}$$

Lengden mellom hvert punkt må beregnes for å finne enhetsvektorene. Lengden beregnes ved hjelp av Pytagoras' læresetning som vist i ligning 3.26.

$$|\vec{s_i}| = \sqrt{s_{i_x}^2 + s_{i_y}^2 + s_{i_z}^2} \tag{3.26}$$

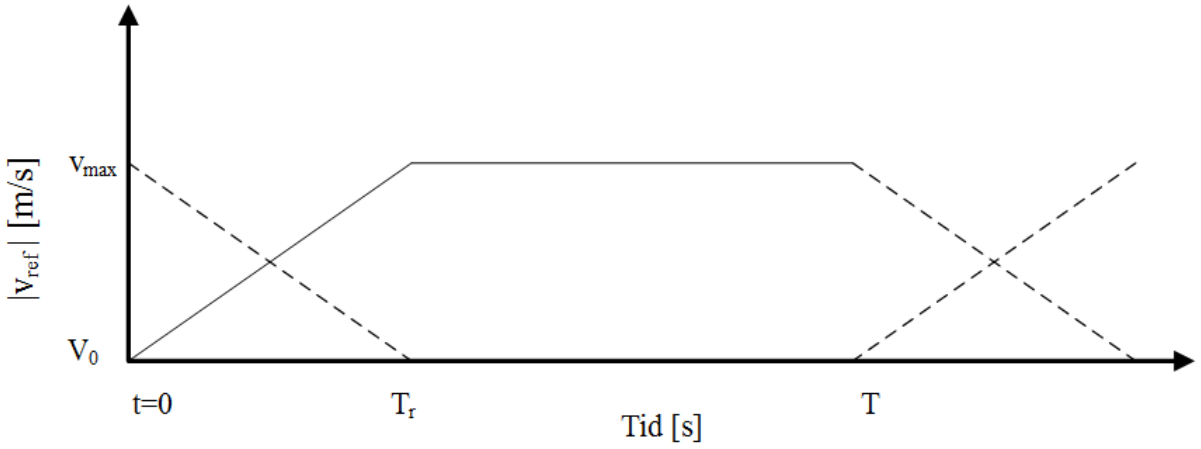
Tiden mellom hvert punkt gis av ligning 3.27.

$$T_i = \frac{|\vec{s_i}|}{v_{max}} \tag{3.27}$$

Enhetsvektoren mellom hvert punkt er gitt av ligning 3.28.

$$\vec{e_i} = \frac{\vec{s_i}}{|\vec{s_i}|} \tag{3.28}$$

Får å unngå høy akselerasjon og retardasjon ved start og stopp er det ønskelig at hastighetsprofilen følger en trapesform som vist i figur 3.18. Trapesen er delt opp i tre deler. Akselerasjon, konstant fart maks fart og retardasjon.



Figur 3.18: Hastighetsreferansen skal følge en trapes

Banegeneratoren vil ikke rampe ned farten før den har nådd et settpunkt, deretter vil den rampe opp farten mot neste punkt. Unntaket er helt i starten og helt i slutten hvor det kun er akselerasjon og retardasjon henholdsvis. Akselerasjonen a er beregnet som vist i ligning 3.29.

$$a = \frac{v_{max}}{T_r} \tag{3.29}$$

hvor;

v_{max} = Maksimal hastighet i $\mathbb{R}^3$
T_r = Rampetid

T_r er en konstant satt til 10 sekunder som i tidligere masteroppgave [1]. Under akselerasjon vil hastigheten være som vist i ligning 3.30.

$$v_a = at \tag{3.30}$$

Under retardasjon er hastigheten som vist i ligning 3.31

$$v_d = v_{max} - a\,(t - T) \tag{3.31}$$

Hvor v_{max} er maksimal fart til krantuppen i $\mathbb{R}^3$. Denne farten kan beregnes ved hjelp av kranens hydrauliske kapasitet eller finnes eksperimentelt. Maksfarten er satt til $v_{max} = 0.15\ m/s$ som satt i tidligere masteroppgave [1].

Hastighetsreferansen v_{ref} beregnes ved å skalere enhetsvektoren avhengig av hvor på dette trapeset man er.

$$v_{ref} = \begin{cases} \vec{e}_i v_a & \text{Akselerasjon} \\ \vec{e}_i v_{max} & \text{Konstant fart} \\ \vec{e}_i v_d & \text{Retardasjon} \end{cases} \tag{3.32}$$

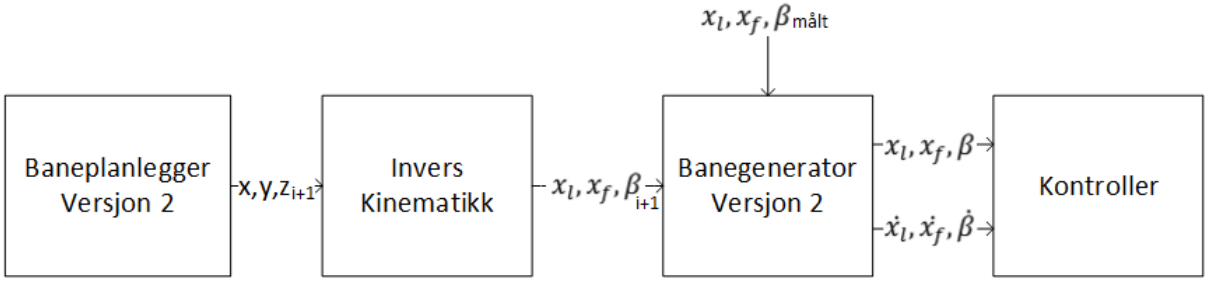
Posisjonsreferansen beregnes ved å integrere hastighetsreferansen som vist i ligning 3.33

$$x_{ref} = \int_0^t v_{ref} dt \tag{3.33}$$

Nå er xyz og $\dot{x}\dot{y}\dot{z}$ referansene tilgjengelig og kan sendes videre til neste blokk i systemet.

3.6.2 Banegenerator versjon 2

Banegenerator versjon 2 jobber i det såkalte aktuatorrommet, og gir referanser i aktuatorposisjonene x_l, x_f, β og aktuatorhastighetene $\dot{x}_l$, $\dot{x_f}$, $\dot{\beta}$. Dette er forskjellig fra versjon 1, som gir xyz koordinater i $\mathbb{R}^3$. Istedenfor å regne ut en rett linje i $\mathbb{R}^3$, finner versjon 2 en rett linje mellom nåværende aktuatorposisjon og ønsket aktuatorposisjon som korrisponderer til ønsket xyz punkt. Siden kinematikken er ulineær vil krantuppen ikke bevege seg lineært i $\mathbb{R}^3$ når aktuatorene følger en rett linje i aktuatorrommet. Den har derimot en tendens til å bevege seg i bue mellom to punkter. En viktig forutsetning er at det ikke er noen hindringer i arbeidsområdet. Med denne metoden unngår man også at aktuatorene følger en unødvendig lang bane, noe som vil spare energi. Figur 3.19 viser blokkdiagrammet av versjon 2 samt relevante blokker.



Figur 3.19: Blokkdiagram av Banegenerator versjon 2

Siden koordinatene fremdeles er gitt i XYZ må nåværende og ønsket aktuatorposisjon beregnes ved hjelp av inverskinematikk. Når banegeneratoren bruker aktuatorposisjon og aktuatorhastighet er det mye lettere å beregne v_{max}. Her er v_{max} maksimal fart for hver av aktuatorene. Alle tre ventilene har trykk-kompensering som gjør oljestrøm proporsjonal med aktuatorhastighet.

Banegenerator versjon 2 har i tillegg flere funksjonaliteter enn versjon 1, som er laget for å sikre god ytelse for systemet. Når kranen skal svinge rundt pidestallen til en nytt punkt vil det alltid være to løsninger, rotere mot klokka eller med klokka. Med mindre kranen skal rotere 180° vil en av retningene være kortere enn den andre. Korteste vei for svingbevegelsen, $\Delta\beta_2$, beregnes og brukes videre. Indeksen til $\Delta\beta$ skriver i hvilken rekkefølge den er regnet ut. Ligning 3.34 viser hvordan $\Delta\beta_1$ beregnes.

$$\Delta\beta_1 = \beta_{ref} - \beta_{målt} \tag{3.34}$$

Korteste vei regnes ut i ligning 3.35.

$$\Delta\beta_2 = \begin{cases} \Delta\beta_1 - 2\pi & \Delta\beta_1 > \pi \\ \Delta\beta_1 + 2\pi & \Delta\beta_1 < -\pi \\ \Delta\beta_1 & -\pi \leq \Delta\beta_1 \leq \pi \end{cases} \tag{3.35}$$

Det er ikke alltid mulig å ta den korteste veien, kranen brukt i denne oppgaven og de fleste rørhåndteringskraner har en begrensning på hvor mye de kan rotere før den må stoppe for å hindre at hydraulikkslangene skades. Maksimal rotasjon er lagt inn både som en boolsk av/på-funksjon og som maksimal og minimum svingvinkel β_{max}, β_{min}.

$$\Delta\beta_3 = \begin{cases} \Delta\beta_2 - 2\pi & \Delta\beta_2 > \beta_{max} \\ \Delta\beta_2 + 2\pi & \Delta\beta_2 < \beta_{min} \\ \Delta\beta_2 & \beta_{min} \leq \Delta\beta_1 \leq \beta_{max} \end{cases} \tag{3.36}$$

Ligning 3.35 og 3.36 sikrer at kranen svinger den korteste veien når det er mulig.

For å hjelpe kontrolleren slik kranen kan ta igjen referansen er det lagt inn en begrensning på maksfarten til aktuatorene, $K_1 = 0.75$. Dette vil redusere farten, men kranen vil kunne ta igjen referansen hvis den henger etter.
Maksimal fart for hver aktuator blir beregnes som vist i ligning 3.38 og 3.39. Maksimal fart, v_{max}, regnes ut for hver aktuator og settes i en vektor for å forenkle notasjonen slik som vist i ligning 3.37.

$$\vec{v}_{max_1} = \begin{bmatrix} v_{max_{1_{Sving}}} \\ v_{max_{1_{Lofte}}} \\ v_{max_{1_{Folde}}} \end{bmatrix} \tag{3.37}$$

$$v_{max_{1_{Sving}}} = \frac{K_1 Q_{sving}}{D_{motor} i_{gir}} \tag{3.38}$$

hvor;

Q_{sving}	=	Maksimal oljestrøm på svingaktuator
i_{gir}	=	Girforhold mellom motor og svingkrans
D_{motor}	=	Fortrengningsvolum til motoren

$$v_{max_{1_{Lofte/Folde}}} = \begin{cases} \frac{K_1 \cdot Q_{inn}}{A \cdot \phi} & \text{Sylinder inn} \\ \frac{K_1 \cdot Q_{ut}}{A} & \text{Sylinder ut} \end{cases} \tag{3.39}$$

hvor;

Q_{inn} = Maksimal oljestrøm når folde- eller løftesylinder går inn.
Q_{ut} = Maksimal oljestrøm når folde- eller løftesylinder går ut.
A = Innvendig areal til sylinderen
ϕ = Arealforholdet til sylinderen

Det er også her ønskelig å følge en trapes hastighetsprofil som i versjon 1, men hastighetsprofilen er for aktuatorene og ikke krantuppen. I tillegg vil det ikke være noe overlapp på trapeset slik som i figur 3.18. Som nevnt over kan maksimal fart regnes ut fra hydrauliske begrensninger, men maksimal akselerasjon a_{max} kan også settes for å forsikre en glatt bevegelse. Maksimal akselerasjon er satt til $a_{max} = 0.03\ m/s^2$. Denne er satt på bakgrunn fra [18], hvor følgende kriterie skal opprettholdes, vist i formel 3.40.

$$T_r \geq \frac{6}{\omega_n} \tag{3.40}$$

Hvor;

ω_n = Egenfrekvensen til systemet, estimert til 18.2 [rad/s] i MATLAB

Opprampingstiden T_r er beregnet som vist i ligning 3.41, dette regnes ut for alle tre aktuatorer.

$$\vec{T}_{r_1} = \begin{bmatrix} T_{r_{1_{Sving}}} \\ T_{r_{1_{Lofte}}} \\ T_{r_{1_{Folde}}} \end{bmatrix} = \frac{\vec{v}_{max_1}}{a_{max}} \tag{3.41}$$

Distansen hver aktuator skal traversere er Δs, vist i ligning 3.42.

$$\vec{\Delta s} = \begin{bmatrix} \Delta s_{Sving} \\ \Delta s_{Lofte} \\ \Delta s_{Folde} \end{bmatrix} = \begin{bmatrix} \Delta\beta_3 \\ x_{l_{ref}} - x_{l_{målt}} \\ x_{f_{ref}} - x_{f_{målt}} \end{bmatrix} \tag{3.42}$$

Tiden det tar for hele bevegelsen, T_1, er gitt ved ligning 3.43, dette regnes ut for alle tre aktuatorer.

$$\vec{T}_1 = \begin{bmatrix} T_{1_{Sving}} \\ T_{1_{Lofte}} \\ T_{1_{Folde}} \end{bmatrix} = \frac{\vec{\Delta s} + \vec{T}_{r_1}\vec{v}_{max_1}}{\vec{v}_{max_1}} \tag{3.43}$$

Siden det kan være en begrensing på akselerasjonen må man akseptere at hastighetsprofilen ikke alltid blir en trapes. Dersom a_{max} er lav og distansen man skal traversere er kort vil det ikke være tid til å akselerere helt opp til v_{max}. Når dette skjer er $T_{r1} \geq \frac{T_1}{2}$. Da vil profilen bli diskontinuerlig og uendelig retardasjon vil skje ved $t = T_{r1}$. Da må hastighetsprofilen settes til en trekantprofil. Formel 3.45, 3.46 og 3.48 viser justeringen som må gjøres for å oppnå trekantprofil. For å finne ny totaltid, T_2, må man bruke bevegelsesligningen for konstant akselerasjon.

$$\Delta x = v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \tag{3.44}$$

Hvor;

Δx = lengden som skal traverseres, $\frac{1}{2}\vec{\Delta s}$
v_0 = starthastighet, satt til 0
a = a_{max}
t = $\frac{T}{2}$

Ut ifra dette kan man regne ut en ny totaltid, avhengig om man har en trekantprofil eller ikke, vist i formel 3.45.

$$\vec{T_2} = \begin{bmatrix} T_{2_{Sving}} \\ T_{2_{Lofte}} \\ T_{2_{Folde}} \end{bmatrix} = \begin{cases} \left(\frac{4\vec{\Delta s}}{a_{max}}\right)^{0.5} & \text{Hvis} \quad \vec{T}_{r_1} \geq \frac{T_1}{2} \\ \vec{T}_1 & \text{Hvis} \quad \vec{T}_{r_1} < \frac{T_1}{2} \end{cases} \tag{3.45}$$

En ny rampetid kan også regnes ut, vist i formel 3.46.

$$\vec{T}_{r_2} = \begin{bmatrix} T_{r_{2_{Sving}}} \\ T_{r_{2_{Lofte}}} \\ T_{r_{2_{Folde}}} \end{bmatrix} = \begin{cases} \frac{\vec{T}_2}{2} & \text{Hvis} \quad \vec{T}_{r1} \geq \frac{\vec{T}_1}{2} \\ \vec{T}_{r_1} & \text{Hvis} \quad \vec{T}_{r_1} < \frac{\vec{T}_1}{2} \end{cases} \tag{3.46}$$

Mest sannsynlig vil de tre tidene i T_2 være forskjellige. For at alle akuatorene skal bli ferdig samtidig settes perioden til den største verdien, vist i formel 3.47.

$$T_{max_1} = \max(\vec{T}_2) \tag{3.47}$$

Konsekvensen av dette er at en ny makshastighet, $\vec{v}_{max_2}$, må regnes ut på ny som vist i ligning 3.48

$$\vec{v}_{max_2} = \begin{bmatrix} v_{max_{2_{Sving}}} \\ v_{max_{2_{Lofte}}} \\ v_{max_{2_{Folde}}} \end{bmatrix} = \frac{\vec{\Delta s}}{T_{max_1} - \vec{T}_{r_2}} \tag{3.48}$$

Nå som maks hastighet er satt, kan nødvendig oljestrøm i beregnes, vist i formel 3.49, 3.50 og 3.51.

$$\vec{Q} = \begin{bmatrix} Q_{Sving} \\ Q_{Lofte} \\ Q_{Folde} \end{bmatrix} \tag{3.49}$$

$$Q_{Sving} = \frac{\vec{v}_{max_2} D_{motor} i_{gir}}{104.7} \quad \text{[l/min]} \tag{3.50}$$

$$Q_{Lofte/Folde} = \begin{cases} \frac{3}{50}\vec{v}_{max_2} A\phi & \text{Sylinder inn [l/min]} \\ \frac{3}{50}\vec{v}_{max_2} A & \text{Sylinder ut [l/min]} \end{cases} \tag{3.51}$$

Det blir nå introdusert en ny sikkerhetsfaktor, K_2. Den sjekker om nødvendig oljestrøm overstiger maksimal tilgjengelig oljestrøm fra pumpen. Ligning 3.52 viser hvordan K_2 beregnes.

$$K_2 = \frac{Q_{pumpe}}{Q_{Sving} + Q_{Lofte} + Q_{Folde}} \qquad K_2 \in [0, 1] \tag{3.52}$$

K_2 blir begrenset i programkoden til å ha en maksimalverdi på 1. Dersom K_2 er mindre enn 1 vil den øke tiden aktuatorene bruker på bevegelsen. Dersom K_2 er 1 vil den ikke ha noen påvirkning på farten til systemet. K_2 vil oppdatere seg mellom hvert punkt når nødvendig oljestrøm for det gitte segmentet blir beregnet. Ny totaltid, rampetid og maks hastighet blir så regnet ut på nytt, vist i formel 3.53, 3.54 og 3.55.

$$T_{max_2} = \frac{T_{max_1}}{K_2} \tag{3.53}$$

$$\vec{T}_{r_3} = \begin{bmatrix} T_{r_{3_{Sving}}} \\ T_{r_{3_{Lofte}}} \\ T_{r_{3_{Folde}}} \end{bmatrix} = \begin{bmatrix} \frac{T_{r_{2_{Sving}}}}{K_2} \\ \frac{T_{r_{2_{Lofte}}}}{K_2} \\ \frac{T_{r_{2_{Folde}}}}{K_2} \end{bmatrix} \tag{3.54}$$

$$\vec{v}_{max_3} = \begin{bmatrix} v_{max_3{}_{Sving}} \\ v_{max_3{}_{Lofte}} \\ v_{max_3{}_{Folde}} \end{bmatrix} = \begin{bmatrix} \frac{\vec{\Delta s}}{T_{max_2} - T_{r_3{}_{Sving}}} \\ \frac{\vec{\Delta s}}{T_{max_2} - T_{r_3{}_{Lofte}}} \\ \frac{\vec{\Delta s}}{T_{max_2} - T_{r_3{}_{Folde}}} \end{bmatrix} \tag{3.55}$$

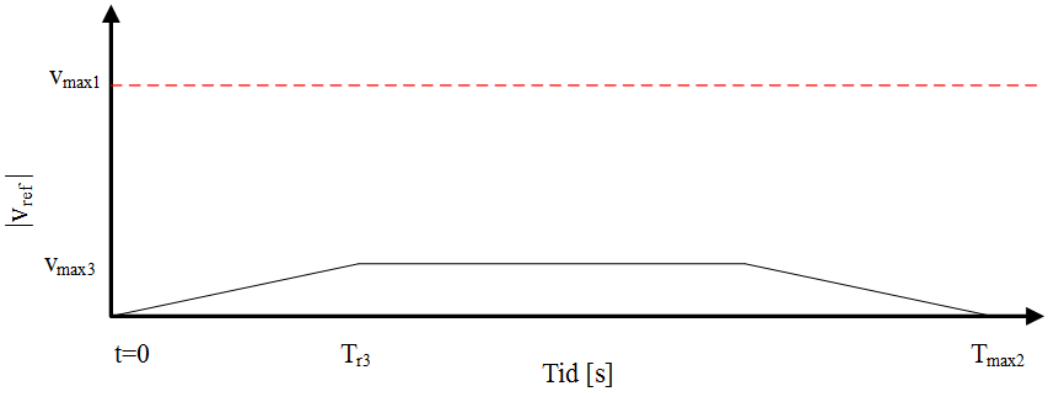
Ligningene frem til hit i dette delkapittelet beregnes kun en gang mellom hvert punkt. Den resterende koden som vil bli beskrevet under kjøres på klokkefrekvensen (10ms) til PLSen. På samme måte som som i banegenerator versjon 1 lages hastighets referansen først for å så integrere til posisjonsreferanse.

$$\vec{v}_{ref} = \begin{bmatrix} \dot{\beta}_{ref} \\ \dot{x}_{lref} \\ \dot{x}_{fref} \end{bmatrix} \tag{3.56}$$

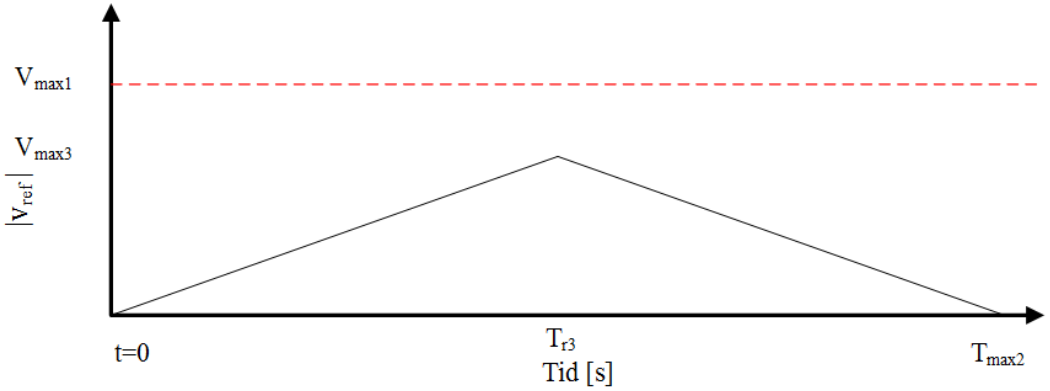
Trapes- eller trekantprofilen beregnes som vist i ligning 3.57, hvor t er den lokale tiden til banegeneratoren som økes med 10ms per syklus og resettet mellom hvert punkt.

$$\vec{v}_{ref} = \begin{cases} \frac{\vec{v}_{max_3} t}{T_{r_3}} & t < \vec{T}_{r_3} \\ \vec{v}_{max_3} & \vec{T}_{r_3} \leq t < T_{max_2} - \vec{T}_{r_3} \\ \vec{v}_{max_3} - \frac{\left(t - T_{max_2} + \vec{T}_{r_3}\right)\vec{v}_{max_3}}{\vec{T}_{r_3}} & T_{max_2} - \vec{T}_{r_3} \leq t < T_{max_2} \\ 0 & t \geq T_{max_2} \end{cases} \tag{3.57}$$

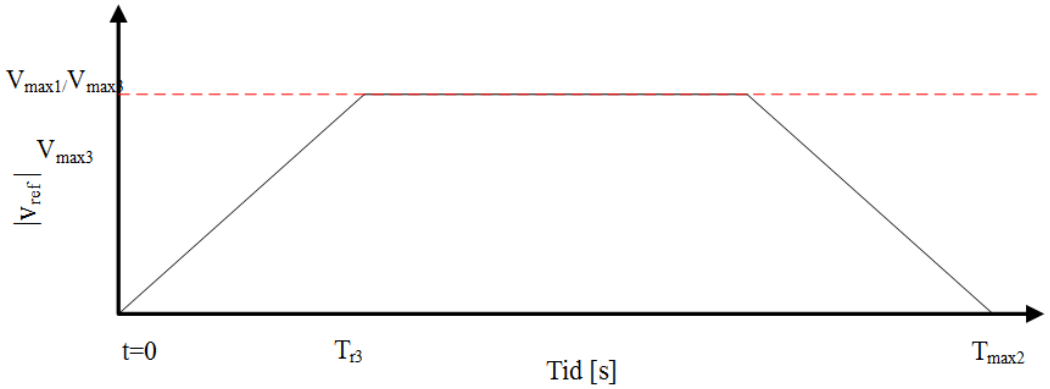
Figur 3.20 viser hvordan hastighetsreferansen kan se ut for aktuatorene under forskjellige forhold.



(a) Aktuatoren skal kjøre en kort distanse og trenger ikke kjøre på v_{max1}



(b) Aktuatoren skal en kjøre kort distanse og/eller er begrenset av a_{max}. Profilen blir en trekant



(c) Aktuatoren bruker lengst tid av alle tre og er ikke begrenset av a_{max}. Aktuatoren kjører på v_{max1}

Figur 3.20: Eksempel hastighetsreferansen til aktuatorene

Det er kun de aktuatorene som tar lengst tid, slik som beskrevet i ligning 3.47, som kan bli en trekantprofil. Som regel er det kun én aktuator som får trekantprofil, siden tidene i $\vec{T}_1$ ofte er forskjellige. Hvis den aktuatoren som bruker lengst tid ikke rekker å akselerere opp til $\vec{v}_{max1}$ blir den da en trekantprofil. De to andre aktuatorene trenger ikke å bevege seg like raskt for å rekke frem på T_{max2} og vil alltid følge en trapesform. Hastigheten integreres som vist i formel 3.58.

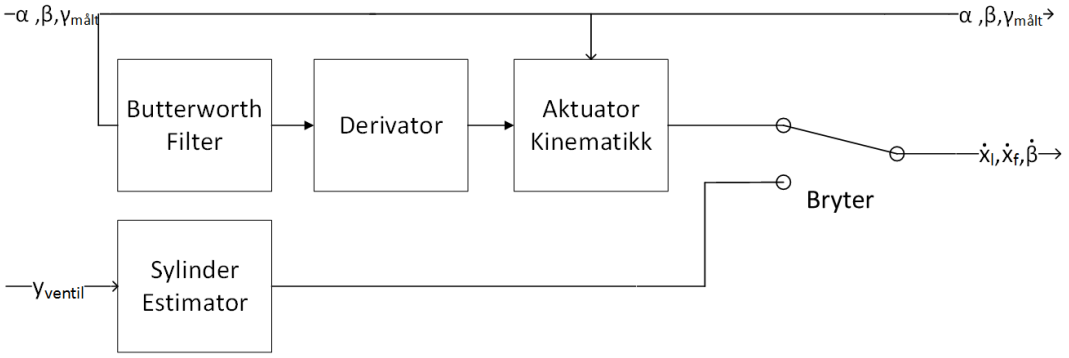
$$\vec{x}_{ref} = \begin{bmatrix} \beta_{ref} \\ x_{lref} \\ x_{fref} \end{bmatrix} = \int_0^t \vec{v}_{ref} dt \tag{3.58}$$

x_{ref} og v_{ref} brukes som referanse til kontrolleren som vist i figur 3.17.

Implikasjonene og begrensninger ved denne løsningen vil bli diskutert i diskusjonskapittelet.

3.7 Sensor og tilbakemelding

Dette delkapittelet dekker oppbyggingen av sensor- og tilbakemeldingsblokken, vist i figur 3.21.



Figur 3.21: Blokkdiagram av sensorblokken

Bryteren skiller mellom metodene som brukes for å beregne aktuatorhastigheten. NOVN hadde ønske om å finne en løsning som gjorde det mulig å bruke den deriverte av vinkelen som beskrevet i kapittel 1.2. Standard metode er å benytte ventilposisjonen som tilbakemelding.

3.7.1 Estimering av aktuatorhastighet fra ventilposisjon

Alle ventilene har trykkompensering som gjør at aktuatorhastigheten, v, er proporsjonal med ventilåpningen, y_{ventil}, som vist i ligning 3.59.

$$v = K \cdot y_{ventil} \tag{3.59}$$

Forsterkningen K er et konstant tall som er beregnet fra hydraulisk data som vist i formel 3.60 og 3.61.

$$K_{Sving} = \frac{Q_{sving}}{V_{motor} \cdot i_{gir}} \tag{3.60}$$

$$K_{Lofte/Folde} = \begin{cases} \frac{Q_{inn}}{A \cdot \phi} & \text{Sylinder inn} \\ \frac{Q_{ut}}{A} & \text{Sylinder ut} \end{cases} \tag{3.61}$$

Parameterne brukt på denne kranen er vist i tabell 3.6.

Tabell 3.6: Forsterkningsparameter fra ventilposisjon til sylinderfart

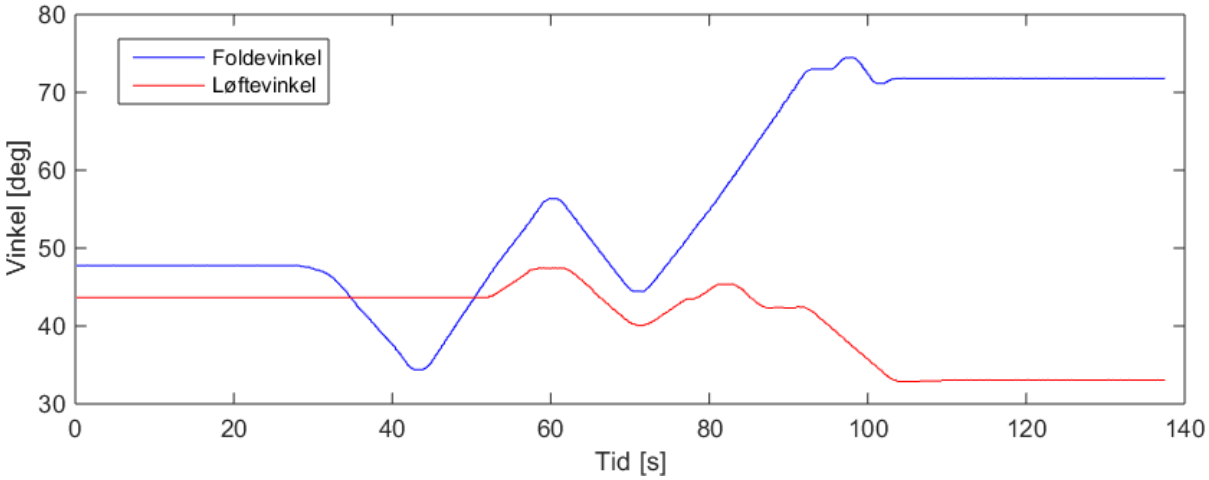
	Sylinder inn	Sylinder ut	Sving
K_{sving}			0.0643
K_{lofte}	0.0442	0.4331	
K_{folde}	0.0599	0.0643	

Selv om Q_{ut} er dobbel så stor som Q_{inn}, så er $G_{Lofte/Folde}$ nesten lik ut og inn, siden arealforholdet ϕ er cirka 0.5.

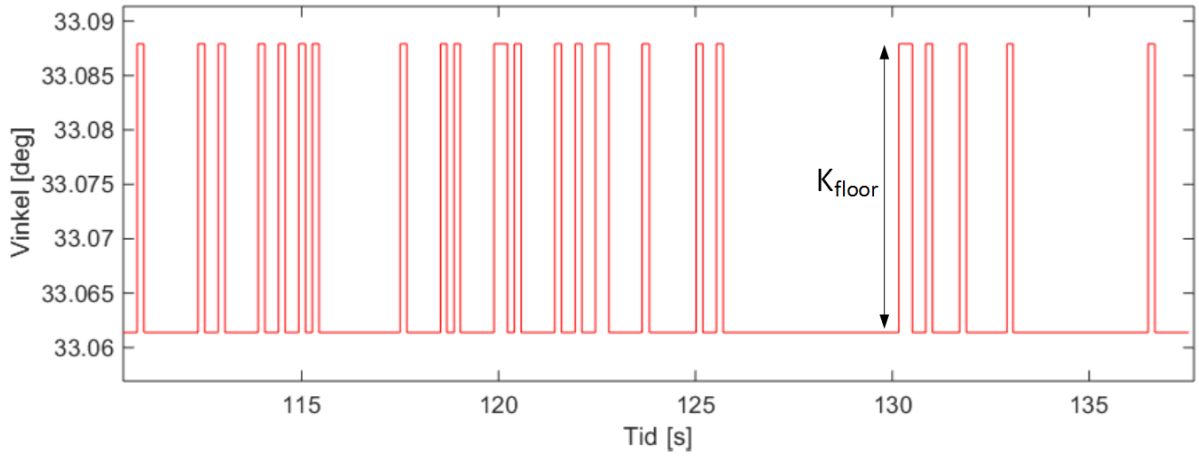
3.7.2 Numerisk derivering og filtrering

Numerisk derivasjon av vinkelmålingene er problematisk ikke bare på grunn av støy, men oppløsningen til sensoren. Sensoren kan kun detektere en endring på minimumsoppløsningen av

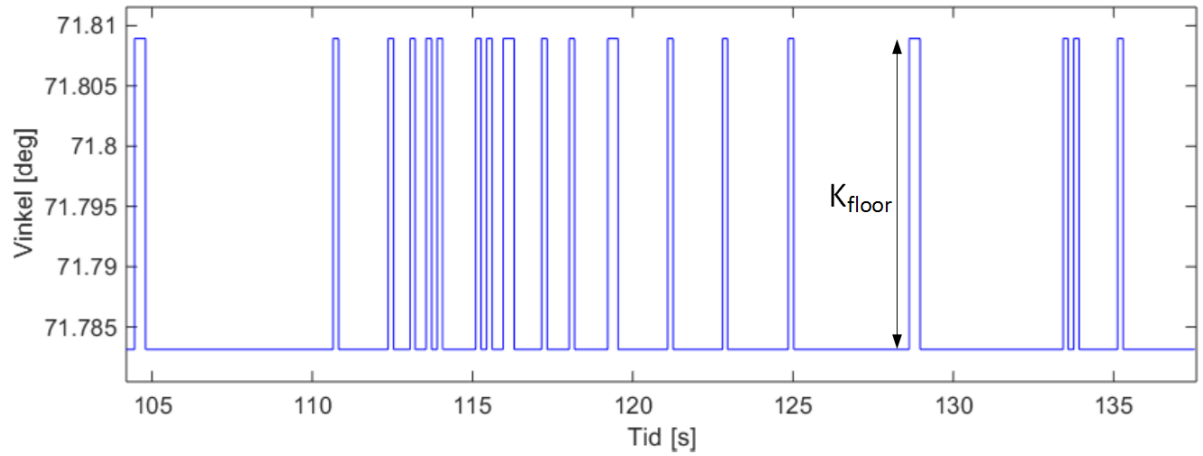
sensoren, K_{floor}. Figur 3.22 viser plot av sensormålinger, hentet fra den fysiske kranen. Ved å zoome inn på vibrasjonene i steady state så kan man se sensoroppløsningen.



(a) Sensormåling av løfte- og foldevinkel i grader



(b) Zoomet inn på steady state verdi for løftevinkel



(c) Zoomet inn på steady state verdi for foldevinkel

Figur 3.22: Rådata hentet fra sensormålinger til fysisk kran

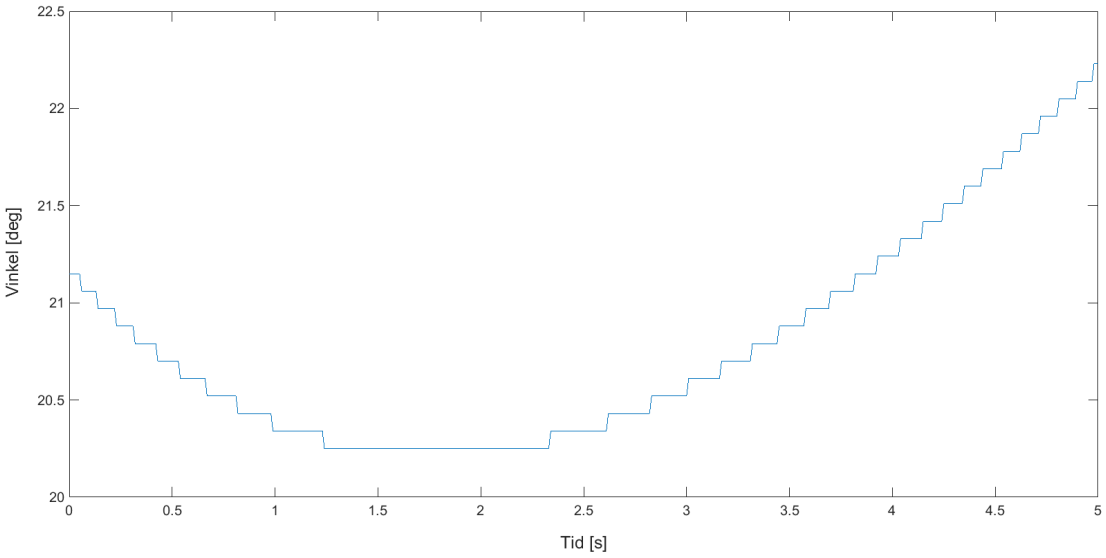
K_{floor} har enheten $\frac{grader}{steg}$. K_{floor} for sving ble gitt av NOVN, men løfte og folde har blitt estimert ut fra målinger rådata som vist i figur 3.22. Det er forskjellig oppløsning på bomvinkelsensoren og encoderen for svingvinkelen. Parameterene er vist i tabell 3.7.

Tabell 3.7: K_{floor} for alle tre sensorene

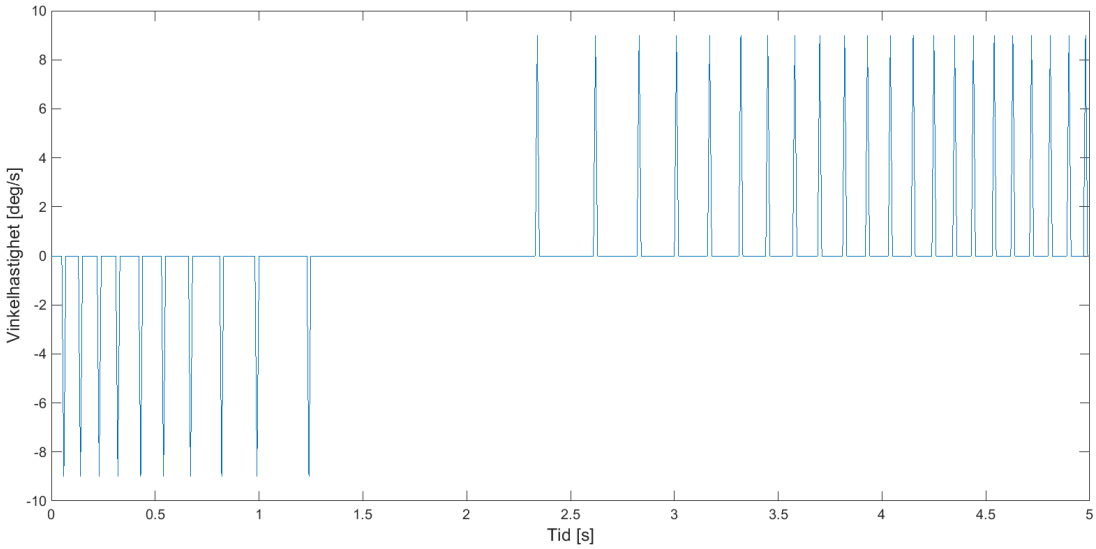
K_{floor}	[grader/steg]
Sving	0.01038
Lofte	0.02590
Folde	0.02590

Som man ser er oppløsningen på løfte og folde lik. På kranen har hovedbom og knekkbom samme fysiske sensor, som er med på å bekrefte estimeringene.

På grunn av disse brå endringene i vinkelmålingene får den deriverte vinkel skarpe topper som vist i figur 3.23.



(a) Vinkelmåling med sensoroppløsning K_{floor}



(b) Derivert vinkel uten filter

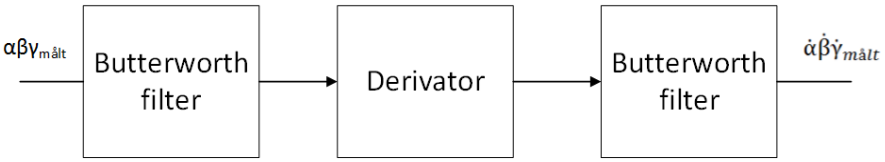
Figur 3.23: Vinkelmåling og numerisk derivert vinkel

Ved høyere oppløsning, altså lavere K_{floor}, vil toppene i den deriverte vinkel bli lavere. Siden man ikke blir helt kvitt disse toppene i den deriverte vinkel må man filtrerer. Figur 3.21 viser

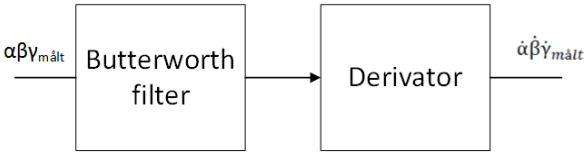
at målt vinkel deriveres og filtreres slik at den kan brukes til å regne ut sylinderhastighet. Den numeriske derivasjonen brukt i PLS koden er *backward euler*, vist i ligning 3.62

$$\dot{\alpha}_i = \frac{\alpha_i - \alpha_{i-1}}{\Delta t} \tag{3.62}$$

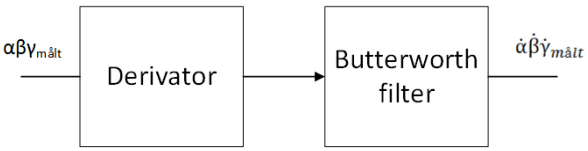
I kapittel 2.5 blir butterworth filteret introdusert. Man må finne avkutningsfrekvens ω_c og orden for å skrive transferfunksjonen til et butterworth filter. I tillegg har tre moduser for filtreringen blitt testet som vist i 3.24.



(a) Modus 1



(b) Modus 2



(c) Modus 3

Figur 3.24: Oversikt over topologier av filteret

Det ble brukt minimeringsalgoritmer for å finne avkutningsfrekvensen til butterworth filteret. Det ble satt som begrensing at filterfrekvensen skulle holde seg under Nyquistfrekvensen, $f_c < \frac{f_s}{2}$. Startverdien ble satt til 20 rad/s. Alle tre moduser ble testet på filter fra 1. til 6. orden. Avkutningsfrekvensen, orden og modus som korresponderer til lavest funksjonsverdi f_{val} ble brukt i PLS koden. Optimaliseringsalgoritmene som ble brukt er: Fmincon, Fminsearch og Patternsearch, som må kjøres 18 ganger hver, og Genetic Algorithm som kjøres 1 gang med heltallsbegrensning på modus og orden. Dette gjør at Genetic Algorithm tvinges til å finne beste modus og orden. Tabellene under viser resultatene fra optimaliseringen for knekkbommen, samme optimalisering ble kjørt for sving og løfte.

Tabell 3.8: Filteroptimalisering med Fmincon

(a) Fmincon f_{val}

	Modus		
Orden	1	2	3
1	0.0265	0.0340	0.0340
2	0.0270	0.0238	0.0238
3	0.0321	0.0248	0.0248
4	0.0378	0.0275	0.0275
5	0.1240	0.0305	0.0305
6	0.1344	0.0570	0.0570

(b) Fmincon ω

	Modus		
Orden	1	2	3
1	16.7808	6.5818	6.5816
2	22.7316	13.8035	13.8035
3	25.7029	17.7134	17.7175
4	27.0402	21.7158	21.7255
5	20.6512	22.8626	22.8628
6	25.3750	20.2642	20.2687

Fmincon viser at et 2. ordens filter med modus 2 og 3 har lavest funksjonsverdi og er den beste

løsningen.

Tabell 3.9: Filteroptimalisering med Fminsearch

(a) Fminsearch f_{val}

	Modus		
Orden	1	2	3
1	0.0265	0.0340	0.0340
2	0.0270	0.0238	0.0238
3	0.0321	0.0248	0.0248
4	0.0378	0.0275	0.0275
5	0.0455	0.0305	0.0305
6	0.0538	0.0359	0.0359

(b) Fminsearch ω

	Modus		
Orden	1	2	3
1	16.7801	6.5815	6.5815
2	22.7317	13.8029	13.8029
3	25.7014	17.7124	17.7124
4	27.0367	21.7253	21.7253
5	30.2524	22.8625	22.8625
6	35.4899	26.2153	26.2152

Fminsearch samsvarer med Fmincon og viser at modus 2/3 og et 2. ordens filter best.

Tabell 3.10: Filteroptimalisering med Patternsearch

(a) Patternsearch f_{val}

	Modus		
Orden	1	2	3
1	0.0265	0.0340	0.0340
2	0.0270	0.0238	0.0238
3	0.0321	0.0248	0.0248
4	0.0378	0.0275	0.0275
5	0.0455	0.0305	0.0305
6	0.0538	0.0359	0.0359

(b) Patternsearch ω

	Modus		
Orden	1	2	3
1	16.7801	6.5816	6.5816
2	22.7317	13.8028	13.8028
3	25.7014	17.7124	17.7124
4	27.0367	21.7253	21.7253
5	30.2525	22.8626	22.8625
6	35.4899	26.2153	26.2152

Patternsearch gir verdier som er veldig likt Fminsearch.

Tabell 3.11: Filteroptimalisering med Genetic Algorithm

f_{val}	ω	Orden	Modus
0.0238	13.8028	2	2

Med Genetic Algorithm så er modus 2 og 2. orden den beste løsningen.

Man kan også se hvordan K_{floor} påvirker resultatet, for å kunne vurdere om et 2. ordens filter med modus 2 vil være best uavhengig av K_{floor}. En test ble kjørt med Fmincon med forskjellige verdier av K_{floor} for å se om et 2. ordens filter med modus 2 alltid er beste løsning. Dette vil være interessant hvis man på andre krantyper benytter sensorer med en annen oppløsning. Ved å sammenligne minste funksjonsverdi og gjennomsnittlig snittlig funksjonsverdi med valgt løsning, kan man se hvor god løsningen er.

Tabell 3.12: 2. orden, modus 2 vs K_{floor}

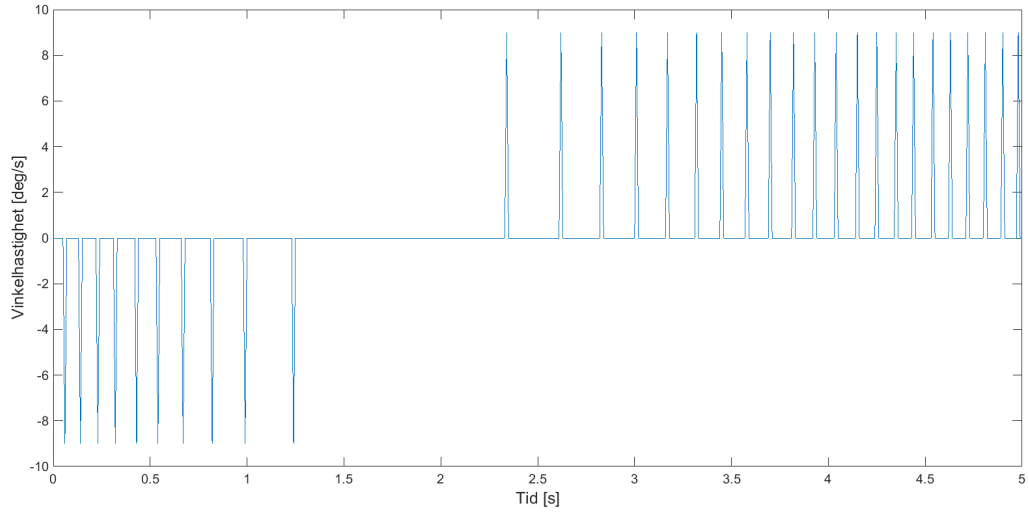
K_{floor}	0.005	0.01	0.05	0.1
min(f_{val})	0.0137	0.0167	0.0257	0.0364
mean(f_{val})	0.0336	0.0364	0.048	0.0642
2. orden, modus 2	0.0137	0.0167	0.0257	0.0364

Som tabell 3.12 viser er 2. orden med modus 2 best uavhengig av K_{floor}. De valgte avkutningsfrekvensene er vist i tabell 3.13

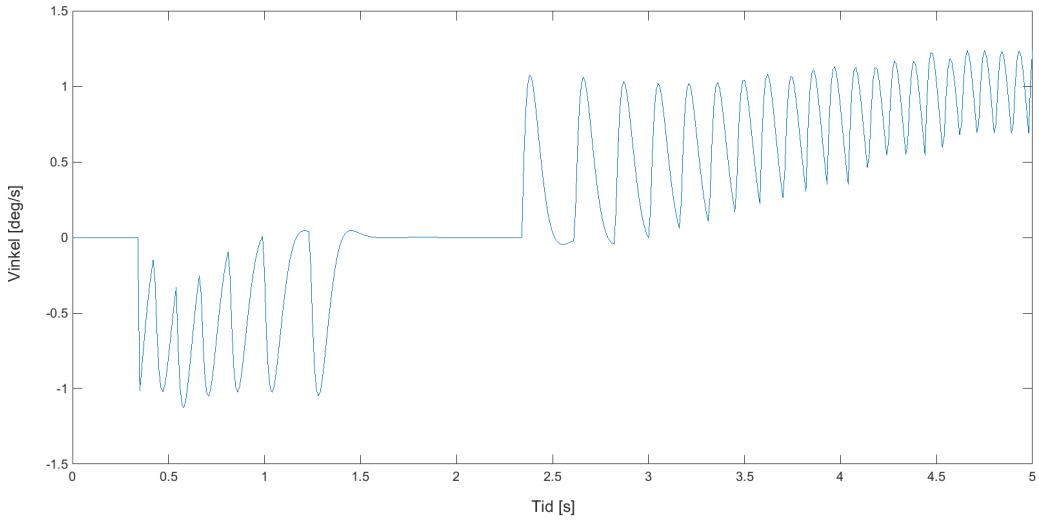
Tabell 3.13: Avkutningsfrekvens for butterworthfilter til alle tre sensorene

ω_c	[rad/s]
Sving	36.26
Lofte	26.14
Folde	13.8

Figur 3.25 viser hvordan filtreringen av vinkelen før derivering reduserer toppene som oppstår ellers.



(a) Derivert løftevinkel uten filter



(b) Derivert løftevinkel med filtrering

Figur 3.25: Vinkelhastighet med og uten filtrering

Figuren viser at toppene til den deriverte vinkelen har blitt redusert betraktelig. Nå som vinkelhastigheten har blitt estimert, kan den sendes videre til regneblokken for aktuatorkinematikk.

3.8 Programvareimplementering

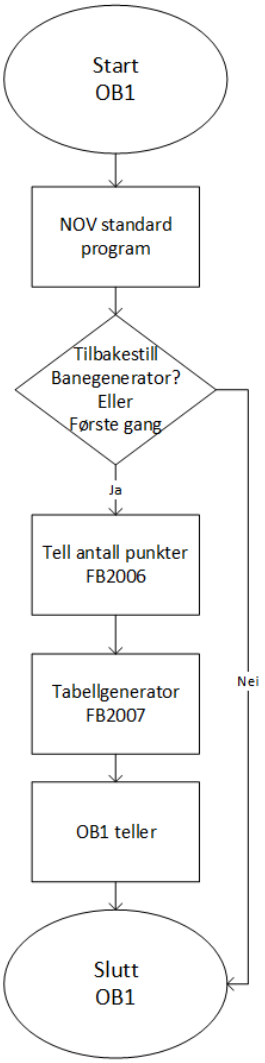
PLSen som blir brukt i dette prosjektet er en Siemens CPU315-2 PN/DP. Den er gitt av oppdragsgiver NOVN og er tilsvarende det utstyret som brukes i produksjon i dag. PLSen har en Ethernet kontakt som gjør det enkelt å koble den til et TCP/IP nettverk. All PLS koden av FB, FC og globale DB har blitt skrevet i Simatic Step7 SCL. Videre har disse blokkene blitt organisert i OB blokker i FBD. Full programkode for alle blokker skrevet i dette prosjektet ligger i vedlegg C.

Nummerering av programblokkene relatert til dette prosjektet er som følger.

Tabell 3.14: Nummerering av blokker i PLS

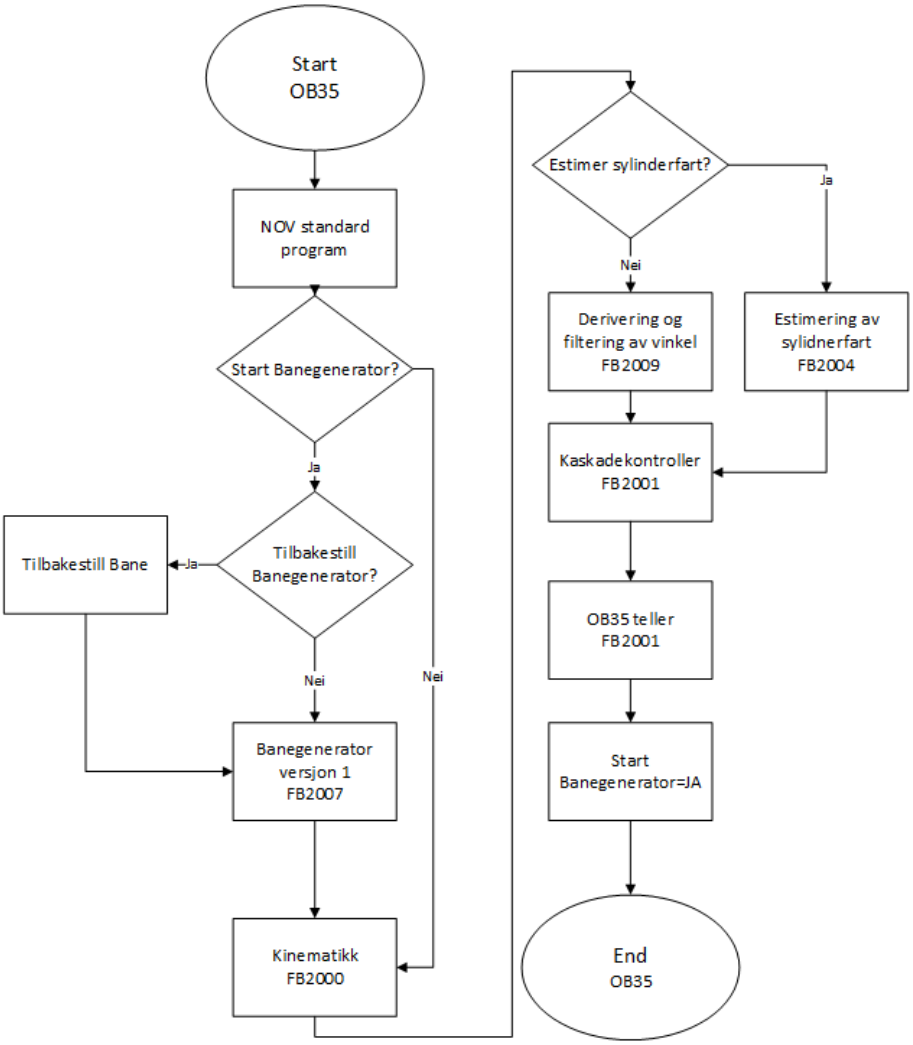
Blokker	Symbolsk navn
Function Block	FB 2000, FB 2001..
Instance Data Block	DB 2000, DB 2001..
Function Call	FC 3000, FC 3001..
Global Data Block	DB 4000, DB 4001..

Strukturen i PLS programmet er forskjellig fra banegenerator versjon 1 og 2. Figur 3.26 og 3.27 viser hvordan OB1 og OB35 er satt opp. OB35 har en syklustid på 10 ms. OB1 kjører så raskt som mulig, og syklustiden varierer ut i fra hvor stort program man har på PLSen. Banegenerator versjon 2 bruker ikke blokker i OB1 utenom det som allerede ligger inne som standard NOVN programkode.



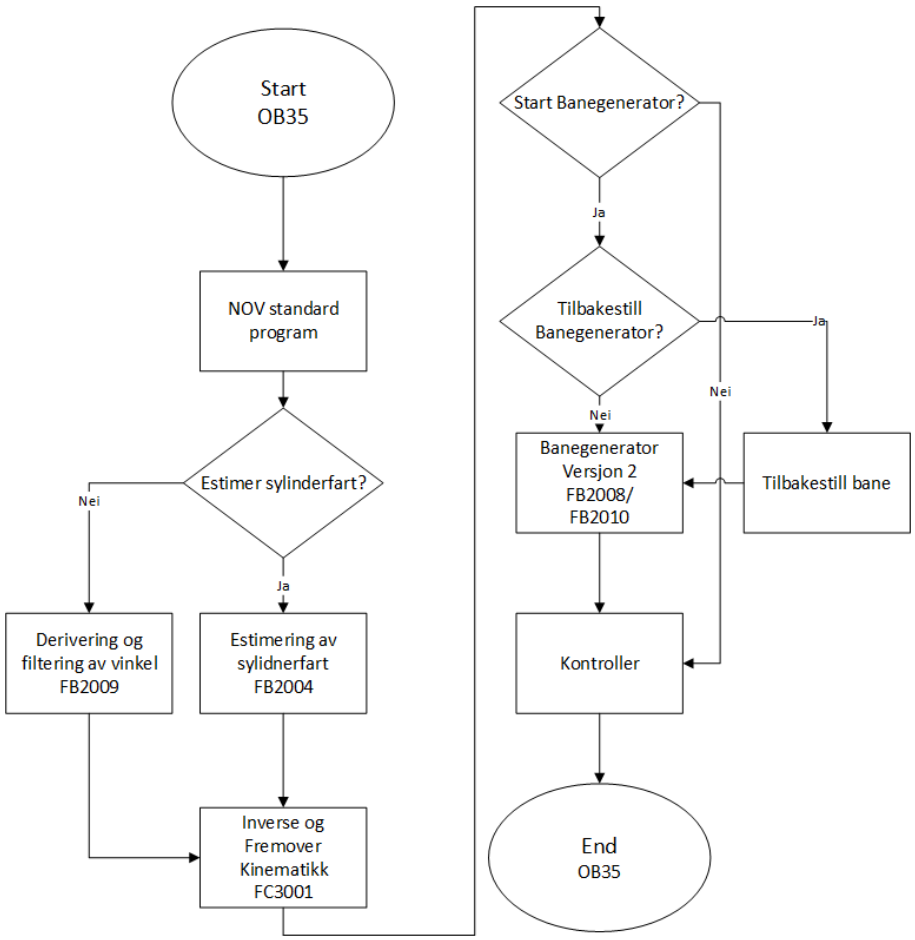
Figur 3.26: Blokkdiagram av OB1 ved bruk av banegenerator versjon 1

Grunnen til at tabellgeneratoren er satt i OB1 er fordi den krever omfattende databehandling. FB2006 teller hvor mange punkter kranen skal gå innom. FB2007 lager en stor tabell med alle verdiene beskrevet av ligningene 3.25, 3.26, 3.27 og 3.28. Hvis disse funksjonsblokkene settes i OB35 ville ikke PLSen rukket å bli ferdig på 10ms.



Figur 3.27: Blokkdiagram av OB35 ved bruk av banegenerator versjon 1

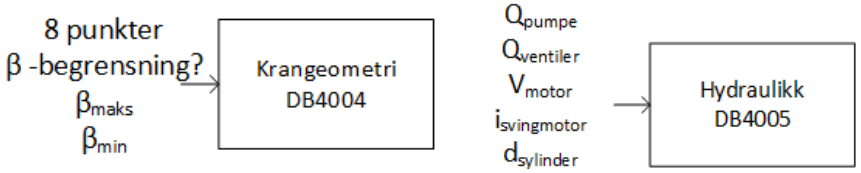
OB35 må kjøre en gang for å regne ut faktisk *xyz* verdi før banegeneratoren kan startes, dette gir også nok tid til FB2006 og FB2007 i OB1 å bli ferdig med å generere tabellene. Ved bruk av banegenerator versjon 2 har PLS koden en litt annerledes struktur vist i figur 3.28.



Figur 3.28: Blokkdiagram av OB35 ved bruk av banegenerator versjon 2

Banegenerator versjon 2 bruker ikke en omfattende tabell med posisjonsvektorer, enhetsvektorer, lengder mellom punkt og tid mellom punkt slik som versjon 1. Dette gjør at alt kan settes i OB35 og kjøre på 10ms syklustid.

Banestyringen skal virke på rørhåndteringskraner av alle størrelser og det er derfor laget to globale datablokker DB4004 og DB4005 hvor man legger inn spesifikasjoner til den gitte kranen. Figur 3.29 viser hvilke parametere som må defineres i DB4004 og DB4005. I DB4004 fyller man inn koordinatene til de åtte punktene vist i figur 2.9 og informasjon om kranens svingvinkel begrensninger. I DB4005 fylles informasjon om det hydrauliske systemet inn.



(a) Krangeometri - DB4004

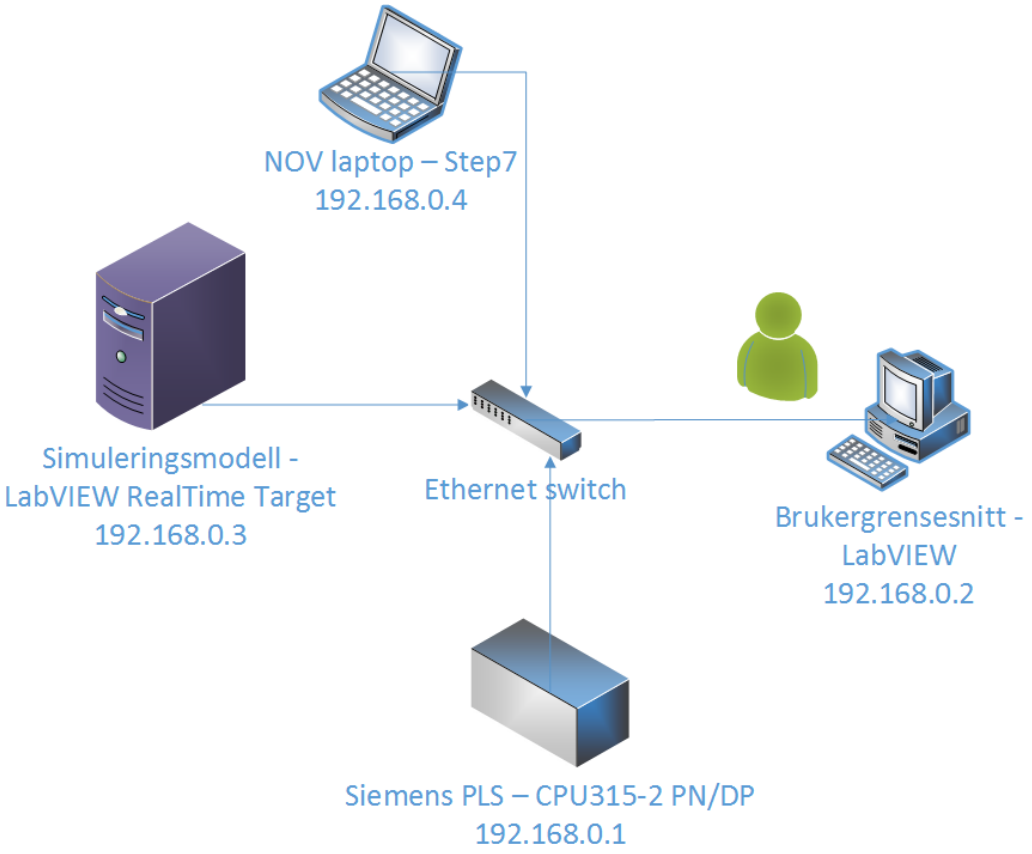
(b) Hydrauliske spesifikasjoner - DB4005

Figur 3.29: Globale datablokker for kranspesifikke parametere

I denne oppgaven ble disse verdiene satt direkte i datablokken, men ved videre utvikling av system arkitekturen vil disse verdiene settes på et høyere nivå fra for eksempel en HMI.

3.9 HiL-simulering og testing

Mesteparten av testingen har blitt gjort ved bruk av en HiL-simulering. Figur 3.30 viser hvordan systemet er satt opp.

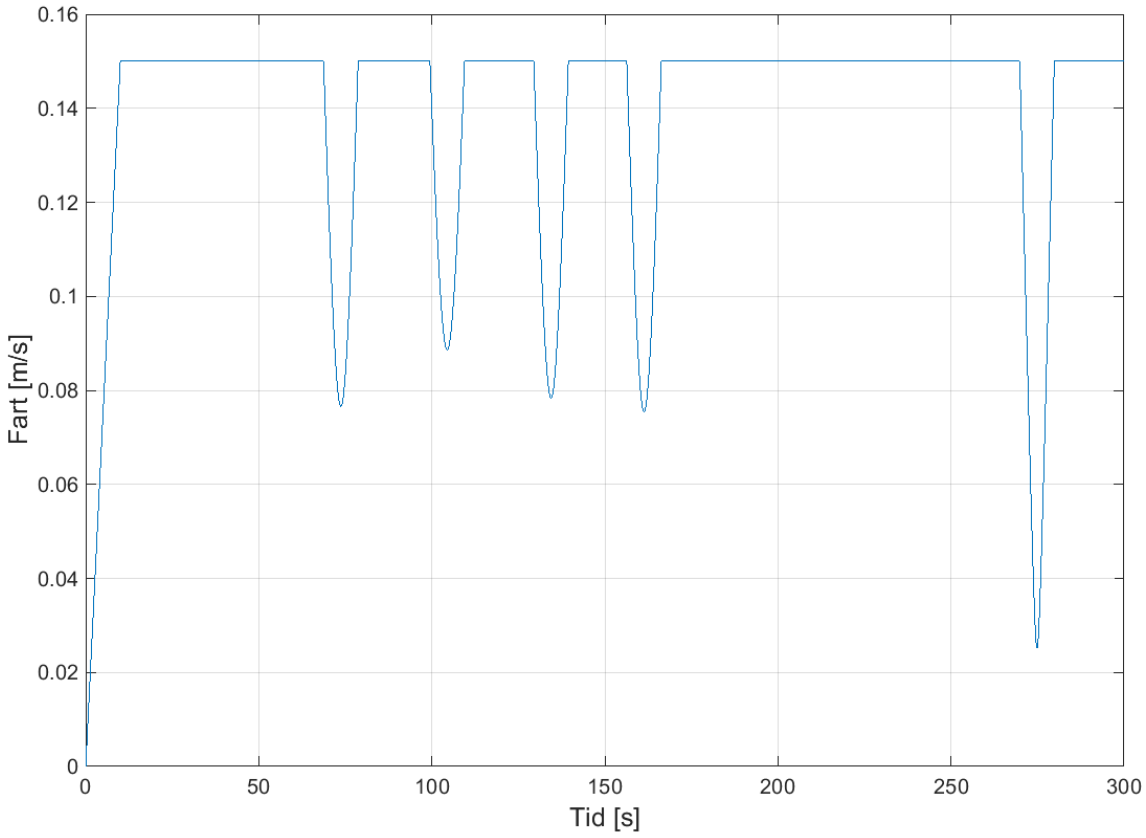


Figur 3.30: HiL-simulering blokkdiagram

Komponentene er koblet sammen i et lukket TCP/IP nettverk. NOVN laptop brukes til å laste opp PLS programmet og logge data fra PLSen. Datamaskinen med brukergrensesnittet styrer kommunikasjonen mellom PLS og simuleringsmodellen. Siemens PLSen av type CPU315-2 PN/DP og inneholder kontroller, baneplanlegger, aktuatorhastighetsestimering og kinematikk. Simuleringsmodellen er en dedikert datamaskin som kjører matematisk modell av kranen.

3.9.1 HiL-testing av banegenerator versjon 1

Det ble gjort begrenset testing av Banegenerator versjon 1, da den ble testet i masteroppgaven fra 2013 [1]. Arbeidet i dette prosjektet har fokusert mest på banegenerator versjon 2, og videre arbeid på versjon 1 ble skrinlagt. Det ble allikevel gjort litt testing på selve funksjonaliteten av versjon 1, for å avdekke eventuelle svakheter. En av svakhetene til banegenerator versjon 1 kommer fra avrundingen som skjer mellom to punkter. Når referansen avrundes så synker den tangetielle hastigheten. Dette er vist i figur 3.31.

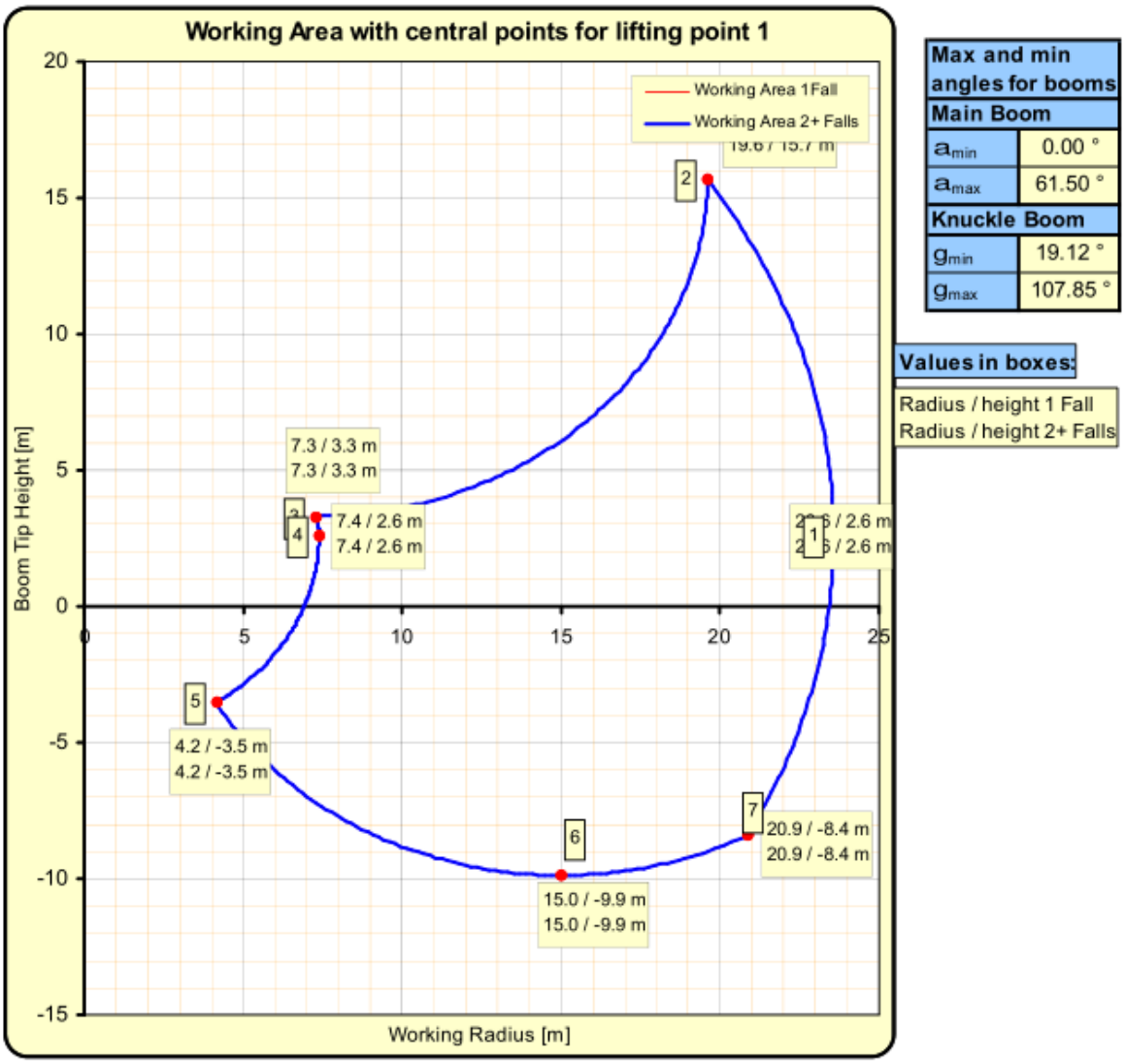


Figur 3.31: Krantuppens tangentielle hastighet, versjon 1

Disse hastighetsreduksjonene kommer når retningen endres. Dette tar ikke banegenerator versjon 1 hensyn til når tiden mellom hvert punkt blir regnet ut. Dette gjør at integreringen fra hastighetsreferanse til posisjonsreferanse blir unøyaktig. Resultatet av dette er at man aldri kommer frem til det punktet man skal til, og dette avviket blir større og større for hvert punkt. I tillegg til problemene med stort avvik laget versjon 1 baner som er umulig for kranen å nå. Dette var spesielt fremtredende når man skulle rotere kranen rundt z-aksen. Referansen gitt direkte mellom to punkter i $\mathbb{R}^3$, mens kranen stanget mot enden av sitt arbeidsområde. Spesielt problemet med arbeidsområdet gav ideen som ble til versjon 2.

3.9.2 HiL-testing av banegenerator versjon 2

For å teste ytelsen til systemet med banegenerator versjon 2 ble kranen kjørt til tilfeldige punkter innenfor arbeidsområdet over en lang tidsperiode. Å definere arbeidsområdet i aktuatorrommet er trivielt da man kun behøver å holde seg innenfor maksimum og minimum sylinderlengde. Begrensning på sving vinkel tar ligning 3.36 seg av. Figur 3.32 viser kranens arbeidsområde hentet fra dokumentasjon [19].



Central points on the working area curve

		1 Fall				2+ Falls			
		WR [mm]	ZL [mm]	α [°]	γ [°]	WR [mm]	ZL [mm]	α [°]	γ [°]
Point 1	Max WR	23572	2570	27.8	107.8	23572	2570	27.8	107.8
Point 2	Max height	19601	15664	61.5	107.8	19601	15664	61.5	107.8
Point 3	Min 1, a = Max	7343	3309	61.5	19.1	7343	3309	61.5	19.1
Point 4	Min at boom pin	7380	2570	55.8	19.1	7380	2570	55.8	19.1
Point 5	Min 2, a = Min	4153	-3530	0.0	19.1	4153	-3530	0.0	19.1
Point 6	Min height	15001	-9874	0.0	79.8	15000	-9874	0.0	79.8
Point 7	Max at boom horizontal	20860	-8408	0.0	107.8	20859	-8408	0.0	107.8

Figur 3.32: Kranens arbeidsareal

Figur 3.32 er $x'z'$ plot av arbeidsrommet til kranen, men kranen kan også rotere rundt z-aksen. Dersom arbeidsområdet skulle blitt definert i xyz ville dette blitt en komplisert funksjon av de åtte punktene, vist i figur 2.9. Ved å kjøre en tilfeldig bane utsettes kranen for de fleste situasjoner den kan havne i under drift, samtidig som man dekker hele arbeidsrommet. Derfor ble programkoden modifisert for å generere tilfeldige koordinater i aktuatorrommet. Det ble brukt en *Linear congruential generator*, den lager et pseudotilfeldig tall basert på forrige verdi som vist i ligning 3.63.

$$X_{n+1} = (aX_n + c) \, mod \, m \tag{3.63}$$

hvor;

X_{n+1}	=	Ny pseudotilfeldig verdi
X_n	=	Forrige pseudotilfeldig verdi
a	=	Skaleringsparameter
c	=	Skaleringsparameter
m	=	Modulus, $X_{max} - X_{min}$
X_0	=	Initialverdi, satt til 0

X_n er en vektor med tre koordinater i aktuatorrommet som brukes istedenfor å hente verdier fra koordinatene i baneplanleggeren, vist i figur 3.17.

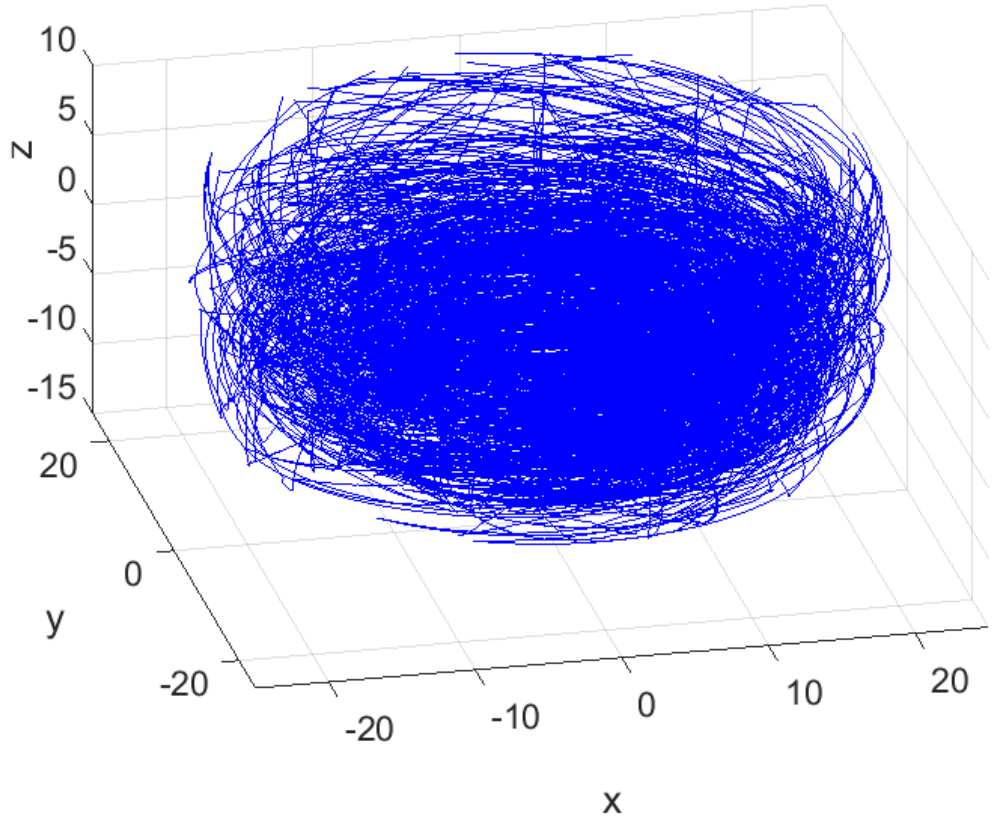
$$\vec{X}_n = \begin{bmatrix} \beta_{ref} \\ x_{lref} \\ x_{fref} \end{bmatrix} \tag{3.64}$$

En ny verdi for X_{n+1} beregnes en gang mellom hver punkt, det vil si at når referansen kommer til settpunkt, så finner den et nytt punkt og beveger seg mot det. På denne måten får man en stor mengde data som tester kranen i alle mulige posisjoner gitt nok tid.

Tabell 3.15: Parametere for pseudotilfeldig generator

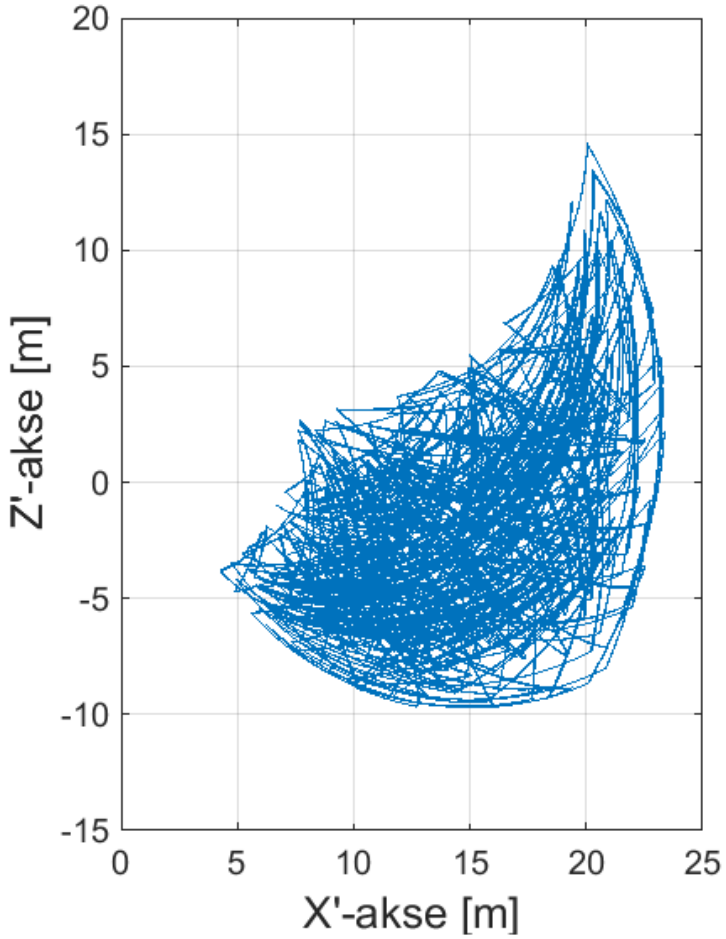
Parametere	Verdi	Enhet
m_{sving}	12.56	[rad]
m_{lofte}	2.3	[m]
m_{folde}	2.8	[m]
a_{sving}	10	
a_{lofte}	3	
a_{folde}	3	
c_{sving}	100	
c_{lofte}	1	
c_{folde}	1	

m-parameteren er avhengig av spesifikasjonene på kranen, mens a og c er satt eksperimentelt. Kranen ble kjørt i HiL i cirka 20 timer der den fulgte en pseudotilfeldig bane. Figur 3.33 viser banen kranen har kjørt i xyz i løpet av disse 20 timene.



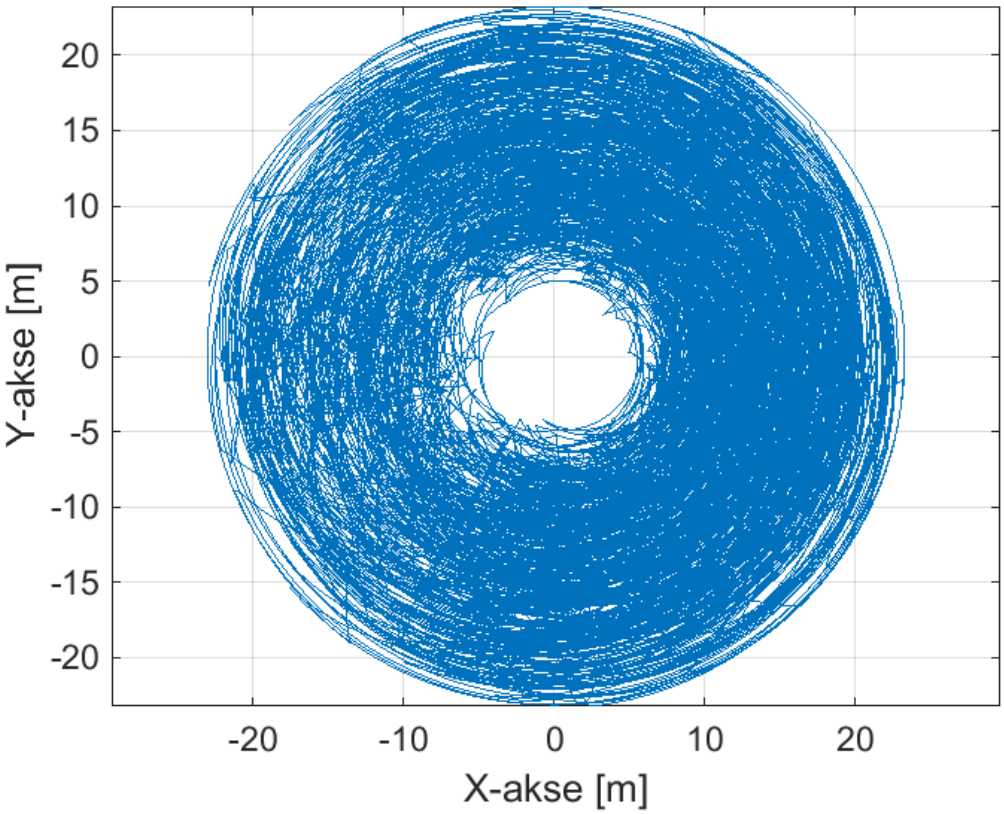
Figur 3.33: Tilfeldig bane kjørt i cirka 20 timer

Det er ikke lett å tyde denne figuren, men ved å vise resultatene i kranens koordinatsystem $x'z'$-planet blir det lettere å tolke dataen. Dette vises i figur 3.34



Figur 3.34: $x'z'$ plot

Figur 3.34 viser kranen har holdt seg innenfor arbeidsområdet vist i figur 3.32 og har dekket store deler av det. Figur 3.35 viser plot av banen sett ovenfra.



Figur 3.35: *xy* plot av hele banen

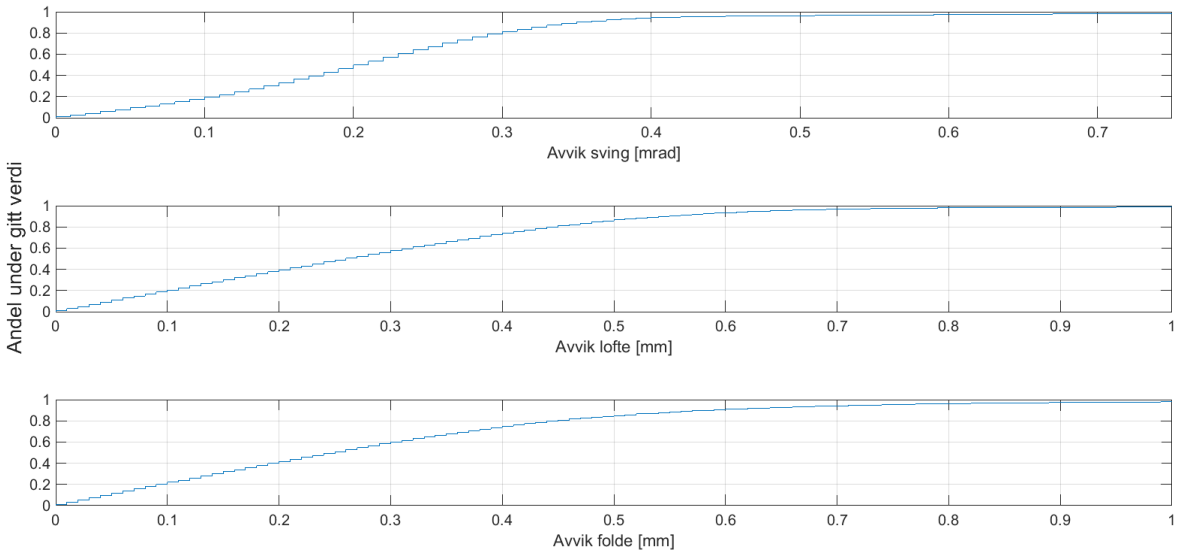
Som figuren viser har kranen dekket et område på 360°.

Ytelsen til systemet blir målt ved å se på hvor stort avvik aktatoren har fra referansen over hele tidsperioden. Som forklart i delkapittel 2.4 ble flere metoder brukt til å innregulere systemet. Parameterne fra alle metodene ble testet over en kortere periode der de kjører samme pseudo-tilfeldig bane.

Tabell 3.16: Alle metoder

Metode	Modus	Avvik	sving [mrad]	lofte [mm]	folde [mm]
Fmincon	A	RMS	0.3071	0.3484	0.4076
		Maks	2.0700	3.9100	2.4700
	B	RMS	0.3415	0.7067	0.3215
		Maks	2.4100	3.7900	2.0000
Fminsearch		RMS	0.2933	1.7152	0.5462
		Maks	2.2800	33.7100	5.3800
Genetic Algorithm	A	RMS	0.2849	0.4221	0.4435
		Maks	1.9600	5.0400	3.4400
	B	RMS	0.2822	1.9598	1.9930
		Maks	1.9700	5.9900	6.7200
Pattern Search	A	RMS	0.2764	0.6763	0.5017
		Maks	2.0600	13.7400	3.5100
	B	RMS	0.2892	0.3737	0.3458
		Maks	1.9200	3.8300	2.9600
PID tuner		RMS	0.6494	0.6395	0.6709
		Maks	3.1300	7.3100	5.1500
Semimanuell		RMS	5.6880	1.7693	0.8633
		Maks	35.3600	13.8500	6.4600

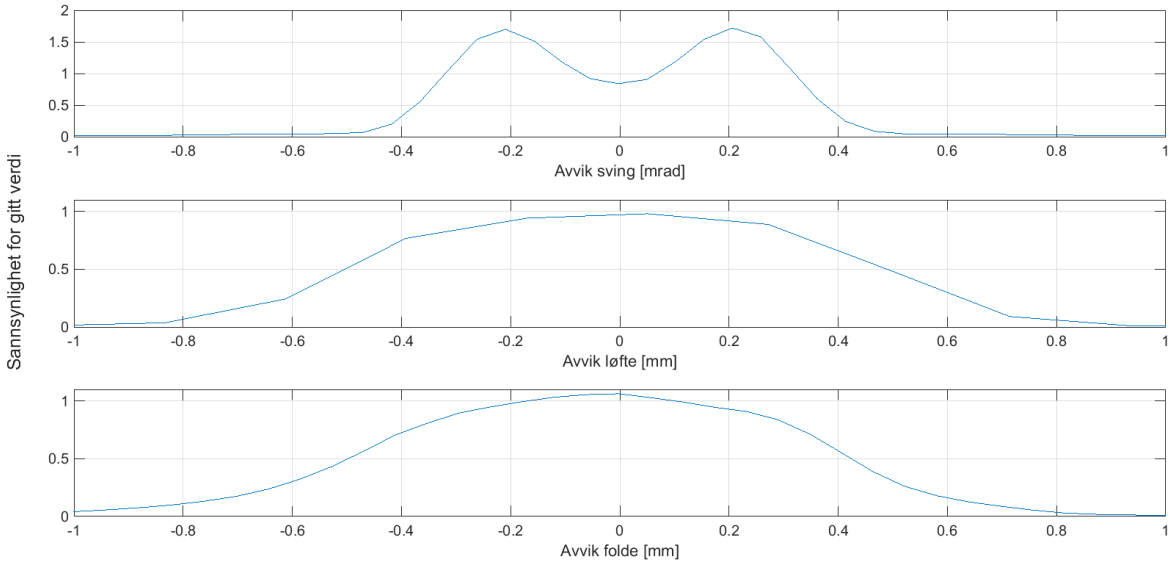
Beste løsning velges ved å ta løsningen med lavest sum av alle tre RMS verdier. Patternserach modus B gir marginalt bedre løsning enn Fmincon modus A og velges derfor som beste løsning. Patternsearch løsningen blir igjen testet over en lengre periode for å få en stor mengde data. Figur 3.36 er den kumulative sannsynlighetsfordelingen til avviket for denne løsningen. Den viser hvor stor andel av målingene som er under en gitt verdi. Figuren er generert i MATLAB ved hjelp av *cdfplot* funksjonen.



Figur 3.36: Kumulativ fordeling av avviket til krantupp fra referansen

Den deriverte av den kumulative sannsynlighetsfordelingen er sannsynlighetstettheten. Figur

3.37 viser hvor stor sannsynlighet det er for en gitt verdi oppstår. Figuren er generert i MATLAB ved hjelp av *ksdensity* funksjonen.



Figur 3.37: Sannsynlighetstetthet av avviket til krantupp fra referansen

All testing har blitt utført ved hjelp av estimering av sylinderfart som beskrevet i kapittel 3.7.1. Derivering og filtrering av vinkel fra kapittel 3.7.2 var ikke hovedfokus i denne oppgaven, men det har også blitt gjort og testet. Dette betyr at bryteren på figur 3.21 settes i *opp* posisjon. Banen var den samme som da alle kontrollparameterne ble testet og patternsearch modus B parametere ble brukt.

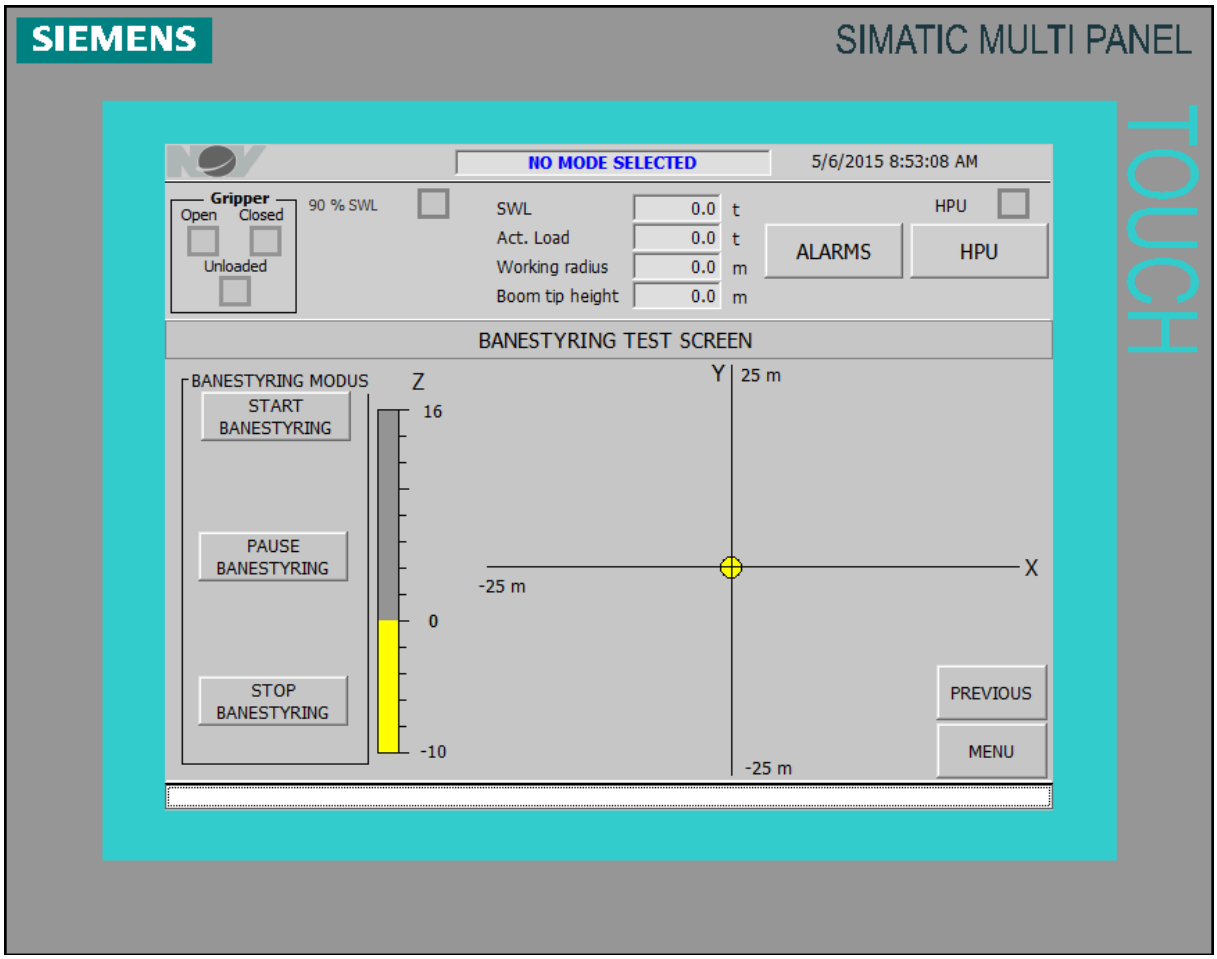
Tabell 3.17: Avvik ved bruk av numerisk derivering av vinkel og beste kontrollparametere

Metode	Avvik	sving [mrad]	lofte [mm]	folde [mm]
Numerisk derivasjon og beste løsning	RMS	8.3140	7.3736	6.1996
	Maks	24.4400	16.7300	15.0800

Avviket ble en del større enn ved brukt av sylinderfartestimering.

3.9.3 In-House-testing av banegenerator versjon 2

Det ble utført en In-House-test på NOVN sine kontorer. Der ble all nyutviklet kode lagt inn i prosjekt G4554 og testet. Testoppsettet bestod av en liten simulator som simulerer innganger og utganger, samt kranbevegelsene. Modellen som ligger til grunn er en veldig forenklet modell, så det er kun programfunksjonaliteten som testes. Alle innganger og utganger ble koblet til banestyring systemarkitekturen som vist i figur 3.3, og erstatter tilkoblingene fra HiL-simuleringen. Systemet ble testet for diverse programfeil, og for å se hvordan banestyring fungerte parallelt med resten av programmet som ligger inne som standard. Det ble laget en egen skjerm i WinCC som brukes til å operere banestyringen under drift, vist i figur 3.38.



Figur 3.38: HMI til banestyringen

Denne skjermen inneholder tre knapper for start, stop og pause. Det ble også utviklet en logikk for å koble disse tre knappene til kontrollsystemet. I tillegg vises et xy-plot over krantuppens posisjon, samt en graf for krantuppens høyde. Testingen viste at banestyringen fungere etter det hadde blitt implementert i riktig prosjekt, og var klar til å testes på fysisk kran.

3.9.4 Fysisk-testing av banegenerator versjon 2

Fysisk test på rørhåndteringskranen ble utført 7. og 8. mai på prosjektnummer G4554 i Søgne. Kranen var plassert på et eget testfundament. Et bilde av kranen vises i figur 3.39.

Figur 3.39: Bilde av fysisk kran som ble testet

Denne kranen hadde ikke gjennomgått den interne testen som gjøres før levering, som gjorde arbeidet vanskelig. Den interne testen vil blant annet innebære innstilling av ventilene og dødbåndskompensering. Sensoren som måler ventilposisjonen var også ødelagt, og hadde ikke blitt byttet ut. Et DMA-kort benyttes for å styre strømmen til spolen inni ventilen, og bruker ventilposisjon som tilbakemelding. Fraværet av denne tilbakemeldingen gjorde at ventilene hadde veldig stort dødbånd og mye hysterese. Disse ulinearitetene gjør det hydrauliske systemet tregere og mer ustabilt. Som figur 3.21 viser er det mulighet for å derivere bomvinkelmålingene for å skape en tilbakemelding til den indre hastighetssløyfen på kaskadekontrolleren. Dette gjorde at testen kunne utføres, dog med redusert ytelse på kranen.

Først ble det utført en test med de kontrollparameterne som ble funnet med optimaliseringsalgoritmene. En enkel bane ble laget mellom to punkter for å teste disse parameterne. Det viste seg at disse parameterne var alt for aggressive, og kranen ristet veldig mye under kjøringen. Deretter ble parameterne fra MATLAB PID Tuner testet. Disse er ikke like aggresive, men kranen var fortsatt veldig ustabil når banen ble kjørt. Dette vil diskuteres senere i kapittel 4.4. Det

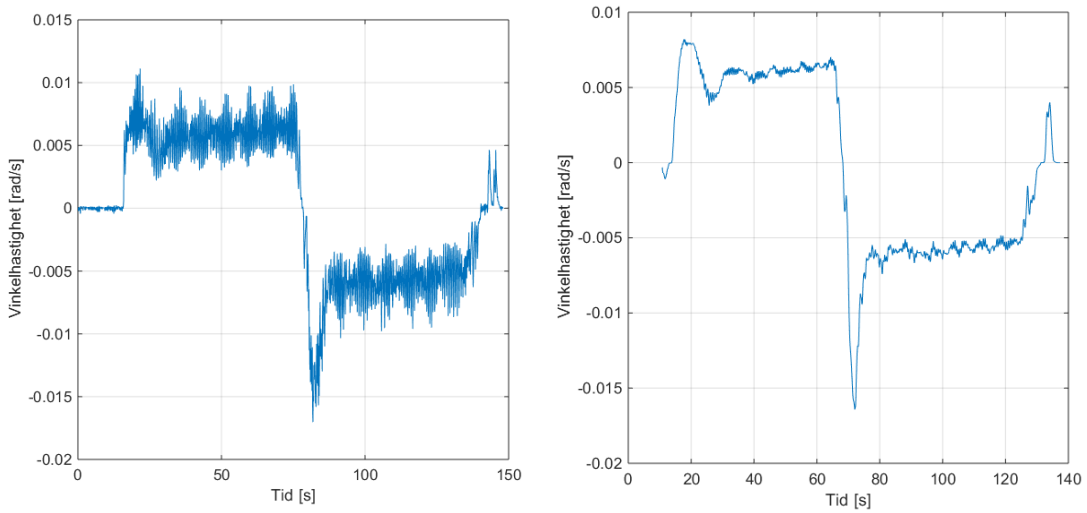
ble derfor brukt litt tid på å innregulere kontrolleren manuelt, for å få akseptabel respons på systemet. Etter noen forsøk ble parameterne satt, som vist i tabell 3.18. Parameterne er like for alle tre aktuatorene.

Tabell 3.18: Kontrollparametere brukt på fysisk test

K_{pytre}	K_{pindre}	K_{iindre}
0.4	1	10

Disse kontrollparameterne gjorde at systemet ble stabilt, og det ble ikke brukt mer tid på innregulering. Flere tester ble kjørt for å verifisere at systemet oppførte seg stabilt, blant annet ved å benytte den pseudo-tilfeldige koordinatgeneratoren.

Siden derivering av vinklene ble brukt til å lage tilbakemelding på indre sløyfe, ble effekten av filteret på systemet også undersøkt. Det ble utført to tester, en test hvor filterne hadde den avkutningsfrekvensen som har blitt beregnet med optimaliseringsalgoritmene, og en test hvor avkutningsfrekvensen ble satt til en tidel av dette. Dette ble gjort for å se hvorvidt det er best med et glatt signal eller et hurtig signal. En lav avkutningsfrekvens vil glatte ut signalet, men vil også gi tidsforsinkelser. Filteret som brukes i nåværende versjon av aksestyringen har lav avkutningsfrekvens. Derfor er dette av stor interesse for å kunne forbedre aksestyringen. Banen som ble kjørt inneholdt to punkter i xz-planet. Figur 3.40 viser den deriverte vinkelhastigheten $\dot{\alpha}$ ved forskjellig avkutningsfrekvens på filteret.

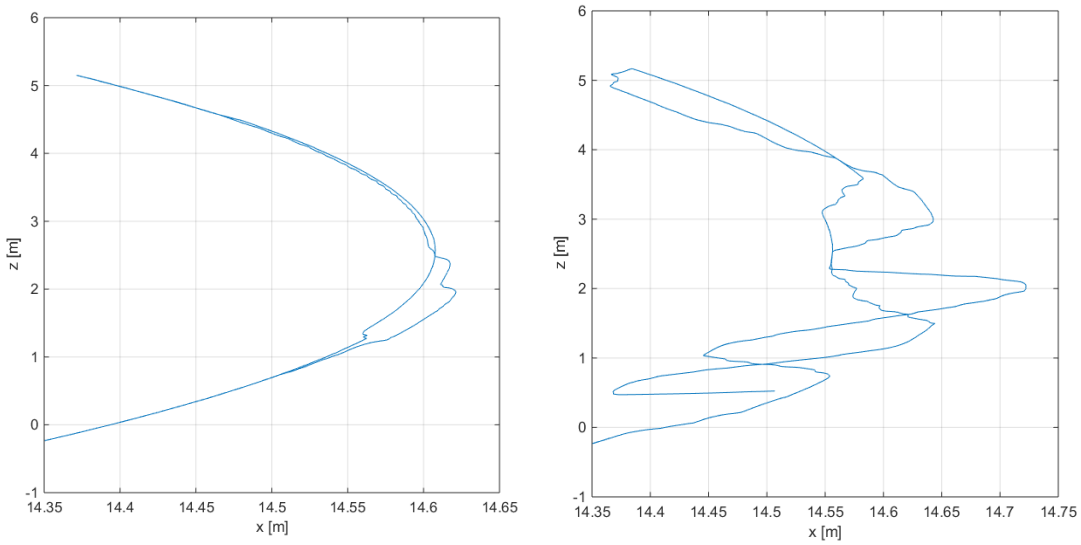


(a) Høy avkutningsfrekvens

(b) Lav avkutningsfrekvens

Figur 3.40: Vinkelhastigheten $\dot{\alpha}$ ved forskjellig avkutningsfrekvens

Resultatene fra disse to testene viste at det var best med høy avkutningsfrekvens. Dette kommer godt til syne når man plotter posisjonen til krantuppen i xz-planet, vist i figur 3.41.

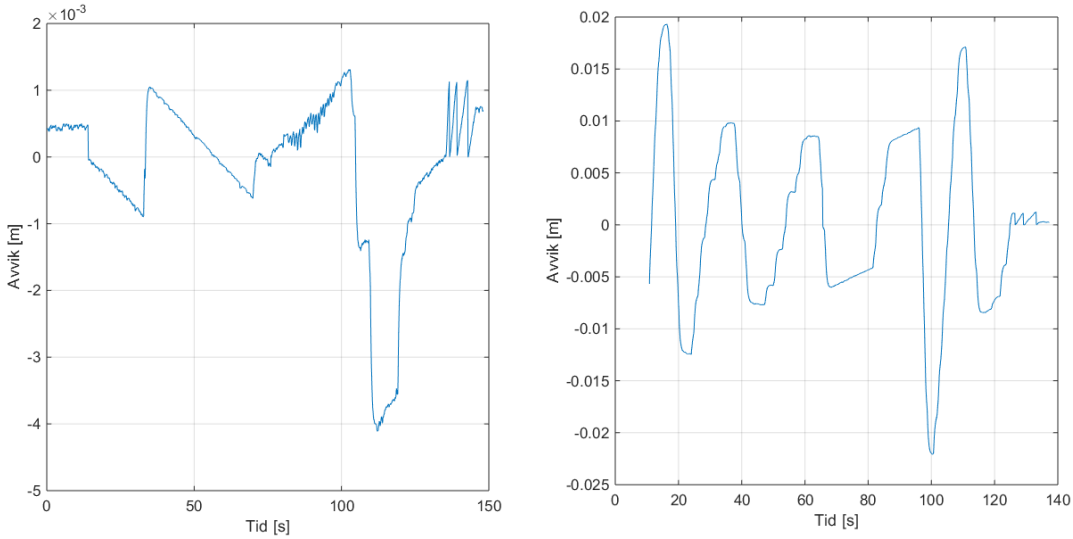


(a) Høy avkutningsfrekvens

(b) Lav avkutningsfrekvens

Figur 3.41: XZ-plot ved forskjellig avkutningsfrekvens

Det viste seg at foldesylinderen var spesielt avhengig av hurtig tilbakemelding på indre sløyfe. Tidsforsinkelsen fra filteret gjorde aktuatorbevegelsen ustabil. Vi kan sammenligne avviket i de to testene, vist i figur 3.42.

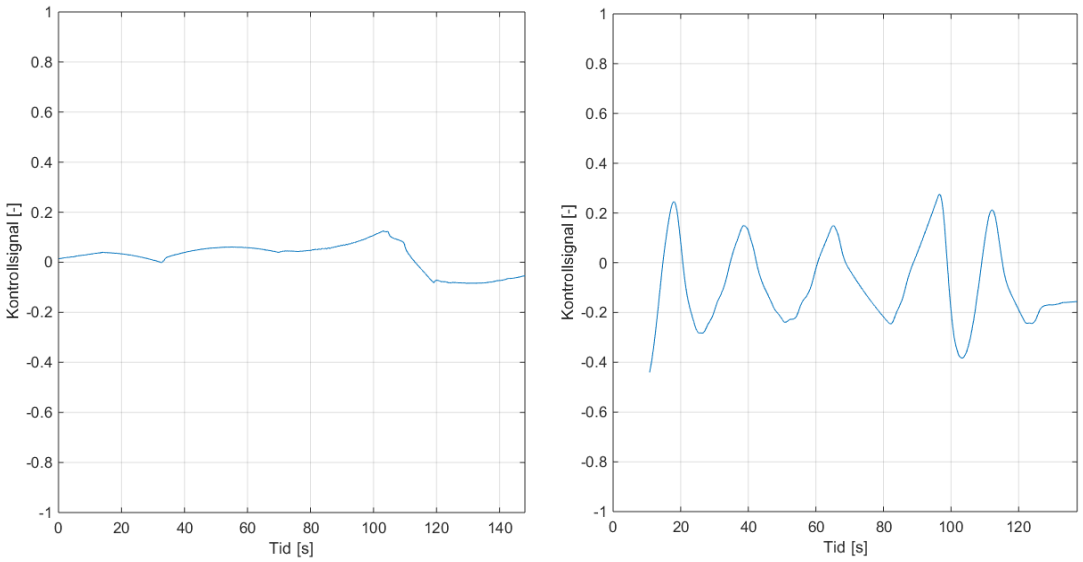


(a) Høy avkutningsfrekvens

(b) Lav avkutningsfrekvens

Figur 3.42: Avvik i foldesylinder ved forskjellig avkutningsfrekvens

Man kan se at avviket er mye høyere når man benytter en lav avkutningsfrekvens på filteret. Disse resultatene gjenspeiler seg også i kontrollsignalet som sendes til ventilen, vist i figur 3.43.



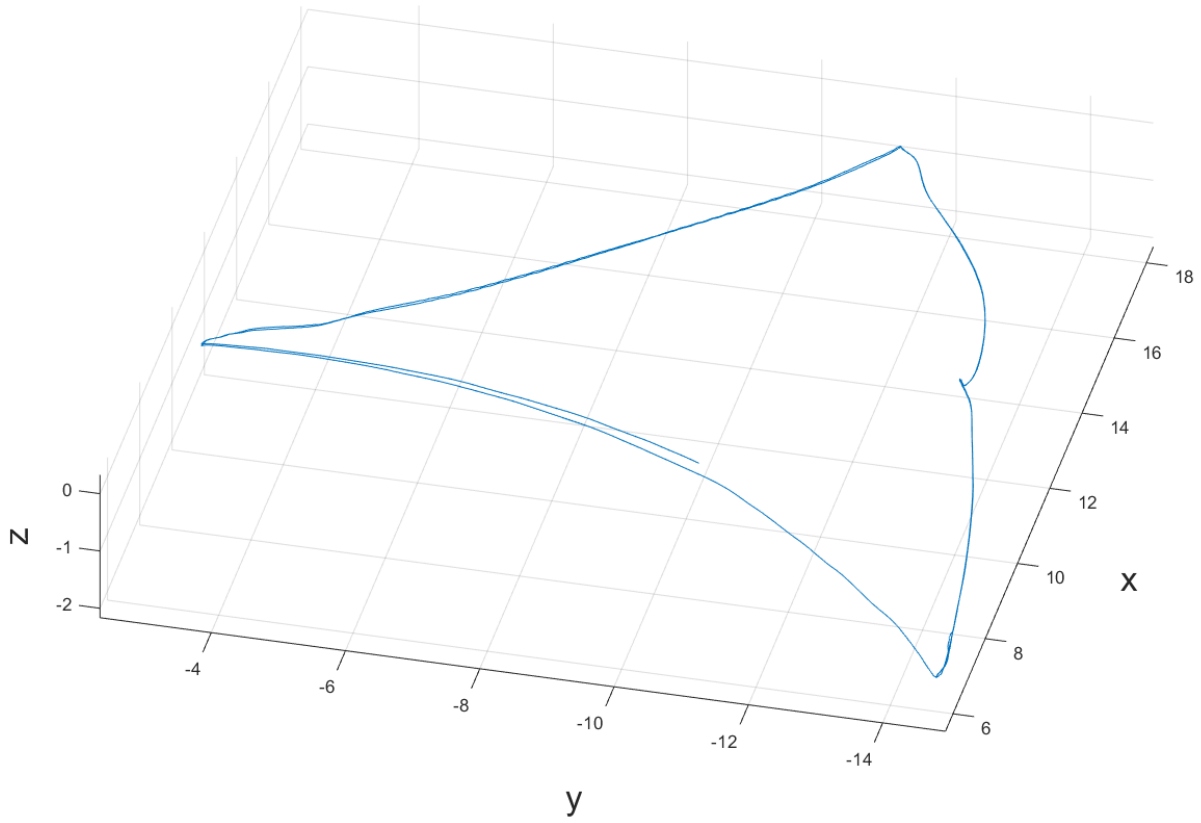
(a) Høy avkutningsfrekvens

(b) Lav avkutningsfrekvens

Figur 3.43: Kontrollsignal til foldesylinder ved forskjellig avkutningsfrekvens

Som vist over, er kontrollsignalet veldig ustabilt ved bruk av lav avkutningsfrekvens på filterne.

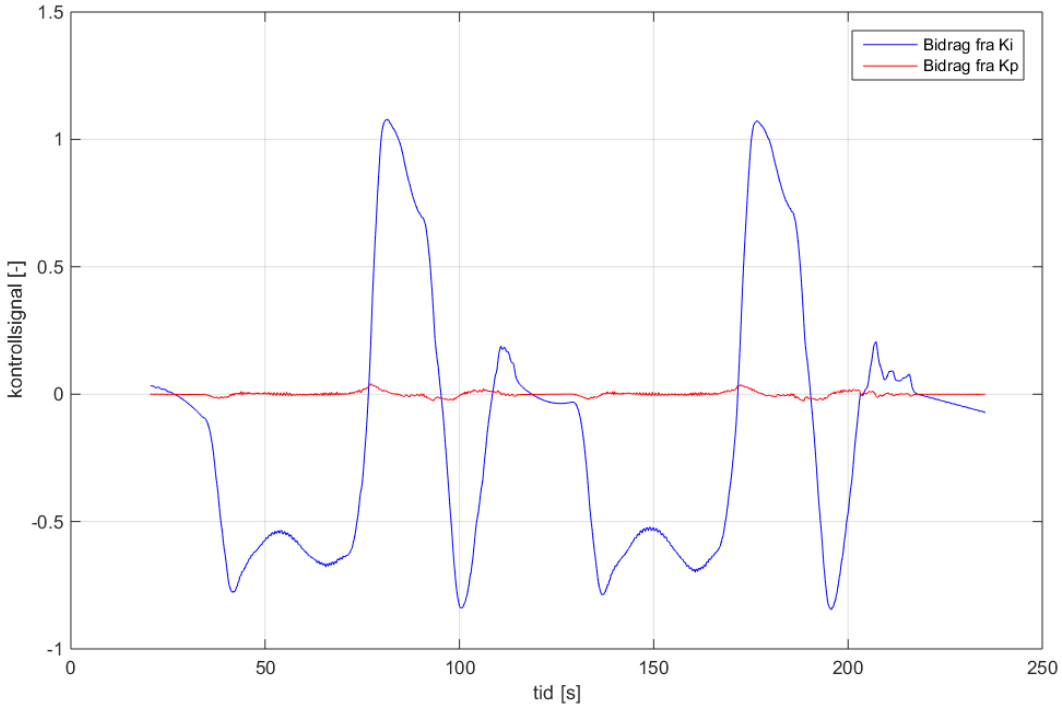
Nå som filteret var testet, ble en ny bane laget for å teste ytelsen til resten av systemet. Banen hadde åtte punkter. Et plot av banen i *xyz* vises i figur 3.44.



Figur 3.44: *xyz*-plot av banen på fysisk test

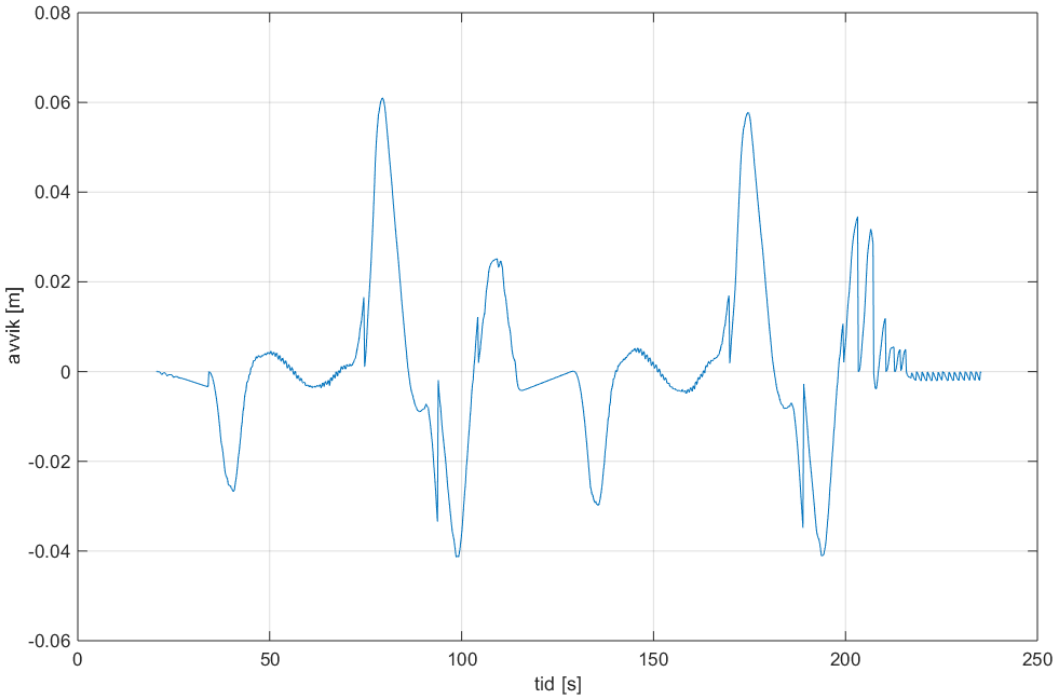
Det viste seg at systemet fikk oversving mellom hvert punkt. Når dette skjedde ved siste punkt i banen så klarte kontrolleren å regulere krantuppen slik at den ender opp i siste punkt med

veldig lavt avvik. Det var dog ikke noe avvik når kranen hadde konstant fart mellom punktene. Integrator reset ble også deaktivert, da dette ga store rykninger i systemet. Den største andelen av kontrollsignalet kommer fra integratorleddet i kontrolleren, vist i figur 3.45.



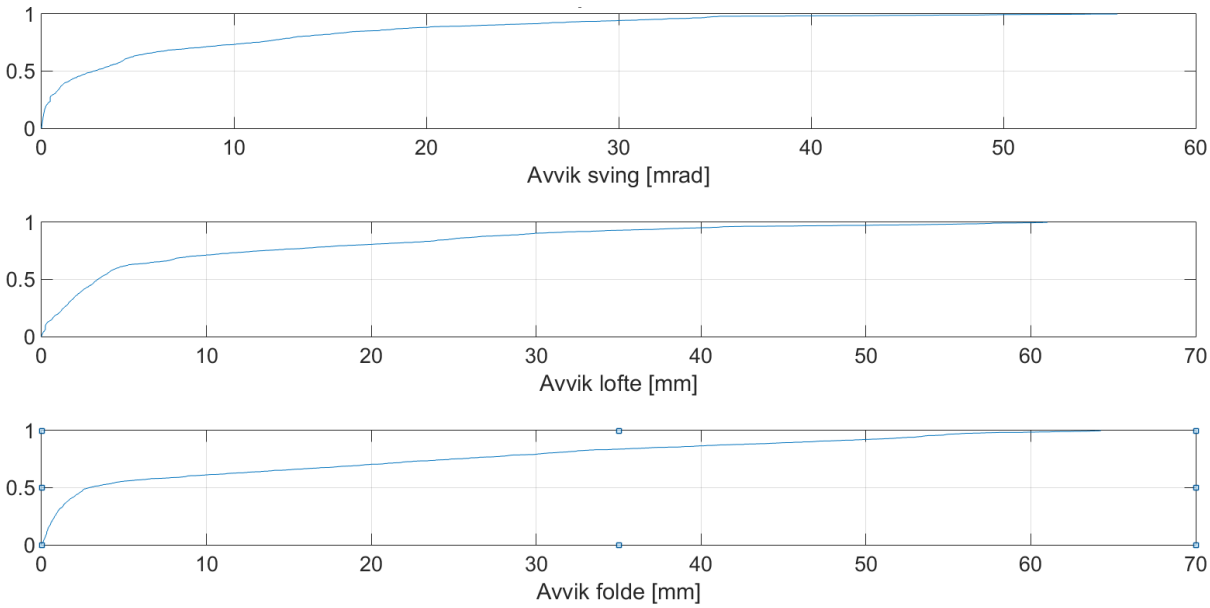
Figur 3.45: Bidrag fra K_p og K_i på kontrollsignalet til løftesylinder

Ved å resette bidraget fra integratoren så vil kontrollsignalet raskt bevege seg mot null, og kranen stopper veldig brått. Systemet ble også testet med forskjellige verdier av K_1, for å se hvordan dette påvirker ytelsen. Med en høy K_1 gikk aktuatorene i metning, og det viste seg at $K_1 = 0.75$ ga grei ytelse. Figur 3.46 viser avviket på løftesylinderen når kranen ble kjørt gjennom banen.



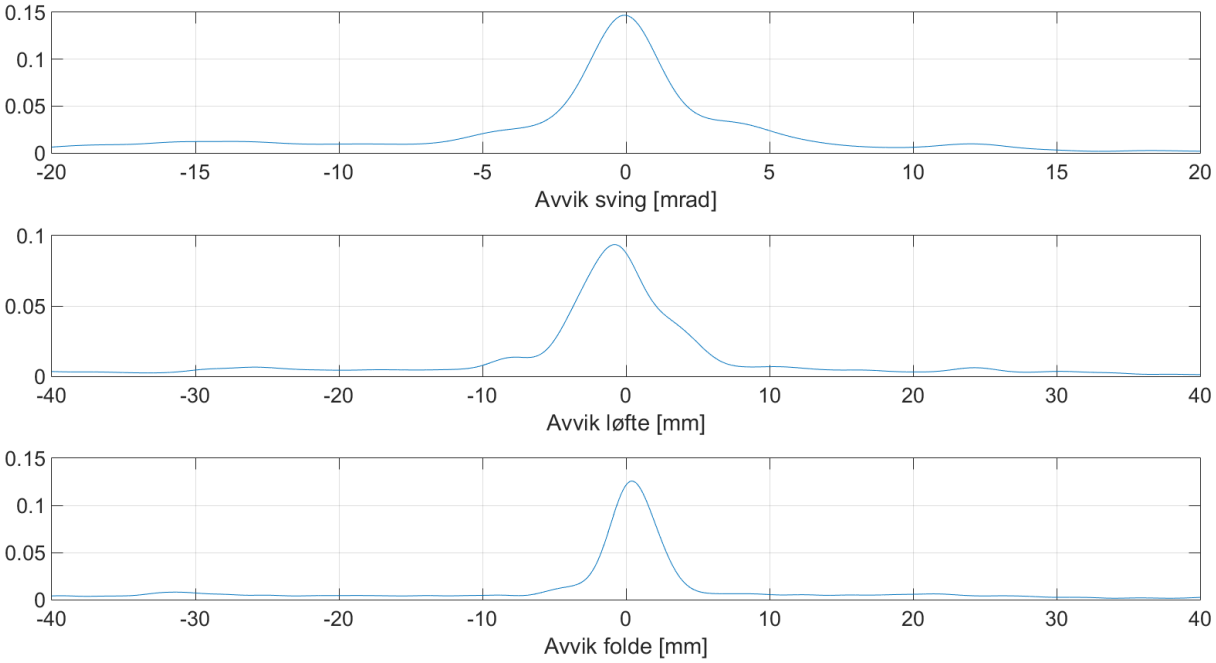
Figur 3.46: Avviket til løftesylinder på fysisk test

Man kan se at man får et stort avvik åtte steder i løpet av testen, for hvert av de åtte punktene. I segmentene mellom hver av punktene er avviket mye mindre. Man kan også se på den kumulative sannsynlighetsfordelingen til avviket under denne testen, vist i figur 3.47.



Figur 3.47: Kumulativt avvik på fysisk test

Denne figurer viser at en stor andel av avviket er under 10 mm og 10 mrad. Man kan også se på sannsynlighetstettheten til feilsignalet, vist figur 3.48.



Figur 3.48: Sannsynlighetstetthet til avvik på fysisk test

Man kan se at sannsynlighetstettheten er normalfordelt rundt 0, som tilsier at det ikke er noe stasjonært avvik under testen.

4. Diskusjon

I dette kapittelet vil resultatene fra kapittel 3 gjennomgås og deres validitet og implikasjoner diskuteres. kapittelet er delt inn slik det skal være lettere å finne løsningen og diskusjonen rundt de enkelte delmål fra kapittel 1.2. Utover implikasjonen og validiteten vil begrensninger og videre utvikling rundt løsningene diskuteres.

4.1 Systemarkitektur

Løsningen i denne oppgaven som inneholder koden til banegeneratorene, kinematikkblokkene og kaskadekontrolleren er en del av et større system. Siden arbeidet med banestyring er i en tidlig fase har arbeidsomfanget i denne oppgaven blitt begrenset slik som vist i figur 3.2 og 3.3. Det har blitt lagt fokus på videre implementering i et mer avansert system. Måten dette har blitt gjort på er muligheten til å endre parametere i koden for eksempel via inngangssignal til funksjonsblokker. Problembeskrivelsen sier krantuppen skal kunne følge et sett med koordinater eller en tredimensjonal kurve i rommet. Disse punktene eller kurven må komme fra et høyere nivå i systemarkitekturen, baneplanleggeren. Baneplanleggeren var utenfor arbeidsomfanget til denne oppgaven, men måtte allikevel tas hensyn til for å få blokkene lenger ned i system arkitekturen til å fungere. Baneplanleggeren var i denne oppgaven kun en liste med punkter innenfor arbeidsrommet til kranen. Det ble ikke tatt hensyn til andre hindringer enn maksimal rotasjon på *dragchain*. Det anbefales som videre arbeid å videreutvikle baneplanleggeren slik den kan ta hensyn til statiske og muligens dynamiske hindringer.

4.2 Banegenerator

Å lage en banegenerator var hovedmålet for denne oppgaven. Banegeneratoren skal generere hastighets- og posisjonsreferanse for en bane mellom to punkter. Dagens system har manuell og aksestyring, ikke banestyring, det vil si at løsningene som presenteres her er starten på automatiseringen av rørhåndteringskraner for NOVN. Mye av de kinematiske begrensingene brukt i banegeneratorene er hentet fra dagens aksestyring. Banegenerator versjon 1 minner om dagens aksestyring som tillater bevegelse langs rette linjer i xyz. Dette er grunnen til versjon 1 presenteres som en løsning i tillegg til versjon 2.

4.2.1 Banegenerator versjon 1

Versjon 1 hadde store problemer som ikke enkelt lar seg løse i programmeringsmiljøet som Siemens SCL tilbyr. Siden banegenerator versjon 1 lager en rette linje i $\mathbb{R}^3$ kan banegeneratoren lage en referanse som går utenfor arbeidsrommet, figur 3.32. Dette kan unngås ved å fylle listen L vist i ligning 3.24 med koordinater som lager en bane innenfor arbeidsrommet. Et problem med mange punkter tett etter hverandre er hvordan banegeneratoren lager referansen mellom to punkter. Den er beregnet for større bevegelser da rampetiden er satt til et fast tall, ikke som i versjon 2 hvor T_r oppdateres. Banegeneratoren mottar disse koordinatene fra baneplanleggeren,

se figur 3.17. Ved videre utvikling av en banegenerator i $\mathbb{R}^3$ må baneplanleggeren også videreutvikles slik arbeidsrommet beregnes som en funksjon av de åtte punktene, figur 2.9, som definerer geometrien til kranen. Det må også tas hensyn til avstanden mellom punktene kranen skal innom. Dersom disse punktene er veldig tett på hverandre må man tillate en mye kortere rampetid hvis ikke vil hastigheten være veldig lav. Å definere et arbeidsrom vil være lettere i et høynivåspråk som for C++ eller MATLAB istedenfor Siemens SCL kode. Hindringer i arbeidsrommet kan være et reelt problem som må tas hensyn til på samme måte som kinematiske begrensinger for å opprettholde ytelse og sikkerhet. På grunn av ulinearitetene mellom krantuppen og aktuatorbevegelsene vil bevegelsene til aktuatorene ikke være optimale når krantuppen tvinges til å følge en rett linje. Versjon 1 tvinger aktuatorene til å gå lenger enn versjon 2. En av de største ulempene med versjon 1 er at maksimal hastighet til krantuppen er definert i *xyz*. Det betyr at hastigheten til aktuatorene variere stort ut i fra kranens positur. Dette betyr at man må sette en veldig konservativ maksimal hastighet for å unngå store avvik. Svakheten til banegenerator versjon 1 er også dens styrke, nemlig at den er visuelt intuitiv siden den beveger seg i en rett linje i $\mathbb{R}^3$. Dette minner mest om dagens aksestyring og kan være ønskelig. Banegenerator versjon 1 presenteres derfor som en mulig løsning.

4.2.2 Banegenerator versjon 2

Versjon 2 er det endelige konseptet som presenteres i denne oppgaven. Den har en god del mer funksjonalitet enn versjon 1. Grunnen til dette er at versjon 1 tidlig i prosjektet ble revidert til versjon 2 og videre arbeid på versjon 1 ble skrinlagt. Banegenerator versjon 2 er fremtidsrettet i den forstand at den har parametere som inngangssignal som kan settes høyere opp i system arkitekturen, for eksempel a_{max} og K_1. K_1 skal forhindre at kranen blir hengende etter referansen. K_1 tillater banegeneratoren kun å bruke en viss prosent av kapasiteten til systemet, 75% i vårt tilfelle. Ved å sette inn en sikkerhetsfaktor får kontrolleren mer spillerom og det tillater den å ta igjen referansen hvis den henger etter. Hvis det skulle vise seg at K_1 er for stor og kranen fremdeles henger etter er det ingen funksjonalitet som kan gjøre noe med dette. Derfor bør K_1 settes konservativt til et lavt tall. Hvis høy fart er essensiell for operasjonen kan en optimal verdi finnes ved nøye analyse. En av svakhetene til begge banegeneratorene er at den ikke oppdaterer banen dersom det er stort avvik på krantuppen og referansen. Hele banen blir beregnes i starten og banegeneratoren antar at kranen klarer å holde følge med. Dette gikk bra under testing på grunn av K_1 som kapittel 3.9.2 viste. Det kunne blitt lagt inn en grense for maksimalt avvik hvor banegeneratoren ventet på kranen hvis denne grensen ble oversteget, men da ender man opp med et stasjonært avvik lik grensen man har satt. En mitigerende effekt for dette er at banegeneratoren bruker nåværende posisjon for å lage ny referanse, og ikke siste punkt i forrige bane. For å forhindre dette problemet kan en adaptiv begrensning K_3 utvikles som bremser banegeneratoren ned og kompenserer for at K_1 er satt for høy. K_3 måtte ha blitt oppdatert hele tiden og ikke bare satt i starten slik som K_1 og K_2.

Banegeneratoren har en fordel at alle tre aktuatorer har samme start og sluttid. Dette gjør at trapes- og trekantprofilene kan følges som planlagt. Dersom pumpen hadde kjørt på fult, og en aktuator ble ferdig før de andre ville mye effekt vært ubrukt resten av det segmentet. K_2 vil sørge for at maksimal oljestrøm ikke overstiges mellom to punkter.

4.3 Kontroller og innreguleringsmetoder

Kontrolleren er en viktig del av banestyringen. Banegeneratoren lager banen, mens kontrolleren sørger for at kranen følger den gitte banen. Selv om det blir presentert to konsept for banestyring, versjon 1 og versjon 2, vil kontrolleren være lik ved bruk av begge versjonene. Fra tidligere oppgave [1] ble det foreslått kontrollstrategier for implementering av banestyring. Disse ble ikke

i sin helhet benyttet i det endelige konseptet.

4.3.1 Design av kontroller

Det ble laget en kaskadekontroller som bruker aktuatorlengder og hastigheter som referanse og tilbakemelding. Kaskadekontrolleren som har blitt designet inneholder en fremoverkobling på indre sløyfe. Ved å bruke en fremoverkobling så kan man eliminere stasjonære avvik ved en rampe som inngangssignal. I tillegg så har det blitt implementert både *anti windup* og *integrator reset* som forsikrer et robust og stabilt system som håndterer raske endringer i settpunktet og metning i aktuatorene. *Anti windup* holder verdien i integratoren under en grense når en aktuator går i metning. At aktuatorene går i metning kan være en reel problemstilling dersom man ønsker å kjøre fort. Utviklingen av den nye kontrolleren har gitt tilfredsstillende resultater noe som resultatene fra HiL simuleringen og fysisk test viser, se figur 3.36 og 3.47.

4.3.2 Metode for optimal innregulering

Det tar ikke lang tid å manuelt finne noen kontrollparametere som kan brukes til å teste funksjonalitet, dog med veldig dårlig ytelse. Manuell innregulering ble i størst mulig grad unngått når det var mulig. Metoden som ble laget for innregulering av simuleringsmodellen i HiL-simulering og In-House testing var å bruke minimeringsalgoritmer i MATLAB til å finne optimale kontrollparametere. Dette gav tilfredsstillende resultater da avviket under både HiL-simulering og In-house testing var veldig lave, se tabell 3.16. Parameterne som ble funnet er vist i tabell 3.4. Hver optimaliseringsalgoritme har funnet parametere med omtrent samme størrelsesorden, som betyr de er i nærheten av det samme lokale minimumspunktet i målfunksjonen. Selve optimaliseringen i MATLAB tar noen timer, dersom alle optimaliseringsalgoritmene brukes, gitt at man har transferfunksjonen fra ventilpådrag til sylinderfart. Optimaliseringstiden kan reduseres betydelig ved å kun bruke én metode og modus, vårt tilfelle var Pattern Search modus B best. Dette gjør det raskt og enkelt å finne parametere til kontrolleren som gir lavt avvik. Ved å bruke disse parameterne vil man få et inntrykk av hvilken ytelse systemet er i stand til å levere. De optimale kontrollerparameterne er basert på en forenklet modell. De optimaliserte kontrollparameterne ble brukt i HiL simuleringen og In-House test, men ikke på den fysiske kranen. Det var flere grunner til at de optimaliserte parameterene ikke gav tilfredsstillende ytelse på den fysiske kranen. Kranen hadde ikke fullført intern testing og manglet tilbakemelding på ventilposisjon. Dette skapte et stort dødbånd og hysterese på ventilene som bidrar til ytterligere forskjeller på modell og kran. Ved en ferdig testet kran med ventiltilbakemelding og en mer detaljert modell kan det være mulig å finne kontrollparametere som kan brukes direkte på det fysiske systemet. Dette ville redusert idriftsettelsestiden.

4.4 Testing

Testing av banestyringen på kranen ble gjort på to måter; Hil-simulering og fysisk test på en ny rørhåndteringskran. Som forklart i kapittel 3.9.3 måtte koden justeres i en *inhouse test* for å passe den fysiske kranen inngangssignalene. Løsningen er lagt opp til å enkelt kunne bytte spesifikasjoner fra kran til kran ved hjelp av datablokkene DB4004 og DB4005 som beskrevet i kapittel 3.8, selv om dette ikke var nødvendig denne testen i Søgne.

4.4.1 HiL-testing

Utvidet testing har kun blit utført på banegenerator versjon 2. I motsetning til versjon 1 vil versjon 2 alltid holde seg innenfor arbeidsrommet da den kun styrer aktuatorene mellom minimum og maksimum. Arbeidsområdet har gått fra et komplisert tre-dimensjonalt rom til tre en-dimensjonale akser. Figurene 3.33, 3.34 og 3.35 viser at denne banegeneratoren sørget for

å holde kranen innenfor arbeidsområdet samt vært i store deler av det. Tabell 3.16 viser den beste løsningen hvor summen av RMS verdiene til de tre aktuatorene var minst viste seg å være Patternsearch Modus B.

I boksen under er et utdrag fra NOVN sin teststandard for aksestyring som sier hvor mye avvik krantuppen kan ha under testing.

Typical position error values:0m-0,10m.
Errors exceeding 0,10m should only be short peaks.
Typically we see short peaks up to around 0,15m in 1-1,5 seconds.
Maximum position error should not exceed 0,15m at test stand.
Limit for PLC alarm > 0,2m in 3 seconds

Avviket som svingmotoren bidrar med er enkel å regne ut da den der oppgitt i radianer. Ligning 4.1 viser hvor mye avvik sving bevegelsen bidrar meg ved maksimal arbeidsradius.

$$e_s = 23572\ mm \cdot \sin\left(\frac{1.92}{1000}\right) = 45.26\ mm \tag{4.1}$$

Sving bevegelsen bidrar maksimalt med 45.26 mm avvik i xy planet. Siden banegeneratoren arbeider i aktuatorrommet er det ikke helt klart hvor stort avviket i $x'z'$ er ut fra avviket i aktuatorrommet. Maksimalt avvik i $x'z'$ beregnes ut fra *worst case*. *Worst case* er hvis avviket er på maks samtidig som kranen står i den posituren som gir størst endring i $x'z'$ ved en endring i aktuatorene. Det er heller ikke helt klart i hvilket punkt dette skjer. Det antas at det er et av de syv punktene vist i figur 3.32. Standarden sier en typisk posisjonsfeil er mellom 0 cm og 10 cm. Metoden for å finne *worst case* positur var å bruke invers kinematikk fra figur 3.1 på hvert av de syv punktene og sende et inngangssignal på ± 10 cm og se hvor stor summen av endringene til de to aktuatorene er. Tabellen under viser hvor stort avvik i aktuatorene ble ved å endre X og Z referansen med 10 cm. Dette ble gjort for alle syv punkter vist i figur 3.32. Tabellen viser prosessen for punkt 2, $(19.6\ m, 0\ m, 15.6\ m)$.

Tabell 4.1: Avvik i aktuatorer ved 10 cm endring i $x'z'$ for punkt 2

Punkt 2	Akse	Δx_l	Δx_f	Sum
+10 cm	X	12 mm	13 mm	25 mm
	Z	0 mm	9 mm	9 mm
-10 cm	X	12 mm	13 mm	25 mm
	Z	0 mm	9 mm	9 mm

For å finne punktet med *worst case* positur ble feilen $e_{l/f}$ beregnet for alle syv punkter som vist i ligning 4.2.

$$e_{l/f} = \min\left(\Delta x_l + \Delta x_f\right) \tag{4.2}$$

Punkt 2 hadde lavest $e_{l/f}$, det vil si når kranen står på maksimal høyde er det størst avvik i $x'z'$ ved en endring i sylindrene. Som tabellen viser er endringen som oppstår i sylindrene ved 10 cm avvik større enn summen av maksimalt avvik vist i tabell 3.16 under Pattern Search modus B.

$$3.83\ mm + 2.96\ mm < 9\ mm \tag{4.3}$$

Det er viktig å merke seg at dette er absolutt verste tilfelle, både maksimalt avvik på flere aktuatorer og kranens positur må være i punkt 2 for å komme i nærheten av grensen satt i standarden. Dette betyr at løsningen holder seg innenfor kravene som stilles i standarden. Dette er ikke tilfelle ved numerisk derivering av vinkelmålingen. Ved høyere oppløsning på sensoren ville avviket på løfte- og foldesylinder i tabell 3.17 gått ned. For videre utvikling av banestyring

bør de kinematiske regneblokkene fra aktuatorlengde til *xyz* regnes ut analytisk for å gi et klarere bilde over forholdet mellom aktuatoravvik og avvik i *xyz*. I tillegg anbefales det at NOVN investerer i nye sensorer med høyere oppløsning.

Figur 3.36 viser at det er en liten andel av målingene som er i nærheten av maksimalverdien som har blitt brukt til å finne maksimalt avvik i *xyz*. Figur 3.37 viser spredningen av avviket rundt null avvik. Løfte- og foldesylinderen ser ut til å følge en normalfordeling rundt null avvik som er forventet. Sving bevegelsen ser ut til å ha et stasjonært avvik nå cirka $\pm 0.25\ mrad$. Dette kan komme av de to toppene er veldig nær oppløsningen til sensoren, K_{floor}. Figur 3.36 viser at cirka 40% av dataen er under $K_{floor} = 0.1812\ mrad$.

4.4.2 Fysisk-testing

Den fysiske testen i Søgne ble utført på en kran som ikke hadde gjennomgått den interne testen. Dette gjorde at innreguleringsmetodene som er utvilket ikke kunne analysere på en god nok måte. Kranen ble dog innregulert manuelt i løpet av kort tid, dette viser hvor robust og fleksibelt selve kontrolleren er, og hvor enkelt systemet kan innreguleres. Selv om kontrolleren er kompleks og inneholder mange parametere, så gjør kontrollerens oppbygging at aktuatorene er uavhengig av hverandre, og kan innreguleres hver for seg.

Selv om kranen manglet tilbakemelding fra ventilposisjonen, så kunne løsningen hvor man deriverer og filtrerer vinkelen brukes for å lage tilbakemelding til den indre kontrollsløyfen. Dette virket godt på kranen, og er en løsning som kan benyttes på andre kraner uten tilbakemelding på ventilposisjonen. Ytelsen til filterne ble også analysert, og det viste seg at løsningen med optimaliserte avkutningsfrekvenser er en god løsning, spesielt i forhold til den løsningen som blir brukt på dagens aksestyring.

Selv med manuelt innregulerte kontrollparametere på en kran med mangler, så viste resultatene at det ikke ble noe stasjonært avvik når kranen ble kjørt i en bane. Det var dog noe oversving etter hvert segment. Dette skyldes at kranen ikke var ordentlig testet. Oversvingen kommer blant annet av at ventilene hadde mye dødbånd og hysterese. På grunn av hysteresen så vil ikke ventilene lukke seg i tide, og krantuppen vil kjøre forbi settpunktet. Dette ville ikke vært tilfelle hvis kranen hadde hatt fungerende sensorer på ventilposisjon. Ventilene styres av et DMA-kort som skal ta seg av hysteresen, og dødbåndet vil bli kompensert for i programkoden. Dette vil føre til at kontrollsignalet og oljestrømmen gjennom ventilene vil bli tilnærmet proporsjonalt. I tillegg så vil DMA-kortet regulere ventilene hurtig, som vil si at man kan ha mer aggressive kontrollparametere. Siden mesteparten av kontrollsignalet kommer fra integratoren i kontrolleren, så vil K_i ha påvirkning på hvor hurtig kontrollsignalet kan forandre seg. Hvis man har en mer aggresiv innregulering med høy K_i, så kan kontrollsignalet snu retning mye raskere, og dette vil også gjøre at krantuppen kan følge en referanse som endrer seg raskt. Dette vil redusere eventuell oversving kraftig. Det ble besluttet å deaktivere integrator reset, nettopp fordi store deler av kontrollsignalet kom fra integratoren. Hvis kranen hadde hatt fungerende sensorer, så kunne denne løsningen vært testet ut bedre.

Selve banegeneratoren fungerte godt. Banen som blir generert skaper en glatt akselerasjon og retardasjon. Dette vil redusere belastningen på kranen. I tillegg så er maksimal tillatt akselerasjon en justerbar parameter, så denne kan reduseres hvis kranen får for høy akselerasjon. Siden banegeneratoren jobber i aktuatorrommet, så vil aktuatorene holde konstant hastighet i mesteparten av banen. Dette er en fordel, siden referansen på indre sløyfe er konstant og kontrolleren trenger ikke jobbe så hardt for å følge et varierende settpunkt. Et konstant settpunkt vil redusere oscillasjoner i systemet, og man kan tillate en mer aggressiv innregulering.

5. Konklusjon

Hovedproblemstillingen i denne oppgaven var å videreutvikle banestrying for NOVN sine hydrauliske offshore rørhåndteringskraner. Det ble satt flere delmål som implementering av kinematikk og kontroller i PLS kode, innregulering av kontrollparamentere ved hjelp av minimeringsalgoritmer i MATLAB samt finne en løsning på problematikk med støy ved derivering av vinkelmåling når sensoren har lav oppløsning.

For å kunne innregulere kontrollparameterne ble en transferfunksjon av systemet estimert. Transferfunksjonen kommer fra modellen av kranen som ble laget. Modellen er blant annet basert på dokumentasjonstegninger for det hydrauliske systemet til en rørhåndteringskran. Kontrollparameterne som ble funnet er P-og PI-forsterkningene til kaskadesløyfen på alle tre aktuatorene; tilsammen ni parametere. Sensorene som måler vinklene til bommene har relativt lav oppløsning som viser seg å være problematisk ved derivering av signalet. Det ble implementert et 2. ordens Butterworth filter som filtrerer vinkelen før den deriveres numerisk. Dette gav mindre støy i vinkelhastighet som igjen gjør vinkelhastighet mer anvendelig. Banegeneratoren som lager referansesignalet til kontrolleren genererer en rett linje mellom to aktuatorlengder og ikke en rett linje mellom to punkter i $\mathbb{R}^3$. Dette sikrer at banegeneratoren ikke generer koordinater utenfor arbeidsrommet til kranen i tillegg til den korteste og mest effektive veien for aktuatorene vil bli valgt. Banestyringen ble programmert i Siemens SCL og implementert på en Siemens CPU315-2 PN/DP. Testingen ble gjort med LabVIEW Real Time Target i en HiL-simulering. Resultatene viser minimalt avvik fra aktuatorene som betyr at kranen klarer å følge banen. Avviket på aktuatorene er i størrelsesorden millimeter som i *xyz* betyr at kranen holder seg innefor posisjonsavviket som er akseptabelt ved brukt av dagens aksestyring. Når resultatene fra HiL-simuleringen var tilfredsstillende ble en *In-house test* mot en kransimulator utført hos NOVN. Denne testen ble gjort for å forberede koden til den fysiske kranen, inngang og utgangssignal blir satt til riktig i forhold til I/O kort og lignende. Den 7. og 8. Mai 2015 ble det utført testing av banestyring på en fysisk rørhåndteringskran i Søgne med prosjektnummer G4554. Som beskrevet i diskusjonen var ikke den interne testingen av kranen ferdig da banestyringen ble testet. I tillegg var det ikke mulig å bruke samme konfigurasjon på kontrollsystemet som under HiL testing da det manglet tilbakemelding på ventilposisjonene. Det ble i stedet brukt løsningen hvor de deriverte vinkelmålingene kunne brukes som tilbakemelding istedenfor ventilposisjon slik at testen lot seg gjennomføre. Fraværet av ventilposisjonssensoren bidrar dog til å redusere ytelsen og stabiliteten til kranen betraktelig. Kontrollparameterne som ble funnet via minimeringsalgoritmene kunne ikke brukes på den fysiske kranen, grunnet kranens mangler, men også trolig siden kranen er mer kompleks enn modellen. Testen gav allikevel tilfredsstillende resultater, kranen kunne følge en gitt bane i rommet med avvik på noen millimeter i aktuatorrommet under bevegelse og topper på noen centimeter ved bytte til nytt settpunkt. Dette avviket kan mest sannsynlig reduseres ved å fullføre interntest av kranen og sette inn tilbakemelding på ventilposisjon.

Ved å bruke banestyring, og spesielt banestyring som arbeider med rette linjer i aktuatorrommet, vil kjøringen bli mer effektiv. Den vil også være konsekvent i forhold til en operatør som

er påvirket av menneskelige faktorer. I følge NOVN er store deler rørhåndteringkranens arbeid veldig repetitivt, ved å introdusere banestyring kan disse delene bli gjort mer effektive. Dagens kranfører vil overvåke under banestyringen og overta ved hjelp av manuell styring eller aksestyring på de delene hvor det ikke er hensiktsmessig å bruke banestyring.

Med tanke på videre arbeid anbefales det å legge vekt på høyere nivå i systemarkitekturen som for eksempel baneplanlegging, dette innebærer kollisjonsunngåelse. Resultatene for avvik er oppgitt i aktuatorlengder som ikke er lette å tolke i XYZ, det anbefales å beregne analytisk forholdet mellom aktuatorlenger og posisjon i $\mathbb{R}^3$ for enkelt å bedømme ytelsen til systemet.

Dagens system er enten manuell styring av ventilpådraget eller såkalt aksestyring, som tillater kjøring langs rette linjer i *xyz*. Løsningen som presenteres i denne oppgaven er neste steg mot automatisering av rørhåndteringskranene til NOVN. Testingen som har blitt gjort 7. og 8. Mai viser at konseptet virker og kan videreutvikles til et ferdig produkt.

Figurer

Tabeller

Bibliografi

[1] J. O. Omland and P. Semb, "Utvikling av banestyring for hydraulisk offshorekran," Master's thesis, Universitetet i Agder, 2013.

[2] K. J. Åström and T. Hägglund, *PID Controllers: Theory, Design and Tuning*. 1995.

[3] A. Frederiksen, "Model-based design of advanced motor control systems," 2013.

[4] L. International, "Software verification & validation techniques and technologies," 2013.

[5] I. Headquarters, "Introduction to simulationX." Introduction to SimulationX, 2015.

[6] National-Instruments, "Introduction to G programming," 01 2009.

[7] H. Berger, *Automating with STEP 7 in STL and SCL*. Publicis Corporate Publishing, 4 ed., 2007.

[8] MathWorks, "The language of technical computing," 2 2015.

[9] MathWorks, "Automatically tune pid controller gains." http://se.mathworks.com/discovery/pid-tuning.html, 05 2015.

[10] MathWorks, "Fmincon documentation." `http://se.mathworks.com/help/optim/ug/fmincon.html`. 13-april-2015.

[11] MathWorks, "Fminsearch documentation." `http://se.mathworks.com/help/matlab/ref/fminsearch.html`. 13-april-2015.

[12] MathWorks, "Pattern search documentation." `http://se.mathworks.com/help/gads/patternsearch.html`. 13-april-2015.

[13] MathWorks, "Genetic algorithm documentation." `http://se.mathworks.com/help/gads/ga.html`. 13-april-2015.

[14] ElectronicsHub, "Butterworth filter." `http://www.electronicshub.org/butterworth-filter/`. 02-Mars-2015.

[15] A. Oppenheim, *Discrete Time Signal Processing*. 2010.

[16] O. Birkeland, "Kinematikk for aksestyring," *National Oilwell, Tech. Rep*, 2013.

[17] P. Kruchten, "Architectural blueprint the 4+1 view model of software architecture." IEEE SOftware 12 (6), 11 1995. 42-50.

[18] M. K. Bak and M. R. Hansen, "Analysis of offshore knuckle boom crane - part two: Motion control," 2014.

[19] NOVN, "Load chart data for plc.xmcd page a03.6 of 9." Internt dokument. OC1891PL-N-RD-003-rev2-A03.

A. Oppgavebeskrivelse

Utvikling av banestyring for hydraulisk offshorekran

Oppgavebeskrivelse til masteroppgave i Mekatronikk ved UiA for studentene Tor-André Rettedal Hetland og Konrad Johan Jensen i samarbeid med National Oilwell Varco Norway

Introduksjon
National Oilwell Varco Norway (NOVN) er en av verdens ledende leverandører av rørhåndteringskraner, offshorekraner og hivkompenserte subsea-kraner til skip og plattformer. Rørhåndteringskranene danner ledd i et større system av elektro-hydraulisk utstyr som delvis automatisert og med stor presisjon håndterer rør og risere på offshore borerigger og boreskip. Det er store krav til effektivitet og sikkerhet i rørhåndteringsprosessen.

Bakgrunn for oppgaven
For å lette kranoperatørens arbeid og øke graden av automatisering, har NOVN utviklet og implementert det vi kaller aksekontroll på den siste generasjonen av rørhåndteringskraner. Det innebærer at kranas hovedbom-, knekkbom- og svingbevegelser reguleres slik at krantuppverktøyet beveger seg lineært langs en av aksene til et predefinert, fartøyfast koordinatsystem. NOVN har også begynt arbeid med å utvikle banestyring for kranene, hvor operatøren eksempelvis kan definere et sett med punkter/waypoints i rommet som krantuppen skal følge. Dette kan gi store tidsbesparelser i arbeid hvor man utfører en repetitiv oppgave. Et eksempel er ved håndtering og frakt av rør fra pipedeck til catwalk, som idag er eneste ledd i rørhåndteringsprosessen som fortsatt utføres manuelt. NOVN sitt arbeid innen dette feltet har foreløpig kun vært på en teoretisk basis. NOVN ønsker å videreføre dette arbeidet og å ta neste steg mot praktisk implementasjon.

Målsetting
Målet med oppgaven er å nå neste trinn mot fullautomatisert rørhåndtering ved hjelp av banestyring. Krantuppen skal kunne følge en gitt bane i rommet basert på et sett med koordinater eller en tredimensjonal kurve i rommet. Arbeidet vil gå ut på å videreutvikle og implementere programkode for banestyringen. NOVN vil gjerne ha et sluttresultat som kan brukes på flere forskjellige krantyper, og må ikke være fastbundet til ett spesifikt mekanisk design, eller ett sett med hydrauliske komponenter.

Metodikk og verktøy
Kandidatene kan bruke eksisterende materiell fra tidligere arbeid med banestyring. Dette vil være:

- Tidsserier logget i eksisterende kraner for å utvikle transferfunksjoner mellom ventilpådrag og kranbevegelser
- 3D-modeller av NOVN sine kraner
- Mekaniske og hydrauliske modeller modellert i SimulationX
- Programkode for kontroll av kranen, i form av en mal for en rørhåndteringskran
- Reguleringsstrategier for banestyring
- Kinematiske modeller

Dette kan benyttes og modifiseres for deretter å lage programkode og reguleringssløyfer til kranen i Simatic Step7. Den resulterende programkoden kan verifiseres ved hjelp av en Hardware-in-the-loop

simulering. Dette vil gi god innsikt på hvordan systemet vil fungere i drift, og vil åpne for videre betraktninger og diskusjon rundt programkoden og hvordan den skal implementeres for optimal ytelse.

En praktisk implementering av banestyring åpner også for betraktninger med tanke på instrumenteringen og andre faktorer som spiller inn på NOVN sine kraner.

Fysisk test av programkode på en rørhåndteringskran er også aktuelt hvis tiden tillater.

Resultater og sluttprodukt
NOVN ønsker at følgende skal leveres når masteroppgaven er fullført:

- Programkode og reguleringsalgoritmer implementert i Simatic Step7, skrevet i Structured Control Language
- Dokumentasjon fra simuleringer og eventuell fysisk test
- Prosjektrapport som dokumenterer arbeidet

Annet
NOV stiller alt som behøves av PC- og PLS-ustyr, programvare og lokaler til disposisjon for kandidatene.

NOVN kontaktpersoner
Per Øyvind Strandmyr-Ødegaard, Manager, Controls, Crane & Winch, +47 38191998, PerOyvind.Strandmyr-Odegaard@nov.com

Petter Semb, Design Engineer, Controls, Crane & Winch, +47 38191519, Petter.Semb@nov.com

B. Kode skrevet i MATLAB

B.1 Program for PID Tuner

```
clc
clear
close all

z_sving=[];
p_sving=[-170+82.2i -170-82.2i -4.09+17.7i -4.09-17.7i];
k_sving=753280;%total gain = 0.064

z_lofte=[-399 -292 -31.6];
p_lofte=[-170+82.2i -170-82.2i -4.09+17.7i -4.09-17.7i -440 -297 -208
    -46.4 -8.69];
k_lofte=1.4014e9; %total gain = 0.04

z_folde=[-297 -227 -81.4];
p_folde=[-170+82.2i -170-82.2i -4.09+17.7i -4.09-17.7i -440 -297 -208
    -46.4 -8.69];
k_folde=1.0578e9;% total gain = 0.045

sys_sving_zpk=zpk(z_sving,p_sving,k_sving);
sys_lofte_zpk=zpk(z_lofte,p_lofte,k_lofte);
sys_folde_zpk=zpk(z_folde,p_folde,k_folde);

sys_sving_tf=tf(sys_sving_zpk);
sys_lofte_tf=tf(sys_lofte_zpk);
sys_folde_tf=tf(sys_folde_zpk);

%pure integrator from velocity to position
int=tf(1,[1 0]);
%pi-controller - tf([kp ki],[1 0]) - parallel form
cv_sving=tf([1 108],[1 0]);
cv_lofte=tf([1 109],[1 0]);
cv_folde=tf([1 85],[1 0]);

inner_sving=feedback(cv_sving*sys_sving_tf,1)*int;
inner_lofte=feedback(cv_lofte*sys_lofte_tf,1)*int;
inner_folde=feedback(cv_folde*sys_folde_tf,1)*int;

%pi-controller - tf([kp ki],[1 0]) - parallel form
cp_sving=tf([2 0],[1 0]);
```

```
cp_lofte=tf([2  0],[1  0]);
cp_folde=tf([2  0],[1  0]);

outer_sving=feedback(cp_sving*inner_sving,1);
outer_lofte=feedback(cp_lofte*inner_lofte,1);
outer_folde=feedback(cp_folde*inner_folde,1);

subplot(1,2,1)
step(outer_folde)
grid
title('Closed loop step')
subplot(1,2,2)
bode(outer_folde)
grid
title('Closed loop bode')


figure('Color',[1 1 1]);
subplot(1,2,1)
step(inner_folde/int);
xlabel('')
xlabel('t [s]')
ylabel('')
grid
subplot(1,2,2)
step(outer_folde);
xlabel('')
xlabel('t [s]')
ylabel('')
grid
```

B.2 Filteroptimalisering - hovedprogram

```
clc
% clear
close all

optionsga=gaoptimset(@ga); %create default settings for optimization
optionsga=gaoptimset(optionsga,'PlotFcns',{@gaplotbestf},'Display','
    iter','MutationFcn',@mutationadaptfeasible);

optionsfmincon=optimset(@fmincon); %create default settings for
    optimization
optionsfmincon=optimset(optionsfmincon,'PlotFcns', {@optimplotx
    @optimplotfval},'Display', 'iter');

optionsfminsearch=optimset(@fminsearch); %create default settings for
     optimization
optionsfminsearch=optimset(optionsfminsearch,'PlotFcns', {@optimplotx
     @optimplotfval},'Display', 'iter');
```

```
optionspatternsearch=psoptimset(@patternsearch); %create default
    settings for optimization
optionspatternsearch=psoptimset(optionspatternsearch,'PlotFcns', {
    @psplotbestx @psplotbestf},'Display', 'iter');

dt=0.01; %PLC cycle time

% 0.0259 degrees MB - sensor resolution estimated from PLC log
% 0.0269 degrees KB - sensor resolution estimated from PLC log
% slew = 0.001038 degrees per pulse, from 4000 ppr encoder and slew
    gear

sensor=2; % 1-mb    2-kb    3-slew
if sensor==1
K_floor=0.0259;
elseif sensor==2
K_floor=0.0259;
elseif sensor==3
K_floor=0.01038;
end


fs=1/dt;
ws=2*pi*fs;
lb=[1e-5];
ub=[ws/2;]; %Half of the sampling frequency
lb_ga=[lb 1 1];
ub_ga=[ub 6 3];
x0=[20];


% Mode 1 - Filtrering før og etter derivering
% Mode 2 - Filtrering kun før derivering
% Mode 3 - Filtrering kun etter derivering

% [x_fmincon,fval_fmincon]=fmincon(@(x) call_filter(x,2,2,K_floor,0,
    sensor),x0,[],[],[],[],lb,ub,[],optionsfmincon)



for mode=1:3
for order=1:6
    ga_select=0;
    [x_fmincon,fval_fmincon]=fmincon(@(x) call_filter(x,order,mode,
        K_floor,ga_select,sensor),x0,[],[],[],[],lb,ub,[],
        optionsfmincon);
    x_fmincon1(order,mode)=x_fmincon;
    fval_fmincon1(order,mode)=fval_fmincon;
end
end
% for mode=1:3
```

```
% for order=1:6
%     ga_select=0;
%     [x_fminsearch,fval_fminsearch]=fminsearch(@(x) call_filter(x,
    order,mode,K_floor,ga_select,sensor),x0,optionsfmincon);
%     x_fminsearch1(order,mode)=x_fminsearch;
%     fval_fminsearch1(order,mode)=fval_fminsearch;
% end
% end
% for mode=1:3
% for order=1:6
%     ga_select=0;
%     [x_patternsearch,fval_patternsearch]=patternsearch(@(x)
    call_filter(x,order,mode,K_floor,ga_select,sensor),x0,[],[],[],[],
    lb,ub,[],optionspatternsearch);
%     x_patternsearch1(order,mode)=x_patternsearch;
%     fval_patternsearch1(order,mode)=fval_patternsearch;
% end
% end
% ga_select=1;
% [x_ga,fval_ga]=ga(@(x) call_filter(x,0,0,K_floor,ga_select,sensor),
    max(size(lb_ga)),[],[],[],[],lb_ga,ub_ga,[],[2 3],optionsga);

disp('Fmincon function value')
disp(fval_fmincon1)

% disp('Fminsearch function value')
% disp(fval_fminsearch1)
%
% disp('Patternsearch function value')
% disp(fval_patternsearch1)
%
% disp('GA function value')
% disp(fval_ga)

disp('Fmincon parameters')
disp(x_fmincon1)

% disp('Fminsearch parameters')
% disp(x_fminsearch1)
%
% disp('Patternsearch parameters')
% disp(x_patternsearch1)
%
% disp('GA parameters')
% disp('      wn       order       mode')
% disp(x_ga)



% optionsfmincon=optimset(optionsfmincon,'PlotFcns', {},'Display', '
    off');
```

```
% K_floor_plot1=[1e-10:0.00001:0.001];
% for i=1:max(size(K_floor_plot1))
%     x_plot1(i)=fmincon(@(x) call_filter(x,2,2,K_floor_plot1(i),0,
    sensor),x0,[],[],[],[],lb,ub,[],optionsfmincon);
% end
% K_floor_plot2=[1e-10:0.001:0.1];
% for i=1:max(size(K_floor_plot2))
%     x_plot2(i)=fmincon(@(x) call_filter(x,2,2,K_floor_plot2(i),0,
    sensor),x0,[],[],[],[],lb,ub,[],optionsfmincon);
% end
% %%f=a*x^b+c
% a=0.4585;
% b=-0.7062;
% c=7.496;
% x1=a*K_floor_plot1.^b+c;
% x2=a*K_floor_plot2.^b+c;
% figure
% plot(K_floor_plot1,x_plot1)
% hold on
% %% plot(K_floor_plot1,x1)
% xlabel('K_{floor}')
% ylabel('w_c')
% grid
% figure
% plot(K_floor_plot2,x_plot2)
% hold on
% %% plot(K_floor_plot2,x2);
% xlabel('K_{floor}')
% ylabel('w_c')
% grid
%
% K_floor1=[0.005 0.01 0.05 0.1];
% for i=1:4
% for mode=1:3
% for order=1:6
%     ga_select=0;
%     [x_fmincon,fval_fmincon]=fmincon(@(x) call_filter(x,order,mode,
    K_floor1(i),ga_select,sensor),x0,[],[],[],[],lb,ub,[],
    optionsfmincon);
%     x_fmincon2(order,mode,i)=x_fmincon;
%     fval_fmincon2(order,mode,i)=fval_fmincon;
% end
% end
% end
% K_floor1
% mins=[min(min(fval_fmincon2(:,:,1))) min(min(fval_fmincon2(:,:,2)))
    min(min(fval_fmincon2(:,:,3))) min(min(fval_fmincon2(:,:,4)))]
% means=[mean(mean(fval_fmincon2(:,:,1))) mean(mean(fval_fmincon2
    (:,:,2))) mean(mean(fval_fmincon2(:,:,3))) mean(mean(fval_fmincon2
    (:,:,4)))]
% mode2order2s=[fval_fmincon2(2,2,1) fval_fmincon2(2,2,2)
```

```
    fval_fmincon2(2,2,3) fval_fmincon2(2,2,4)]
```

B.3 Filteroptimalisering - objective function

```
function s=call_filter(x,order_in,mode_in,K_floor,ga,sensor)

if ga==1;
    %integer problems will not always satisfy bounded contraints
    %wn must be positive
    wn=abs(x(1))+1e-100;

    order=x(2);
    mode=x(3);
else
    %fminsearch does not have bounds
    %wn must be positive
    wn=abs(x)+1e-100;

    order=order_in;
    mode=mode_in;
end

Fs=100; %Hz
[b,a]=butter(order,wn,'s');
[z,p,k]=butter(order,wn,'s');
sys=tf(b,a);
opt = c2dOptions('Method', 'tustin');
sys3=c2d(sys,1/Fs,opt);
[num,den]=tfdata(sys3,'v');

start=0;
stop=100;
dt=0.01;
t=[start:dt:stop];
steps=max(size(t));

xk1=0;
xk2=0;
xk3=0;
xk4=0;
xk5=0;
xk6=0;
yk1=0;
yk2=0;
yk3=0;
yk4=0;
yk5=0;
yk6=0;
x1k1=0;
x1k2=0;
x1k3=0;
```

```
x1k4=0;
x1k5=0;
x1k6=0;
y1k1=0;
y1k2=0;
y1k3=0;
y1k4=0;
y1k5=0;
y1k6=0;

if sensor==1
%mainboom - 75 sek
p1 =    2.029e-13 ;
        p2 =   -7.369e-11   ;
        p3 =    1.137e-08   ;
        p4 =   -9.716e-07   ;
        p5 =    5.014e-05 ;
        p6 =      -0.0016   ;
        p7 =      0.03095   ;
        p8 =      -0.3362   ;
        p9 =       0.7913   ;
        p10 =        60.36   ;
        t_final=75;
elseif sensor==2

%knuckleboom - 60 sek
p1 =   -3.784e-13   ;
        p2 =    1.004e-10   ;
        p3 =   -1.267e-08   ;
        p4 =    9.881e-07   ;
        p5 =   -5.037e-05   ;
        p6 =     0.001663   ;
        p7 =     -0.03416   ;
        p8 =       0.4089   ;
        p9 =       -1.161   ;
        p10 =        21.21 ;
t_final=60;
elseif sensor==3
%slew - 37.7 sek
 p1 =      2.7e-10   ;
        p2 =   -4.892e-08;
        p3 =    3.734e-06 ;
        p4 =   -0.0001556   ;
        p5 =     0.003825 ;
        p6 =      -0.0556   ;
        p7 =       0.4429   ;
        p8 =       -1.425   ;
        p9 =         1.48   ;
        p10 =        3.322;
        t_final=37.7;
end
```

```
out2_1=p10;
out1_prev=p10;
for i=1:steps
    if t(i) >=t_final
        time=t_final;
    else
    time=t(i);
    end


    out1 = p1*time^9 + p2*time^8 + p3*time^7 + p4*time^6 + p5*time^5
       + p6*time^4 + p7*time^3 + p8*time^2 + p9*time + p10;
    out1=out1*1;

der_actual=(out1-out1_prev)/dt;
out1_prev=out1;
x1k=floor(out1/K_floor)*K_floor;

if mode~=3
if order==6
    y1k=num(1)*x1k+num(2)*x1k1+num(3)*x1k2+num(4)*x1k3+num(5)*x1k4+
       num(6)*x1k5+num(7)*x1k6-(den(2)*y1k1+den(3)*y1k2+den(4)*y1k3+
       den(5)*y1k4+den(6)*y1k5+den(7)*y1k6);
elseif order==5
    y1k=num(1)*x1k+num(2)*x1k1+num(3)*x1k2+num(4)*x1k3+num(5)*x1k4+
       num(6)*x1k5-(den(2)*y1k1+den(3)*y1k2+den(4)*y1k3+den(5)*y1k4+
       den(6)*y1k5);
elseif order==4
    y1k=num(1)*x1k+num(2)*x1k1+num(3)*x1k2+num(4)*x1k3+num(5)*x1k4-(
       den(2)*y1k1+den(3)*y1k2+den(4)*y1k3+den(5)*y1k4);
elseif order==3
    y1k=num(1)*x1k+num(2)*x1k1+num(3)*x1k2+num(4)*x1k3-(den(2)*y1k1+
       den(3)*y1k2+den(4)*y1k3);
elseif order==2
    y1k=num(1)*x1k+num(2)*x1k1+num(3)*x1k2-(den(2)*y1k1+den(3)*y1k2);
elseif order==1
    y1k=num(1)*x1k+num(2)*x1k1-(den(2)*y1k1);
end
else
    y1k=x1k;
end

xk=(y1k-y1k1)/dt;
y1k6=y1k5;
y1k5=y1k4;
y1k4=y1k3;
y1k3=y1k2;
y1k2=y1k1;
y1k1=y1k;
```

```
x1k6=x1k5;
x1k5=x1k4;
x1k4=x1k3;
x1k3=x1k2;
x1k2=x1k1;
x1k1=x1k;

if mode~=2
if order==6
yk=num(1)*xk+num(2)*xk1+num(3)*xk2+num(4)*xk3+num(5)*xk4+num(6)*xk5+
    num(7)*xk6-(den(2)*yk1+den(3)*yk2+den(4)*yk3+den(5)*yk4+den(6)*yk5
    +den(7)*yk6);
elseif order==5
yk=num(1)*xk+num(2)*xk1+num(3)*xk2+num(4)*xk3+num(5)*xk4+num(6)*xk5-(
    den(2)*yk1+den(3)*yk2+den(4)*yk3+den(5)*yk4+den(6)*yk5);
elseif order==4
yk=num(1)*xk+num(2)*xk1+num(3)*xk2+num(4)*xk3+num(5)*xk4-(den(2)*yk1+
    den(3)*yk2+den(4)*yk3+den(5)*yk4);
elseif order==3
     yk=num(1)*xk+num(2)*xk1+num(3)*xk2+num(4)*xk3-(den(2)*yk1+den(3)*
         yk2+den(4)*yk3);
elseif order==2
     yk=num(1)*xk+num(2)*xk1+num(3)*xk2-(den(2)*yk1+den(3)*yk2);
elseif order==1
     yk=num(1)*xk+num(2)*xk1-(den(2)*yk1);
end
else
     yk=xk;
end

yk6=yk5;
yk5=yk4;
yk4=yk3;
yk3=yk2;
yk2=yk1;
yk1=yk;
xk6=xk5;
xk5=xk4;
xk4=xk3;
xk3=xk2;
xk2=xk1;
xk1=xk;

y1(i)=x1k;
y2(i)=y1k;
y3(i)=xk;
y4(i)=yk;
y5(i)=der_actual;
error(i)=y4(i)-y5(i);
end
```

```
error_acc=0;
for i=100:steps
    error_acc=error_acc+error(i)^2;
end
s=(error_acc/steps)^0.5;%rms error
```

B.4 Kinematikk på algebraisk form

```
%%%%%%%% inputs %%%%%%%%
%referanse for XYZ, meter
x_ref=20.9;
y_ref=0;
z_ref=-8.4;
%Maaling av alpha og gamma, radianer
alpha_actual=1.04;
gamma_actual=0.05;
beta_actual=0.13;
%steptime
dt=0.001;
%velocity reference
vref_x=0.11;
vref_y=0.32;
vref_z=0.003;
%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%% konstanter %%%%%%%
%Innfestingspunkt av MB på kongen
a_x=0;
a_z=2.558;
%Innfestningspunkt mellom MB og KB, i MB ksitt oordinatsystem
b_x_MB=15;
b_z_MB=0;
%Punkt til krantupp, i KB sitt koordinatsystem
c_x_KB=12.368;
c_z_KB=2.215;
%Innfestingspunkt av loftesylinder mot kongen
l1_x=1.12;
l1_z=0.258;
%Innfestingspunkt av loftesylinder mot MB, i MB sitt koordinatsystem
l2_x_MB=4.4;
l2_z_MB=-0.427;
%Innfestingspunkt av foldingssylinder mot MB, i MB sitt
    koordinatsystem
f1_x_MB=10.21;
f1_z_MB=-0.496;
%Innfestingspunkt av foldingssylinder mot KB, i KB sitt
    koordinatsystem
f2_x_KB=1.789;
f2_z_KB=1.291;
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%% utregninger av vinkler %%%%%%%%%
%Lengden mellom pkt. a og pkt. b
l_b_MB=(b_x_MB^2+b_z_MB^2)^0.5;
%Vinkel mellom pkt. a og pkt. b
alpha_b_MB=atan2(b_z_MB,b_x_MB);
%Lengden mellom pkt. b og pkt. c
l_c_KB=(c_x_KB^2+c_z_KB^2)^0.5;
%Vinkel mellom pkt. b og pkt. c
gamma_c_KB=atan2(c_z_KB,c_x_KB);
%%%%%%%%%%%%%%%%%%

%%%%%%% utregning av referanse til sylinderlengde og svingvinkel
    %%%%%%%
radius_ref=(x_ref^2+y_ref^2)^0.5;
beta_ref=atan2(y_ref,x_ref);
alpha_ref=acos((l_b_MB^2+(radius_ref-a_x)^2+(z_ref-a_z)^2-l_c_KB^2)
    /(2*l_b_MB*((radius_ref-a_x)^2+(z_ref-a_z)^2)^0.5))+atan2((z_ref-
    a_z),(radius_ref-a_x))-alpha_b_MB;
gamma_ref=acos((l_b_MB^2+l_c_KB^2-(radius_ref-a_x)^2-(z_ref-a_z)^2)
    /(2*l_b_MB*l_c_KB))-gamma_c_KB+alpha_b_MB;
b_x=a_x+l_b_MB*cos(alpha_ref+alpha_b_MB);
b_z=a_z+l_b_MB*sin(alpha_ref+alpha_b_MB);
l2_x=a_x+cos(alpha_ref)*l2_x_MB-sin(alpha_ref)*l2_z_MB;
l2_z=a_z+sin(alpha_ref)*l2_x_MB+cos(alpha_ref)*l2_z_MB;
f1_x=a_x+cos(alpha_ref)*f1_x_MB-sin(alpha_ref)*f1_z_MB;
f1_z=a_z+sin(alpha_ref)*f1_x_MB+cos(alpha_ref)*f1_z_MB;
f2_x=b_x+f2_x_KB*(sin(alpha_ref)*sin(gamma_ref)-cos(alpha_ref)*cos(
    gamma_ref))+f2_z_KB*(cos(alpha_ref)*sin(gamma_ref)+cos(gamma_ref)*
    sin(alpha_ref));
f2_z=b_z-f2_x_KB*(cos(alpha_ref)*sin(gamma_ref)+cos(gamma_ref)*sin(
    alpha_ref))+f2_z_KB*(sin(alpha_ref)*sin(gamma_ref)-cos(alpha_ref)*
    cos(gamma_ref));
sxl_x=l2_x-l1_x;
sxl_z=l2_z-l1_z;
xl_ref=(sxl_x^2+sxl_z^2)^0.5;
sxf_x=f2_x-f1_x;
sxf_z=f2_z-f1_z;
xf_ref=(sxf_x^2+sxf_z^2)^0.5;
%%%%%%%%%%%%%%%%%%%%

%%%%%%% omregning fra målt vinkel til sylinderlengde %%%%%%%%%%%%%%
b_x_actual=a_x+l_b_MB*cos(alpha_actual+alpha_b_MB);
b_z_actual=a_z+l_b_MB*sin(alpha_actual+alpha_b_MB);
l2_x_actual=a_x+cos(alpha_actual)*l2_x_MB-sin(alpha_actual)*l2_z_MB;
l2_z_actual=a_z+sin(alpha_actual)*l2_x_MB+cos(alpha_actual)*l2_z_MB;
f1_x_actual=a_x+cos(alpha_actual)*f1_x_MB-sin(alpha_actual)*f1_z_MB;
f1_z_actual=a_z+sin(alpha_actual)*f1_x_MB+cos(alpha_actual)*f1_z_MB;
f2_x_actual=b_x_actual+f2_x_KB*(sin(alpha_actual)*sin(gamma_actual)-
    cos(alpha_actual)*cos(gamma_actual))+f2_z_KB*(cos(alpha_actual)*
    sin(gamma_actual)+cos(gamma_actual)*sin(alpha_actual));
f2_z_actual=b_z_actual-f2_x_KB*(cos(alpha_actual)*sin(gamma_actual)+
```

```
    cos(gamma_actual)*sin(alpha_actual))+f2_z_KB*(sin(alpha_actual)*
    sin(gamma_actual)-cos(alpha_actual)*cos(gamma_actual));
sxl_x_actual=l2_x_actual-l1_x;
sxl_z_actual=l2_z_actual-l1_z;
xl_actual=(sxl_x_actual^2+sxl_z_actual^2)^0.5;
sxf_x_actual=f2_x_actual-f1_x_actual;
sxf_z_actual=f2_z_actual-f1_z_actual;
xf_actual=(sxf_x_actual^2+sxf_z_actual^2)^0.5;
%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% omregning fra vref til sylinderhastighet %%%%%%%%%%
phi_l=atan2((l2_z_actual-l1_z),(l2_x_actual-l1_x));
r_l=((l1_x-a_x)*tan(phi_l)-(l1_z-a_z))/(((tan(phi_l))^2+1)^0.5);
phi_f=atan2((f2_z_actual-f1_z_actual),(f2_x_actual-f1_x_actual));
r_f=((f1_x_actual-b_x_actual)*tan(phi_f)-(f1_z_actual-b_z_actual))
    /(((tan(phi_f))^2+1)^0.5);
radius_actual=a_x+b_x_MB*cos(alpha_actual)-b_z_MB*sin(alpha_actual)-
    c_x_KB*cos(alpha_actual+gamma_actual)+c_z_KB*sin(alpha_actual+
    gamma_actual);
xi_a=-b_z_MB*cos(alpha_actual)-b_x_MB*sin(alpha_actual);
xi_b=-b_z_MB*sin(alpha_actual)+b_x_MB*cos(alpha_actual);
xi_c=c_z_KB*cos(alpha_actual+gamma_actual)+c_x_KB*sin(alpha_actual+
    gamma_actual);
xi_d=c_z_KB*sin(alpha_actual+gamma_actual)-c_x_KB*cos(alpha_actual+
    gamma_actual);
v_x_MB=vref_x*cos(beta_actual)+vref_y*sin(beta_actual);
v_z_MB=vref_z;
j11=xi_a+xi_c;
j12=xi_c;
j21=xi_b+xi_d;
j22=xi_d;
alphaDot_ref=(j22*v_x_MB)/(j11*j22 - j12*j21) - (j12*v_z_MB)/(j11*j22
     - j12*j21);
gammaDot_ref=(j11*v_z_MB)/(j11*j22 - j12*j21) - (j21*v_x_MB)/(j11*j22
     - j12*j21);
xlDot_ref=alphaDot_ref*r_l;
xfDot_ref=gammaDot_ref*r_f;
betaDot_ref=(-vref_x*sin(beta_actual)+vref_y*cos(beta_actual))/
    radius_actual;
%%%%%%%%%%%%%%%%
```

B.5 Banegenerator V2

```
clc
clear
close all

amax=0.03; %tr >=6/wn    fra PhD    w1=18.2; %mechanical freq
vmax1=0.0487;%0.0487
vmax2=0.04;%0.04
vmax3=0.064;%0.064
```

```
dx1=0.3;
dx2=0.1;%vmax2*(dx1+vmax1^2/amax)/vmax1-vmax2^2/amax;% gir T2=T1 ved
    trapes
dx3=0.1;%vmax3*(dx1+vmax1^2/amax)/vmax1-vmax3^2/amax;% gir T3=T1 ved
    trapes
dist1=abs(dx1);
dist2=abs(dx2);
dist3=abs(dx3);
Tr1=vmax1/amax;
Tr2=vmax2/amax;
Tr3=vmax3/amax;
T1=(dist1+Tr1*vmax1)/vmax1;
T2=(dist2+Tr2*vmax2)/vmax2;
T3=(dist3+Tr3*vmax3)/vmax3;
if Tr1 >= 0.5*T1
     T1=((4*dist1)/amax)^0.5;
     Tr1=0.5*T1;
end
if Tr2 >= 0.5*T2
     T2=((4*dist2)/amax)^0.5;
     Tr2=0.5*T2;
end
if Tr3 >= 0.5*T3
     T3=((4*dist3)/amax)^0.5;
     Tr3=0.5*T3;
end
T_new=max([T1 T2 T3]);
vmax1_new=dist1/(T_new-Tr1);
vmax2_new=dist2/(T_new-Tr2);
vmax3_new=dist3/(T_new-Tr3);

dt=0.01;
t=[0:dt:T_new];
n=max(size(t));
x1_prev=0;
x2_prev=0;
x3_prev=0;
for i=1:n
     tid1(i)=t(i);
     if tid1<Tr1
          v1(i)=tid1(i)*vmax1_new/Tr1;
     elseif tid1<T_new-Tr1
          v1(i)=vmax1_new;
     elseif tid1(i)<T_new
          v1(i)=vmax1_new-(tid1(i)-T_new+Tr1)*vmax1_new/Tr1;
     else
          v1(i)=0;
     end
     if dx1<0
          v1(i)=-v1(i);
     end
```

```
    x1(i)=x1_prev+v1(i)*dt;
    x1_prev=x1(i);

end

for i=1:n
    tid2(i)=t(i);
    if tid2<Tr2
        v2(i)=tid2(i)*vmax2_new/Tr2;
    elseif tid2<T_new-Tr2
        v2(i)=vmax2_new;
    elseif tid2(i)<T_new
        v2(i)=vmax2_new-(tid2(i)-T_new+Tr2)*vmax2_new/Tr2;
    else
        v2(i)=0;
    end
    if dx2<0
        v2(i)=-v2(i);
    end
    x2(i)=x2_prev+v2(i)*dt;
    x2_prev=x2(i);
end

for i=1:n
    tid3(i)=t(i);
    if tid3<Tr3
        v3(i)=tid3(i)*vmax3_new/Tr3;
    elseif tid3<T_new-Tr3
        v3(i)=vmax3_new;
    elseif tid3(i)<T_new
        v3(i)=vmax3_new-(tid3(i)-T_new+Tr3)*vmax3_new/Tr3;
    else
        v3(i)=0;
    end
    if dx3<0
        v3(i)=-v3(i);
    end
    x3(i)=x3_prev+v3(i)*dt;
    x3_prev=x3(i);
end

str = sprintf('Finished in %f seconds',T_new);
subplot(3,2,1)
plot(tid1,v1)
hold on
plot([0 T_new],[vmax1 vmax1],'r--');
plot([0 T_new],[-vmax1 -vmax1],'r--');
grid
axis([min(t) max(t) -1.1*vmax1 1.1*vmax1])
title(str);
subplot(3,2,3)
```

```
plot(tid2,v2)
hold on
plot([0 T_new],[vmax2 vmax2],'r--');
plot([0 T_new],[-vmax2 -vmax2],'r--');
grid
axis([min(t) max(t) -1.1*vmax2 1.1*vmax2])
subplot(3,2,5)
plot(tid3,v3)
hold on
plot([0 T_new],[vmax3 vmax3],'r--');
plot([0 T_new],[-vmax3 -vmax3],'r--');
grid
axis([min(t) max(t) -1.1*vmax3 1.1*vmax3])
subplot(3,2,2)
plot(tid3,x1)
hold on
plot([0 T_new],[dx1 dx1],'r--');
plot([0 T_new],[0 0],'r--');
axis([min(t) max(t) min(-0.1*dx1,1.1*dx1) max(-0.1*dx1,1.1*dx1)])
grid
subplot(3,2,4)
plot(tid3,x2)
hold on
plot([0 T_new],[dx2 dx2],'r--');
plot([0 T_new],[0 0],'r--');
axis([min(t) max(t) min(-0.1*dx2,1.1*dx2) max(-0.1*dx2,1.1*dx2)])
grid
subplot(3,2,6)
plot(tid3,x3)
hold on
plot([0 T_new],[dx3 dx3],'r--');
plot([0 T_new],[0 0],'r--');
axis([min(t) max(t) min(-0.1*dx3,1.1*dx3) max(-0.1*dx3,1.1*dx3)])
grid
```

B.6 Kontrollsystemoptimalisering - hovedprogram

```
clc
close all

optionsga=gaoptimset(@ga); %create default settings for optimization
optionsga=gaoptimset(optionsga,'PlotFcns',{@gaplotbestindiv
    @gaplotbestf},'Display','iter','MutationFcn',
    @mutationadaptfeasible);

optionsfmincon=optimset(@fmincon); %create default settings for
    optimization
optionsfmincon=optimset(optionsfmincon,'PlotFcns', {@optimplotx
    @optimplotfval},'Display', 'iter');

optionsfminsearch=optimset(@fminsearch); %create default settings for
```

```
    optimization
optionsfminsearch=optimset(optionsfminsearch,'PlotFcns', {@optimplotx
    @optimplotfval},'Display', 'iter');

optionspatternsearch=psoptimset(@patternsearch); %create default
   settings for optimization
optionspatternsearch=psoptimset(optionspatternsearch,'PlotFcns', {
   @psplotbestx @psplotbestf},'Display', 'iter');

z_sving=[];
p_sving=[-170+82.2i -170-82.2i -4.09+17.7i -4.09-17.7i];
k_sving=753280;%total gain = 0.064

z_lofte=[-399 -292 -31.6];
p_lofte=[-170+82.2i -170-82.2i -4.09+17.7i -4.09-17.7i -440 -297 -208
    -46.4 -8.69];
k_lofte=1.4014e9; %total gain = 0.04

z_folde=[-297 -227 -81.4];
p_folde=[-170+82.2i -170-82.2i -4.09+17.7i -4.09-17.7i -440 -297 -208
    -46.4 -8.69];
k_folde=1.0578e9;% total gain = 0.045

sys_sving_zpk=zpk(z_sving,p_sving,k_sving);
sys_lofte_zpk=zpk(z_lofte,p_lofte,k_lofte);
sys_folde_zpk=zpk(z_folde,p_folde,k_folde);

sys_sving_tf=tf(sys_sving_zpk);
sys_lofte_tf=tf(sys_lofte_zpk);
sys_folde_tf=tf(sys_folde_zpk);

lb=[1e-4 1e-4 1e-4 1e-4 1e-4 1e-4 1e-4 1e-4 1e-4];
ub=[1e3 1e3 1e3 1e3 1e3 1e3 1e3 1e3 1e3];
x0=[2 2 2 1 108 1 109 1 85];
[x_fmincon,fval_fmincon]=fmincon(@zero_pole,x0,[],[],[],[],lb,ub,[],
   optionsfmincon)
[x_fminsearch,fval_fminsearch]=fminsearch(@zero_pole,x0,
   optionsfmincon)
[x_patternsearch,fval_patternsearch]=patternsearch(@zero_pole,x0
   ,[],[],[],[],lb,ub,[],optionspatternsearch)
[x_ga,fval_ga]=ga(@zero_pole,max(size(lb)),[],[],[],[],lb,ub,[],[],
   optionsga)
```

B.7 Kontrollsystemoptimalisering - objective function

```
function s=zero_pole(x)
assignin('base','x1',x(1));
assignin('base','x2',x(2));
assignin('base','x3',x(3));
assignin('base','x4',x(4));
assignin('base','x5',x(5));
```

```
assignin('base','x6',x(6));
assignin('base','x7',x(7));
assignin('base','x8',x(8));
assignin('base','x9',x(9));
sim('opt_model_estimation2',224);
n=max(size(pos_error));
pos1=0;
vel1=0;
pos2=0;
vel2=0;
pos3=0;
vel3=0;
for i=1:n
    pos1=pos1+pos_error(i,1)^2;
end
s1=(pos1/n)^0.5;
for i=1:n
    vel1=vel1+vel_error(i,1)^2;
end
s2=(vel1/n)^0.5;
for i=1:n
    pos2=pos2+pos_error(i,2)^2;
end
s3=(pos2/n)^0.5;
for i=1:n
    vel2=vel2+vel_error(i,2)^2;
end
s4=(vel2/n)^0.5;
for i=1:n
    pos3=pos3+pos_error(i,3)^2;
end
s5=(pos3/n)^0.5;
for i=1:n
    vel3=vel3+vel_error(i,3)^2;
end
s6=(vel3/n)^0.5;

s=s1+s3+s5;
% s=s1+s2+s3+s4+s5+s6;
```

C. Kode skrevet i Simatic Step 7 - SCL

C.1 DB4000 - Pathtable

```
DATA_BLOCK PathTable
//
// Block Comment ...
//
    STRUCT
        Table:ARRAY[1..70,1..3] OF REAL:=
    -5.0, 10.0, -3.5,
    -5.0, 10.0, 0.0,
    -3.0, 10.0, -3.5,
    -3.0, 10.0, 0.0,
    -2.0, 10.0, 0.0,
    0.0, 10.0, 0.0,
    0.0, 10.0, -3.5,
    -2.0, 10.0, -3.5,
    -2.0, 10.0, 0.0,
    0.0, 10.0, 0.0,
    1.0, 10.0, 0.0,
    2.0, 10.0, -3.5,
    3.0, 10.0, 0.0,
    5.0, 10.0, 0.0;

            END_STRUCT
BEGIN
END_DATA_BLOCK
```

C.2 DB4004 - Krangeometri

```
DATA_BLOCK krangeometri
//
// Denne datablokken inneholder geometriske mål spesifikke for den
   gitte kranen.


STRUCT
```

```
 a_x : REAL :=0.0; // Innfestingspunkt av MB på kongen
 a_z : REAL :=2.558; // Innfestingspunkt av MB på kongen


b_x_MB : REAL :=15.0; //Innfestningspunkt mellom MB og KB, i MB sitt
    oordinatsystem
b_z_MB : REAL :=0.0; //Innfestningspunkt mellom MB og KB, i MB sitt
    oordinatsystem


c_x_KB : REAL :=12.368; //Punkt til krantupp, i KB sitt
    koordinatsystem
c_z_KB : REAL :=2.215; //Punkt til krantupp, i KB sitt
    koordinatsystem


l1_x : REAL :=1.12; // Innfestingspunkt av løftesylinder mot kongen
l1_z : REAL :=0.258; // Innfestingspunkt av løftesylinder mot kongen


l2_x_MB : REAL :=4.4; //Innfestingspunkt av løftesylinder mot MB, i
    MB sitt koordinatsystem
l2_z_MB : REAL :=-0.427; //Innfestingspunkt av løftesylinder mot MB,
    i MB sitt koordinatsystem


f1_x_MB : REAL :=10.21; // Innfestingspunkt av foldingssylinder mot
    MB, i MB sitt koordinatsystem
f1_z_MB : REAL :=-0.496; // Innfestingspunkt av foldingssylinder mot
    MB, i MB sitt koordinatsystem


f2_x_KB : REAL :=1.789; // Innfestingspunkt av foldingssylinder mot
    KB, i KB sitt koordinatsystem
f2_z_KB : REAL :=1.291; // Innfestingspunkt av foldingssylinder mot
    KB, i KB sitt koordinatsystem

slewlimit : BOOL :=TRUE; // Har kranen en begrensning for sving
    vinkel grunnet hydraulisk dragchain?
beta_min : REAL:=-3.83; // Minimum sving vinkel grunnet hydraulisk
    dragchain, i radianer.
beta_max : REAL:=3.83; // Maximum sving vinkel grunnet hydraulisk
    dragchain, i radianer.


END_STRUCT

BEGIN
END_DATA_BLOCK
```

C.3 DB4005 - Hydraulikk specs

```
DATA_BLOCK Hydraulikk_specs
//
// Spesifikasjonene til det hydrauliske systemet.
// Her står max flow til ventilene og info om aktuatorene.
// Sving - Løfte - Folde
//
    STRUCT
        Q_p : REAL :=447.0;          //l/min - Maks flow fra pumpen

        Q_s : REAL :=210.0; //    l/min - maks flow inn og ut
        v_m_s : REAL :=180.0; //    cm3/rev - 3*60cm3/rev
        i_s : REAL :=1900.0; //     girratio i svingkrans

        Q_i_l : REAL :=80.0; //    l/min - maks flow inn
        Q_o_l : REAL :=160.0; //   l/min - maks flow ut
        d_i_l : REAL :=200.0; //   mm - rod diameter
        d_o_l : REAL :=280.0; //   mm - bore diameter

        Q_i_f : REAL :=85.0; //    l/min - maks flow inn
        Q_o_f : REAL :=170.0; //   l/min - maks flow ut
        d_i_f : REAL :=180.0; //   mm - rod diameter
        d_o_f : REAL :=250.0; //   mm - bore diameter
    END_STRUCT
BEGIN
END_DATA_BLOCK
```

C.4 FB2000 - Kinematikk

```
FUNCTION_BLOCK Kinematikk

VERSION : '0.1'
NAME    : 'Kin_SCL'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
alpha_actual, gamma_actual, beta_actual, x_ref, y_ref, z_ref, vref_x,
    vref_y, vref_z : REAL;

END_VAR

VAR_OUTPUT
xlDot_ref , xfDot_ref, betaDot_ref, xl_ref, xf_ref, beta_ref,
   xl_actual, xf_actual, x_actual, y_actual, z_actual : REAL;
END_VAR

VAR
 // konstanter */
 //Pi
    pi : REAL;
  y_prime_actual : REAL :=0; //Per definisjon
```

```
 // Innfestingspunkt av MB på kongen */
 a_x : REAL;
 a_z : REAL;

 //Innfestningspunkt mellom MB og KB, i MB sitt oordinatsystem */
b_x_MB : REAL;
b_z_MB : REAL;

//Punkt til krantupp, i KB sitt koordinatsystem */
c_x_KB : REAL;
c_z_KB : REAL;

// Innfestingspunkt av løftesylinder mot kongen */
l1_x : REAL;
l1_z : REAL;

//Innfestingspunkt av løftesylinder mot MB, i MB sitt koordinatsystem
    */
l2_x_MB : REAL;
l2_z_MB : REAL;

// Innfestingspunkt av foldingssylinder mot MB, i MB sitt
   koordinatsystem */
f1_x_MB : REAL;
f1_z_MB : REAL;

// Innfestingspunkt av foldingssylinder mot KB, i KB sitt
   koordinatsystem */
f2_x_KB : REAL;
f2_z_KB : REAL;



END_VAR

VAR_TEMP


// Vinkler som beregnes */
l_b_MB : REAL;
alpha_b_MB : REAL;
l_c_KB : REAL;
gamma_c_KB : REAL;

x_prime_actual, z_prime_actual : REAL;

// Mellomregninger */
radius_ref, alpha_ref, gamma_ref, b_x, b_z, l2_x, l2_z, f1_x, f1_z,
   f2_x, f2_z, sxl_x, sxl_z, sxf_x, sxf_z, b_x_actual : REAL;
 b_z_actual, l2_x_actual, l2_z_actual, f1_x_actual, f1_z_actual,
```

```
    f2_x_actual,f2_z_actual, sxl_x_actual, sxl_z_actual, sxf_x_actual
     : REAL;
 sxf_z_actual, phi_l, r_l, phi_f, r_f, radius_actual, xi_a, xi_b,
    xi_c, xi_d, v_x_MB, v_z_MB, j11, j12, j21, j22, alphaDot_ref,
    gammaDot_ref : REAL;


END_VAR

BEGIN
pi:=DB4002.pi;
a_x:=DB4004.a_x;
a_z:=DB4004.a_z;
b_x_MB:=DB4004.b_x_MB;
b_z_MB:=DB4004.b_z_MB;
c_x_KB:=DB4004.c_x_KB;
c_z_KB:=DB4004.c_z_KB;
l1_x:=DB4004.l1_x;
l1_z:=DB4004.l1_z;
l2_x_MB:=DB4004.l2_x_MB;
l2_z_MB :=DB4004.l2_z_MB;
f1_x_MB:=DB4004.f1_x_MB;
f1_z_MB:=DB4004.f1_z_MB;
f2_x_KB :=DB4004.f2_x_KB;
f2_z_KB :=DB4004.f2_z_KB;


//Utregninger av vinkler */

// Lengden mellom pkt. a og pkt. b */
l_b_MB :=
(b_x_MB**2+b_z_MB**2)**0.5;

// Vinkel mellom pkt. a og pkt. b */
alpha_b_MB :=
ATAN2(Y:=b_z_MB,X:=b_x_MB);

// Lengden mellom pkt. b og pkt. c */
l_c_KB :=
(c_x_KB**2+c_z_KB**2)**0.5;

// Vinkel mellom pkt. b og pkt. c */
gamma_c_KB :=
ATAN2(Y:=c_z_KB,X:=c_x_KB);

// Utregning av referanse til sylinderlengde og svingvinkel */
radius_ref :=
(x_ref**2+y_ref**2)**0.5;

 beta_ref :=
 ATAN2(Y:=y_ref,X:=x_ref);
```

```
 alpha_ref :=
ACOS((l_b_MB**2+(radius_ref-a_x)**2+(z_ref-a_z)**2-l_c_KB**2)/(2*
    l_b_MB*((radius_ref-a_x)**2+(z_ref-a_z)**2)**0.5))+ATAN2(Y:=(z_ref
    -a_z),X:=(radius_ref-a_x))-alpha_b_MB;

gamma_ref := acos((l_b_MB**2+l_c_KB**2-(radius_ref-a_x)**2-(z_ref-a_z
    )**2)/(2*l_b_MB*l_c_KB))-gamma_c_KB+alpha_b_MB;

 b_x :=
a_x+l_b_MB*cos(alpha_ref+alpha_b_MB);

b_z :=
a_z+l_b_MB*sin(alpha_ref+alpha_b_MB);

l2_x :=
a_x+cos(alpha_ref)*l2_x_MB-sin(alpha_ref)*l2_z_MB;

l2_z :=
a_z+sin(alpha_ref)*l2_x_MB+cos(alpha_ref)*l2_z_MB;

f1_x :=
a_x+cos(alpha_ref)*f1_x_MB-sin(alpha_ref)*f1_z_MB;

f1_z :=
a_z+sin(alpha_ref)*f1_x_MB+cos(alpha_ref)*f1_z_MB;

f2_x :=
b_x+f2_x_KB*(sin(alpha_ref)*sin(gamma_ref)-cos(alpha_ref)*cos(
    gamma_ref))+f2_z_KB*(cos(alpha_ref)*sin(gamma_ref)+cos(gamma_ref)*
    sin(alpha_ref));

f2_z :=
b_z-f2_x_KB*(cos(alpha_ref)*sin(gamma_ref)+cos(gamma_ref)*sin(
    alpha_ref))+f2_z_KB*(sin(alpha_ref)*sin(gamma_ref)-cos(alpha_ref)*
    cos(gamma_ref));

sxl_x :=
l2_x-l1_x;

sxl_z :=
l2_z-l1_z;

xl_ref :=
(sxl_x**2+sxl_z**2)**0.5;

sxf_x :=
f2_x-f1_x;

sxf_z :=
f2_z-f1_z;
```

```
xf_ref :=
(sxf_x**2+sxf_z**2)**0.5;

// omregning fra målt vinkel til sylinderlengde */
b_x_actual :=
a_x+l_b_MB*cos(alpha_actual+alpha_b_MB);

b_z_actual :=
a_z+l_b_MB*sin(alpha_actual+alpha_b_MB);

l2_x_actual :=
a_x+cos(alpha_actual)*l2_x_MB-sin(alpha_actual)*l2_z_MB;

l2_z_actual :=
a_z+sin(alpha_actual)*l2_x_MB+cos(alpha_actual)*l2_z_MB;

f1_x_actual :=
a_x+cos(alpha_actual)*f1_x_MB-sin(alpha_actual)*f1_z_MB;

f1_z_actual :=
a_z+sin(alpha_actual)*f1_x_MB+cos(alpha_actual)*f1_z_MB;

f2_x_actual :=
b_x_actual+f2_x_KB*(sin(alpha_actual)*sin(gamma_actual)-cos(
    alpha_actual)*cos(gamma_actual))+f2_z_KB*(cos(alpha_actual)*sin(
    gamma_actual)+cos(gamma_actual)*sin(alpha_actual));

f2_z_actual :=
b_z_actual-f2_x_KB*(cos(alpha_actual)*sin(gamma_actual)+cos(
    gamma_actual)*sin(alpha_actual))+f2_z_KB*(sin(alpha_actual)*sin(
    gamma_actual)-cos(alpha_actual)*cos(gamma_actual));

sxl_x_actual :=
l2_x_actual-l1_x;

sxl_z_actual :=
l2_z_actual-l1_z;

xl_actual :=
(sxl_x_actual**2+sxl_z_actual**2)**0.5;

sxf_x_actual :=
f2_x_actual-f1_x_actual;

sxf_z_actual :=
f2_z_actual-f1_z_actual;

xf_actual :=
(sxf_x_actual**2+sxf_z_actual**2)**0.5;
```

```
// omregning fra vref til sylinderhastighet */
phi_l :=
ATAN2(Y:=(l2_z_actual-l1_z),X:=(l2_x_actual-l1_x)); // MERK: SKRIV
    INN RIKTIG navn på atan2 funksjonen */

r_l :=
((l1_x-a_x)*tan(phi_l)-(l1_z-a_z))/(((tan(phi_l))**2+1)**0.5);

phi_f :=
ATAN2(Y:=(f2_z_actual-f1_z_actual),X:=(f2_x_actual-f1_x_actual));
    // MERK: SKRIV INN RIKTIG navn på atan2 funksjonen */

r_f :=
((f1_x_actual-b_x_actual)*tan(phi_f)-(f1_z_actual-b_z_actual))/(((tan
    (phi_f))**2+1)**0.5);

radius_actual :=
a_x+b_x_MB*cos(alpha_actual)-b_z_MB*sin(alpha_actual)-c_x_KB*cos(
    alpha_actual+gamma_actual)+c_z_KB*sin(alpha_actual+gamma_actual);

xi_a :=
-b_z_MB*cos(alpha_actual)-b_x_MB*sin(alpha_actual);

xi_b :=
-b_z_MB*sin(alpha_actual)+b_x_MB*cos(alpha_actual);

xi_c:=
c_z_KB*cos(alpha_actual+gamma_actual)+c_x_KB*sin(alpha_actual+
    gamma_actual);

xi_d :=
c_z_KB*sin(alpha_actual+gamma_actual)-c_x_KB*cos(alpha_actual+
    gamma_actual);

v_x_MB :=
vref_x*cos(beta_actual)+vref_y*sin(beta_actual);

v_z_MB :=
vref_z;

j11 :=
xi_a+xi_c;

j12 :=
xi_c;

j21 :=
xi_b+xi_d;

j22 :=
xi_d;
```

```
alphaDot_ref :=
(j22*v_x_MB)/(j11*j22 - j12*j21) - (j12*v_z_MB)/(j11*j22 - j12*j21);

gammaDot_ref :=
(j11*v_z_MB)/(j11*j22 - j12*j21) - (j21*v_x_MB)/(j11*j22 - j12*j21);

xlDot_ref :=
alphaDot_ref*r_l;

xfDot_ref :=
gammaDot_ref*r_f;

betaDot_ref :=
(-vref_x*SIN(beta_actual)+vref_y*COS(beta_actual))/radius_actual;

//Regner ut faktisk x,y,z av krantupp

x_prime_actual :=
a_x+l_b_MB*COS(alpha_b_MB+alpha_actual)+l_c_KB*COS(alpha_b_MB+
    alpha_actual+gamma_actual+gamma_c_KB+pi);

x_actual :=
x_prime_actual*COS(beta_actual);   //Rotasjonsmatrise

y_actual :=
SIN(beta_actual)*x_prime_actual;

z_prime_actual:=
a_z+l_b_MB*SIN(alpha_b_MB+alpha_actual)+l_c_KB*SIN(alpha_b_MB+
    alpha_actual+gamma_actual+gamma_c_KB+pi);

z_actual :=
z_prime_actual; //Rotasjonsmatrise

END_FUNCTION_BLOCK
```

C.5 FB2001 - Kaskadekontroller

```
FUNCTION_BLOCK Cascade_Controller

VERSION : '0.1'
NAME    : 'Ctrl_SCL'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
    xf_actual, xfDot_actual, xl_actual, xlDot_actual, beta_actual,
        betaDot_actual:REAL;
    xf_ref, xfDot_ref, xl_ref, xlDot_ref, beta_ref, betaDot_ref:REAL;
    kp_pos_sving, kp_vel_sving, ki_vel_sving, kp_pos_lofte, kp_vel_lofte,
```

```
        ki_vel_lofte , kp_pos_folde , kp_vel_folde , ki_vel_folde :REAL;
    k_anti_sving , k_anti_lofte , k_anti_folde :REAL;
    steptime :REAL;//steptime in seconds
    integral_reset_sving , integral_reset_lofte , integral_reset_folde :
        BOOL;
END_VAR

VAR_OUTPUT
    valve_sving :REAL;
    valve_lofte :REAL;
    valve_folde :REAL;
END_VAR

VAR
    anti_sving :REAL:=0.0;
    anti_lofte :REAL:=0.0;
    anti_folde :REAL:=0.0;
    previous_integral_sving :REAL:=0.0;
    previous_integral_lofte :REAL:=0.0;
    previous_integral_folde :REAL:=0.0;

    outer_error_sving , outer_error_lofte , outer_error_folde ,
        outer_output_sving , outer_output_lofte , outer_output_folde ,
        inner_error_sving , inner_error_lofte , inner_error_folde :REAL;
    inner_p_sving , inner_p_lofte , inner_p_folde , inner_i_sving ,
        inner_i_lofte , inner_i_folde , before_integral_sving ,
        before_integral_lofte , before_integral_folde :REAL;
    inner_integral_sving , inner_integral_lofte , inner_integral_folde ,
        before_saturation_sving , before_saturation_lofte ,
        before_saturation_folde :REAL;
    after_saturation_sving , after_saturation_lofte ,
        after_saturation_folde , before_anti_sving , before_anti_lofte ,
        before_anti_folde , t :REAL;


    sving_error_absement , lofte_error_absement , folde_error_absement
        : REAL;
END_VAR

BEGIN
//outer loop P-regulator
outer_error_sving:=beta_ref-beta_actual;
outer_error_lofte:=xl_ref-xl_actual;
outer_error_folde:=xf_ref-xf_actual;
outer_output_sving:=kp_pos_sving*outer_error_sving;
outer_output_lofte:=kp_pos_lofte*outer_error_lofte;
outer_output_folde:=kp_pos_folde*outer_error_folde;

//inner loop PI-regulator
inner_error_sving:=betaDot_ref-betaDot_actual+outer_output_sving;
inner_error_lofte:=xlDot_ref-xlDot_actual+outer_output_lofte;
```

```
inner_error_folde:=xfDot_ref-xfDot_actual+outer_output_folde;
inner_p_sving:=inner_error_sving*kp_vel_sving;
inner_p_lofte:=inner_error_lofte*kp_vel_lofte;
inner_p_folde:=inner_error_folde*kp_vel_folde;
inner_i_sving:=inner_error_sving*ki_vel_sving;
inner_i_lofte:=inner_error_lofte*ki_vel_lofte;
inner_i_folde:=inner_error_folde*ki_vel_folde;
before_integral_sving:=inner_i_sving+anti_sving;
before_integral_lofte:=inner_i_lofte+anti_lofte;
before_integral_folde:=inner_i_folde+anti_folde;
inner_integral_sving:=previous_integral_sving+before_integral_sving*
   steptime;
inner_integral_lofte:=previous_integral_lofte+before_integral_lofte*
   steptime;
inner_integral_folde:=previous_integral_folde+before_integral_folde*
   steptime;
previous_integral_sving:=inner_integral_sving;
previous_integral_lofte:=inner_integral_lofte;
previous_integral_folde:=inner_integral_folde;

//reset integral in sharp corners
IF integral_reset_sving=TRUE THEN
    inner_integral_sving:=0.0;
END_IF;
IF integral_reset_lofte=TRUE THEN
    inner_integral_lofte:=0.0;
END_IF;
IF integral_reset_folde=TRUE THEN
    inner_integral_folde:=0.0;
END_IF;

before_saturation_sving:=inner_p_sving+inner_integral_sving;
before_saturation_lofte:=inner_p_lofte+inner_integral_lofte;
before_saturation_folde:=inner_p_folde+inner_integral_folde;

//saturation of signal
IF before_saturation_sving > 1.0 THEN
    after_saturation_sving:=1.0;
ELSIF before_saturation_sving < -1.0 THEN
    after_saturation_sving:=-1.0;
ELSE
    after_saturation_sving:=before_saturation_sving;
END_IF;

IF before_saturation_lofte > 1.0 THEN
    after_saturation_lofte:=1.0;
ELSIF before_saturation_lofte < -1.0 THEN
    after_saturation_lofte:=-1.0;
ELSE
    after_saturation_lofte:=before_saturation_lofte;
END_IF;
```

```
IF before_saturation_folde > 1.0 THEN
    after_saturation_folde:=1.0;
ELSIF before_saturation_folde < -1.0 THEN
    after_saturation_folde:=-1.0;
ELSE
    after_saturation_folde:=before_saturation_folde;
END_IF;

//back calculation antiwindup
before_anti_sving:=after_saturation_sving-before_saturation_sving;
before_anti_lofte:=after_saturation_lofte-before_saturation_lofte;
before_anti_folde:=after_saturation_folde-before_saturation_folde;
anti_sving:=before_anti_sving*k_anti_sving;
anti_lofte:=before_anti_lofte*k_anti_lofte;
anti_folde:=before_anti_folde*k_anti_folde;

//outputs
valve_sving:=after_saturation_sving;
valve_lofte:=after_saturation_lofte;
valve_folde:=after_saturation_folde;

t:=t+steptime;
IF (t>20.0) AND (t<250.0) THEN
//Den integrerte av posisjonsfeilen. For assesere ytelsen til
   systemet.
sving_error_absement:=sving_error_absement+ABS(outer_error_sving*
   steptime);
lofte_error_absement:=lofte_error_absement+ABS(outer_error_lofte*
   steptime);
folde_error_absement:=folde_error_absement+ABS(outer_error_folde*
   steptime);
END_IF;
END_FUNCTION_BLOCK
```

C.6 FB2002 - OB35 counter

```
FUNCTION_BLOCK OB35_COUNTER

VERSION : '0.1'
NAME    : 'CNT35'
FAMILY  : Crane
AUTHOR  : KJTH

VAR
count_OB35:REAL:=0;
END_VAR

VAR_OUTPUT
    start_path : BOOL:=FALSE;
    END_VAR
BEGIN
```

```
count_OB35:=count_OB35+1;
start_path:= TRUE;
END_FUNCTION_BLOCK
```

C.7 FB2003 - OB1 counter

```
FUNCTION_BLOCK OB1_COUNTER

VERSION : '0.1'
NAME    : 'CNT1'
FAMILY  : Crane
AUTHOR  : KJTH

VAR
count_OB1:REAL:=0;
END_VAR
BEGIN
count_OB1:=count_OB1+1;
END_FUNCTION_BLOCK
```

C.8 FB2004 - Valve feedback

```
FUNCTION_BLOCK Valve_feedback_function

VERSION : '0.1'
NAME    : 'VF_SCL'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
valve_sving, valve_lofte, valve_folde,alpha_actual, gamma_actual,
    alpha_dot, beta_dot, gamma_dot   : REAL;
use_valve_feedback : BOOL;
END_VAR

VAR_OUTPUT
xlDot_actual, xfDot_actual, betaDot_actual : REAL;
END_VAR

VAR
     al,phil,af,phif, alpha_b_MB, gamma_c_KB : REAL;
     pi : REAL;

a_x, a_z, b_x_MB, b_z_MB, c_x_KB, c_z_KB, l1_x, l1_z, l2_x_MB,
    l2_z_MB : REAL;
f1_x_MB, f1_z_MB, f2_x_KB, f2_z_KB : REAL;
b_x_actual, b_z_actual, l2_x_actual, l2_z_actual, f1_x_actual,
    f1_z_actual, l_b_MB : REAL;
f2_x_actual, f2_z_actual, phi_l, r_l, phi_f, r_f : REAL;
END_VAR
```

```
BEGIN
pi:=DB4002.pi;

IF use_valve_feedback=FALSE THEN

a_x:=DB4004.a_x;
a_z:=DB4004.a_z;
b_x_MB:=DB4004.b_x_MB;
b_z_MB:=DB4004.b_z_MB;
c_x_KB:=DB4004.c_x_KB;
c_z_KB:=DB4004.c_z_KB;
l1_x:=DB4004.l1_x;
l1_z:=DB4004.l1_z;
l2_x_MB:=DB4004.l2_x_MB;
l2_z_MB :=DB4004.l2_z_MB;
f1_x_MB:=DB4004.f1_x_MB;
f1_z_MB:=DB4004.f1_z_MB;
f2_x_KB :=DB4004.f2_x_KB;
f2_z_KB :=DB4004.f2_z_KB;

alpha_b_MB :=
ATAN2(Y:=b_z_MB,X:=b_x_MB);

gamma_c_KB :=
ATAN2(Y:=c_z_KB,X:=c_x_KB);

l_b_MB :=
(b_x_MB**2+b_z_MB**2)**0.5;

b_x_actual :=
a_x+l_b_MB*cos(alpha_actual+alpha_b_MB);

b_z_actual :=
a_z+l_b_MB*sin(alpha_actual+alpha_b_MB);

l2_x_actual :=
a_x+cos(alpha_actual)*l2_x_MB-sin(alpha_actual)*l2_z_MB;

l2_z_actual :=
a_z+sin(alpha_actual)*l2_x_MB+cos(alpha_actual)*l2_z_MB;

f1_x_actual :=
a_x+cos(alpha_actual)*f1_x_MB-sin(alpha_actual)*f1_z_MB;

f1_z_actual :=
a_z+sin(alpha_actual)*f1_x_MB+cos(alpha_actual)*f1_z_MB;

f2_x_actual :=
b_x_actual+f2_x_KB*(sin(alpha_actual)*sin(gamma_actual)-cos(
    alpha_actual)*cos(gamma_actual))+f2_z_KB*(cos(alpha_actual)*sin(
```

```
    gamma_actual)+cos(gamma_actual)*sin(alpha_actual));

f2_z_actual :=
b_z_actual-f2_x_KB*(cos(alpha_actual)*sin(gamma_actual)+cos(
    gamma_actual)*sin(alpha_actual))+f2_z_KB*(sin(alpha_actual)*sin(
    gamma_actual)-cos(alpha_actual)*cos(gamma_actual));

// omregning fra vref til sylinderhastighet */
 phi_l :=
 ATAN2(Y:=(l2_z_actual-l1_z),X:=(l2_x_actual-l1_x));

 r_l :=
 ((l1_x-a_x)*tan(phi_l)-(l1_z-a_z))/(((tan(phi_l))**2+1)**0.5);

 phi_f :=
 ATAN2(Y:=(f2_z_actual-f1_z_actual),X:=(f2_x_actual-f1_x_actual));

 r_f :=
((f1_x_actual-b_x_actual)*TAN(phi_f)-(f1_z_actual-b_z_actual))/(((TAN
    (phi_f))**2+1)**0.5);


xlDot_actual:=r_l*alpha_dot;
xfDot_actual:=r_f*gamma_dot;
betaDot_actual:=beta_dot;
ELSE


al:=0.25*pi*DB4005.d_o_l**2;
phil:=(DB4005.d_o_l**2-DB4005.d_i_l**2)/DB4005.d_o_l**2;
af:=0.25*pi*DB4005.d_o_f**2;
phif:=(DB4005.d_o_f**2-DB4005.d_i_f**2)/DB4005.d_o_f**2;

IF valve_lofte < 0 THEN
     xlDot_actual:=valve_lofte*DB4005.Q_i_l/(al*phil*0.06);
ELSE
     xlDot_actual:=valve_lofte*DB4005.Q_o_l/(al*0.06);
END_IF;

IF valve_folde < 0 THEN
     xfDot_actual:=valve_folde*DB4005.Q_i_f/(af*phif*0.06);
ELSE
     xfDot_actual:=valve_folde*DB4005.Q_o_f/(af*0.06);
END_IF;

betaDot_actual:=valve_sving*DB4005.Q_s*104.7/(DB4005.v_m_s*DB4005.i_s
    ); //Rad/s

END_IF;
END_FUNCTION_BLOCK
```

C.9 FB2005 - Tablegenerator

```
FUNCTION_BLOCK TableGenerator

VERSION : '0.1'
NAME    : 'Table'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
max_speed: REAL;
tablerows : INT;
END_VAR

VAR_OUTPUT
path_on : BOOL;
END_VAR

VAR

  i: INT :=0;
  vect, unitvect: ARRAY[1..20,1..3]OF REAL;
  length, tid : ARRAY[1..20,1..1] OF REAL;

END_VAR


BEGIN

//Disse vektorene skal brukes i OB35, pathgenerator og må lagres i en
    global datablokk DB4003.Table.
//Forklaring per Kolonne
//1-3 er XYZ posisjonsvektor
//4-6 er XYZ enhetsvektor
//7 er lengden mellom hvert punkt
//8 er tiden mellom hver punkt

  FOR i:= 1 TO tablerows DO

    IF i=1 THEN

        DB4003.Table[i,1]:=PathTable.Table[i,1]-DB4002.x_actual;
        DB4003.Table[i,2]:=PathTable.Table[i,2]-DB4002.y_actual;
        DB4003.Table[i,3]:=PathTable.Table[i,3]-DB4002.z_actual;
    ELSE
        DB4003.Table[i,1]:=PathTable.Table[i,1]-PathTable.Table[i
            -1,1];
        DB4003.Table[i,2]:=PathTable.Table[i,2]-PathTable.Table[i
            -1,2];
        DB4003.Table[i,3]:=PathTable.Table[i,3]-PathTable.Table[i
            -1,3];
```

```
    END_IF;
END_FOR;


//Lengden og tiden
FOR i:= 1 TO tablerows DO

    DB4003.Table[i,7]:= (DB4003.Table[i,1]**2+DB4003.Table[i,2]**2+
        DB4003.Table[i,3]**2)**0.5;
    DB4003.Table[i,8] := DB4003.Table[i,7]/max_speed;

END_FOR;

//Enhetsvektor

FOR i:= 1 TO tablerows DO

    DB4003.Table[i,4]:= DB4003.Table[i,1]/DB4003.Table[i,7];
    DB4003.Table[i,5]:= DB4003.Table[i,2]/DB4003.Table[i,7];
    DB4003.Table[i,6]:= DB4003.Table[i,3]/DB4003.Table[i,7];
END_FOR;

path_on:=TRUE;
END_FUNCTION_BLOCK
```

C.10 FB2006 - Tablerows

```
FUNCTION_BLOCK Tablerows

VAR
   i,tablerows1, tablerows2, tablerows3, total: INT;
END_VAR

VAR_OUTPUT
    out : REAL;
    tablerows: INT;
END_VAR


BEGIN

    i:=1;
    WHILE i <100 DO

     IF PathTable.Table[i,1]=0 THEN
        tablerows1:=0;
    ELSE
        tablerows1:=1;
     END_IF;

     IF PathTable.Table[i,2] =0 THEN
```

```
            tablerows2:=0;
        ELSE
            tablerows2:=1;
     END_IF;

     IF PathTable.Table[i,3] =0 THEN
        tablerows3:=0;
    ELSE
        tablerows3:=1;
    END_IF   ;

    total:=
    ABS(tablerows1)+ABS(tablerows2)+ABS(tablerows3);

    IF total=0 THEN
        tablerows:=i-1;
        i:=100;
        ELSE
            i:=i+1;
     END_IF;
END_WHILE;



END_FUNCTION_BLOCK
```

C.11 FB2007 - Pathgenerator V1

```
FUNCTION_BLOCK Pathgenerator_v1

VERSION : '0.1'
NAME    : 'Pathgen'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
tablerows: INT;
ramptime, steptime, max_speed, max_avvik : REAL;
reset : BOOL:=FALSE;
start : BOOL:=FALSE;
END_VAR

VAR_OUTPUT
    x_ref, y_ref, z_ref, vref_x, vref_y, vref_z: REAL;
    paused: BOOL:=FALSE;
END_VAR


VAR
t_seg, mintime, a, vd, va : REAL;
t : REAL:=0.0;
```

```
i, h : INT;
teller : INT:=0;
tacc : ARRAY[1..20,1..1] OF REAL;
finish : BOOL:=FALSE;
END_VAR

BEGIN

IF reset=TRUE THEN
    t:=0.0;
END_IF;

IF start=TRUE THEN
finish:=FALSE;


//Akkumulert tid
    FOR i:= 1 TO tablerows DO
        IF i=1 THEN
        tacc[i,1]:=DB4003.Table[i,8];
    ELSE
        tacc[i,1]:=DB4003.Table[i,8]+tacc[i-1,1];
        END_IF;
    END_FOR;




// h : REAL - definere en variabel som holder verdien av kolonnen man
    ønsker å være i til en hver tid
h:=1;

FOR i:=1 TO tablerows DO
    IF t>tacc[i,1] THEN
        h:=i+1;
    END_IF;
END_FOR;



//Tidsholder for hver kollone. Tiden for hvert segment.

FOR i:=1 TO tablerows DO
    IF i=1 THEN
        t_seg:=t;
    ELSE
        t_seg:=t-tacc[h-1,1];
    END_IF;
END_FOR;
```

```
   //Kalkulering av hastighet til en hver tid

 FOR i:=1 TO tablerows BY 1 DO
IF i=1 THEN
mintime:=DB4003.Table[i,8];
ELSE
IF DB4003.Table[i,8] < mintime THEN
mintime:=DB4003.Table[i,8];
END_IF;
END_IF;
END_FOR;

IF ramptime > mintime THEN
    ramptime:=mintime;
END_IF;

a:= max_speed/ramptime;

//Lager hastighetsprofilen som skal brukes som referanse


IF t_seg < ramptime THEN
    va:=a*t_seg;
    vd:=max_speed-va;
//Forklaring av DB4003.Table per kolonne Kolonne
//1-3 er XYZ posisjonsvektor
//4-6 er XYZ enhetsvektor
//7 er lengden mellom hvert punkt
//8 er tiden mellom hver punkt

    IF h> tablerows THEN
        vref_x:=vd*DB4003.Table[h-1,4];
        vref_y:=vd*DB4003.Table[h-1,5];
        vref_z:=vd*DB4003.Table[h-1,6];
    ELSIF  h>1 THEN
        vref_x:=va*DB4003.Table[h,4]+vd*DB4003.Table[h-1,4];
        vref_y:=va*DB4003.Table[h,5]+vd*DB4003.Table[h-1,5];
        vref_z:=va*DB4003.Table[h,6]+vd*DB4003.Table[h-1,6];
    ELSE
        vref_x:=va*DB4003.Table[h,4];
        vref_y:=va*DB4003.Table[h,5];
        vref_z:=va*DB4003.Table[h,6];
    END_IF;
ELSE
    IF h>tablerows THEN
        vref_x:=0.0;
        vref_y:=0.0;
        vref_z:=0.0;
    ELSE
        vref_x:=max_speed*DB4003.Table[h,4];
```

```
            vref_y:=max_speed*DB4003.Table[h,5];
            vref_z:=max_speed*DB4003.Table[h,6];
END_IF;
END_IF;

IF t_seg < steptime THEN
     x_ref:=DB4002.x_actual+vref_x*steptime;
     y_ref:=DB4002.y_actual+vref_y*steptime;
     z_ref:=DB4002.z_actual+vref_z*steptime;
     t:=t+steptime;
     teller:=teller+1;
END_IF;

      IF (tacc[h,1]-t) < steptime THEN //Sjekk om referansen er nær
          ønsket punkt h
          //Sjekk om det er stort avvik på ref og kran
           IF ((DB4002.x_actual-x_ref)**2+(DB4002.y_actual-y_ref)**2+(
              DB4002.z_actual-z_ref)**2)**0.5 < max_avvik THEN
              t:=t+steptime;
              x_ref:=x_ref+vref_x*steptime;
              y_ref:=y_ref+vref_y*steptime;
              z_ref:=z_ref+vref_z*steptime;
          END_IF;
     ELSE
          t:=t+steptime;
          x_ref:=x_ref+vref_x*steptime;
          y_ref:=y_ref+vref_y*steptime;
          z_ref:=z_ref+vref_z*steptime;
     END_IF;
END_IF;


END_FUNCTION_BLOCK
```

C.12 FB2008 - Pathgenerator V2

```
FUNCTION_BLOCK Pathgenerator_v2

VERSION : '0.2'
NAME    : 'Pathgen'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
xl_ref, xf_ref, beta_ref, xl_act, xf_act, beta_act, amax, steptime :
   REAL;
start, reset : REAL;
tablerows : INT;

END_VAR
```

```
VAR_OUTPUT
    xl_setp, xf_setp, beta_setp, xl_dot_setp, xf_dot_setp,
       beta_dot_setp: REAL;
    i : INT;
    int_reset : BOOL;
END_VAR


VAR
int_reset_counter : INT:=0;
t : REAL:=0.0;
pi : REAL;
slewlimit : BOOL;
beta_min, beta_max : REAL;
first_time: BOOL:=TRUE;
finished : BOOL:=FALSE;
vmax_s, vmax_l, vmax_f : REAL;
Q_s, Q_l, Q_f : REAL;
dx_l, dx_s, dx_f, K1, K2, K, Tr_s_1, Tr_l_1, Tr_f_1, Tr_s_2, Tr_l_2,
    Tr_f_2, al, af, phil,phif : REAL;
dist_s, dist_l, dist_f, T_s, T_l, T_f, T_max_1, T_max_2 : REAL;
vmax_s_1, vmax_l_1, vmax_f_1,vmax_s_2, vmax_l_2, vmax_f_2, xs_prev,
    xl_prev, xf_prev : REAL;
END_VAR

BEGIN

slewlimit:=DB4004.slewlimit;
beta_min:=DB4004.beta_min;
beta_max:=DB4004.beta_max;
pi:=DB4002.pi;
int_reset:=FALSE;

IF reset>0.0 THEN
     t:=0.0;
     i:=1;
     first_time:=TRUE;
     finished:=FALSE;
END_IF;

IF start>0.0 THEN

IF first_time=TRUE THEN  //Denne delen kjøres bare en gang mellom to
    punkter.
     dx_l:=xl_ref-xl_act;
     dx_f:=xf_ref-xf_act;
     dx_s:=beta_ref-beta_act;

     WHILE dx_s > 2*pi DO
          dx_s:=dx_s-2*pi;
     END_WHILE;
```

```
    WHILE dx_s <-2*pi DO
        dx_s:=dx_s+2*pi;
    END_WHILE;

    IF dx_s > pi THEN
        dx_s:=dx_s-2*pi;
    ELSIF dx_s <-pi THEN
        dx_s:=dx_s+2*pi;
    END_IF;

IF slewlimit=TRUE THEN
    IF (dx_s+beta_act) > beta_max THEN
        dx_s:=dx_s-2*pi;
    ELSIF (dx_s+beta_act) <beta_min THEN
        dx_s:=dx_s+2*pi;
    END_IF;
END_IF;


al:=0.25*pi*DB4005.d_o_l**2;
phil:=(DB4005.d_o_l**2-DB4005.d_i_l**2)/DB4005.d_o_l**2;
af:=0.25*pi*DB4005.d_o_f**2;
phif:=(DB4005.d_o_f**2-DB4005.d_i_f**2)/DB4005.d_o_f**2;



K1:=0.75; //Vi ønsker å ikke bruke mer enn 75% av maks flow til
    ventilene. Dette vil alltid holde summen av alle flowene under det
    pumpen kan levere.


IF dx_l < 0 THEN
    vmax_l:=K1*DB4005.Q_i_l/(al*phil*0.06);
ELSE
    vmax_l:=K1*DB4005.Q_o_l/(al*0.06);
END_IF;

IF dx_f < 0 THEN
    vmax_f:=K1*DB4005.Q_i_f/(af*phif*0.06);
ELSE
    vmax_f:=K1*DB4005.Q_o_f/(af*0.06);
END_IF;

vmax_s:=K1*DB4005.Q_s*104.7/(DB4005.v_m_s*DB4005.i_s);

Tr_s_1:=vmax_s/amax;
Tr_l_1:=vmax_l/amax;
Tr_f_1:=vmax_f/amax;

dist_s:=ABS(dx_s);
```

```
dist_l:=ABS(dx_l);
dist_f:=ABS(dx_f);

T_s:=(dist_s+Tr_s_1*vmax_s)/vmax_s;
T_l:=(dist_l+Tr_l_1*vmax_l)/vmax_l;
T_f:=(dist_f+Tr_f_1*vmax_f)/vmax_f;

IF Tr_s_1 >= 0.5*T_s THEN
    T_s:=((4*dist_s)/amax)**0.5;
    Tr_s_1:=0.5*T_s;
END_IF;

IF Tr_l_1 >= 0.5*T_l THEN
    T_l:=((4*dist_l)/amax)**0.5;
    Tr_l_1:=0.5*T_l;
END_IF;

IF Tr_f_1 >= 0.5*T_f THEN
    T_f:=((4*dist_f)/amax)**0.5;
    Tr_f_1:=0.5*T_f;
END_IF;

T_max_1:= MAX (IN1:=T_s, IN2:=T_l, IN3:=T_f);
vmax_s_1:=dist_s/(T_max_1-Tr_s_1);
vmax_l_1:=dist_l/(T_max_1-Tr_l_1);
vmax_f_1:=dist_f/(T_max_1-Tr_f_1);



Q_s:=vmax_s_1*DB4005.v_m_s*DB4005.i_s/104.7; //    l/min

IF dx_l< 0 THEN
    Q_l:=vmax_l_1*al*phil*0.06;
ELSE
    Q_l:=vmax_l_1*al*0.06;
END_IF;

IF dx_l< 0 THEN
    Q_f:=vmax_f_1*af*phif*0.06;
ELSE
    Q_f:=vmax_f_1*af*0.06;
END_IF;

//K2 er enda en sikkerhetsfaktor som sikrer at man holder seg under
   maks pumpeflow.
K2:=DB4005.Q_p/(Q_s+Q_l+Q_f);
IF K2 > 1 THEN
    K2:=1.0;
END_IF;
T_max_2:=T_max_1/K2;
```

```
Tr_s_2:=Tr_s_1/K2;
Tr_l_2:=Tr_l_1/K2;
Tr_f_2:=Tr_f_1/K2;

vmax_s_2:=dist_s/(T_max_2-Tr_s_2);
vmax_l_2:=dist_l/(T_max_2-Tr_l_2);
vmax_f_2:=dist_f/(T_max_2-Tr_f_2);
xs_prev:=beta_act;
xl_prev:=xl_act;
xf_prev:=xf_act;
first_time:=FALSE;
END_IF;



IF t< Tr_s_2 THEN
    beta_dot_setp:=t*vmax_s_2/Tr_s_2;
ELSIF t<T_max_2-Tr_s_2 THEN
  beta_dot_setp:=vmax_s_2;
ELSIF t< T_max_2 THEN
     beta_dot_setp:=vmax_s_2-(t-T_max_2+Tr_s_2)*vmax_s_2/Tr_s_2;
ELSE
     beta_dot_setp:=0.0;
END_IF;
IF dx_s < 0.0 THEN
     beta_dot_setp:=-beta_dot_setp;
END_IF;
beta_setp:=xs_prev+beta_dot_setp*steptime;
xs_prev:=beta_setp;

IF t< Tr_l_2 THEN
    xl_dot_setp:=t*vmax_l_2/Tr_l_2;
ELSIF t<T_max_2-Tr_l_2 THEN
  xl_dot_setp:=vmax_l_2;
ELSIF t< T_max_2 THEN
     xl_dot_setp:=vmax_l_2-(t-T_max_2+Tr_l_2)*vmax_l_2/Tr_l_2;
ELSE
     xl_dot_setp:=0.0;
END_IF;
IF dx_l < 0.0 THEN
     xl_dot_setp:=-xl_dot_setp;
END_IF;
xl_setp:=xl_prev+xl_dot_setp*steptime;
xl_prev:=xl_setp;

IF t< Tr_f_2 THEN
    xf_dot_setp:=t*vmax_f_2/Tr_f_2;
ELSIF t<T_max_2-Tr_f_2 THEN
  xf_dot_setp:=vmax_f_2;
ELSIF t< T_max_2 THEN
     xf_dot_setp:=vmax_f_2-(t-T_max_2+Tr_f_2)*vmax_f_2/Tr_f_2;
```

```
ELSE
     xf_dot_setp:=0.0;
END_IF;
IF dx_f < 0.0 THEN
     xf_dot_setp:=-xf_dot_setp;
END_IF;
xf_setp:=xf_prev+xf_dot_setp*steptime;
xf_prev:=xf_setp;

t:=t+steptime;
IF t> T_max_2 THEN
     t:=0.0;
     i:=i+1;
     int_reset:=TRUE;
     int_reset_counter:=int_reset_counter+1;
     IF i > tablerows THEN
          i:=tablerows;
          finished:=TRUE;
     END_IF;
     first_time:=TRUE;
END_IF;

IF finished=TRUE THEN
          beta_dot_setp:=0.0;
          xl_dot_setp:=0.0;
          xf_dot_setp:=0.0;
END_IF;
ELSE
          beta_dot_setp:=0.0;
          xl_dot_setp:=0.0;
          xf_dot_setp:=0.0;
END_IF;


END_FUNCTION_BLOCK
```

C.13 FB2009 - Derivering og filtrering

```
FUNCTION_BLOCK der_og_filter

VAR_INPUT
     alpha, beta,gamma, steptime, omega_alpha, omega_beta, omega_gamma
          : REAL;
END_VAR

VAR_OUTPUT
     alpha_dot, beta_dot,gamma_dot : REAL;
END_VAR
```

```
VAR

c, a1, a2, b0, b1, b2 : REAL;
ac0, ac1, ac2, bc0, bc1, bc2, K : REAL;
freqs, der :ARRAY[1..3] OF REAL;
xk, yk : ARRAY[1..3,1..3] OF REAL;
i : INT;
gain, current, prev: REAL;

END_VAR


BEGIN
freqs[1]:=omega_alpha;
freqs[2]:=omega_beta;
freqs[3]:=omega_gamma;


xk[1,3]:=xk[1,2];
xk[1,2]:=xk[1,1];
xk[1,1]:=alpha;

xk[2,3]:=xk[2,2];
xk[2,2]:=xk[2,1];
xk[2,1]:=beta;

xk[3,3]:=xk[3,2];
xk[3,2]:=xk[3,1];
xk[3,1]:=gamma;


    FOR i:= 1 TO 3 DO
        bc0:=0.0;
        bc1:=0.0;
        bc2:=freqs[i]**2;
        ac0:=1.0;
        ac1:=2**0.5*freqs[i];
        ac2:=bc2;
        k:=2.0/steptime;
        b0:=(bc0*K**2+bc1*K+bc2)/(ac0*K**2+ac1*K+ac2);
        b1:=(2.0*bc2-2.0*bc0*K**2)/(ac0*K**2+ac1*K+ac2);
        b2:=(bc0*K**2-bc1*K+bc2)/(ac0*K**2+ac1*K+ac2);
        a1:=(2.0*ac2-2.0*ac0*K**2)/(ac0*K**2+ac1*K+ac2);
        a2:=(ac0*K**2-ac1*K+ac2)/(ac0*K**2+ac1*K+ac2);

 //c:=freqs[i]*(steptime/2);
 //a1:=(2*(c**2)-2)/((c**2)+(2**0.5)*c+1);
 //a2:=((c**2)-(2**0.5)*c+1)/((c**2)+(2**0.5)*c+1);
 //b0:=(c**2)/((c**2)+(2**0.5)*c+1);
 //b1:=2*b0;
```

```
  //b2:=b0;

  gain:=1;//(b0+b1+b2)/(1+a1+a2);

  yk[i,1]:=xk[i,1]*b0+xk[i,2]*b1+xk[i,3]*b2-yk[i,2]*a1-yk[i,3]*a2;

  //Normaliserer pga for lav presisjon i REAL 32 bits float.
  current:=yk[i,1]/gain;
  prev:=yk[i,2]/gain;

  der[i]:=(current-prev)/steptime;

  yk[i,3]:=yk[i,2];
  yk[i,2]:=yk[i,1];
END_FOR;

alpha_dot:=der[1];
beta_dot:=der[2];
gamma_dot:=der[3];



END_FUNCTION_BLOCK
```

C.14 FB2010 - Pathgenerator V2 - random

```
FUNCTION_BLOCK Pathgenerator_v2_random

VERSION : '0.3'
NAME    : 'Pathgen'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
xl_act, xf_act, beta_act, amax, steptime : REAL;
start, reset : REAL;
tablerows : INT;

END_VAR

VAR_OUTPUT
    xl_setp, xf_setp, beta_setp, xl_dot_setp, xf_dot_setp,
        beta_dot_setp: REAL;
    i : INT;
    int_reset : BOOL;
END_VAR


VAR

//Brukt i random bane
```

```
xl_ref, xf_ref, beta_ref : REAL;
xl_ref_temp, xf_ref_temp, beta_ref_temp : DINT;
beta_min_random : DINT:=0;
beta_max_random : DINT:=6280;
xl_min : DINT:=3800;
xl_max : DINT:=6100;
xf_min : DINT:=3800;
xf_max : DINT:=6600;
ab,alofte,afolde,cb,cl,cf,mb,ml,mf : DINT;
beta_prev : DINT:=0;
lofte_prev : DINT:=0;
folde_prev : DINT:=0;

//Vanlig bane gen
int_reset_counter : INT:=0;
t : REAL:=0.0;
pi : REAL;
slewlimit : BOOL;
beta_min, beta_max : REAL;
first_time: BOOL:=TRUE;
finished : BOOL:=FALSE;
vmax_s, vmax_l, vmax_f : REAL;
Q_s, Q_l, Q_f : REAL;
dx_l, dx_s, dx_f, K1, K2, K, Tr_s_1, Tr_l_1, Tr_f_1, Tr_s_2, Tr_l_2,
   Tr_f_2, al, af, phil,phif : REAL;
dist_s, dist_l, dist_f, T_s, T_l, T_f, T_max_1, T_max_2 : REAL;
vmax_s_1, vmax_l_1, vmax_f_1,vmax_s_2, vmax_l_2, vmax_f_2, xs_prev,
   xl_prev, xf_prev : REAL;
END_VAR

BEGIN



slewlimit:=DB4004.slewlimit;
beta_min:=DB4004.beta_min;
beta_max:=DB4004.beta_max;
pi:=DB4002.pi;
int_reset:=FALSE;

IF reset>0.0 THEN
    t:=0.0;
    i:=1;
    first_time:=TRUE;
    finished:=FALSE;
END_IF;



IF start>0.0 THEN
```

```
IF first_time=TRUE THEN  //Denne delen kjøres bare en gang mellom to
    punkter.

     mb:=beta_max_random-beta_min_random;
     ml:=xl_max-xl_min;
     mf:=xf_max-xf_min;
     ab:=10;
     alofte:=3;
     afolde:=3;
     cb:=100;
     cl:=1;
     cf:=1;


     beta_ref_temp:= ((ab*beta_prev + cb) MOD mb)+beta_min_random;
     xl_ref_temp:=((alofte*lofte_prev + cl) MOD ml)+xl_min;
     xf_ref_temp:=((afolde*folde_prev + cf) MOD mf)+xf_min;

     beta_prev:=beta_ref_temp;
     lofte_prev:=xl_ref_temp;
     folde_prev:=xf_ref_temp;

     beta_ref:=DINT_TO_REAL(beta_ref_temp)/1000.0;
     xl_ref:=DINT_TO_REAL(xl_ref_temp)/1000.0;
     xf_ref:=DINT_TO_REAL(xf_ref_temp)/1000.0;

     dx_l:=xl_ref-xl_act;
     dx_f:=xf_ref-xf_act;
     dx_s:=beta_ref-beta_act;

     WHILE dx_s > 2*pi DO
          dx_s:=dx_s-2*pi;
     END_WHILE;

     WHILE dx_s <-2*pi DO
          dx_s:=dx_s+2*pi;
     END_WHILE;

     IF dx_s > pi THEN
          dx_s:=dx_s-2*pi;
     ELSIF dx_s <-pi THEN
          dx_s:=dx_s+2*pi;
     END_IF;

IF slewlimit=TRUE THEN
     IF (dx_s+beta_act) > beta_max THEN
          dx_s:=dx_s-2*pi;
     ELSIF (dx_s+beta_act) <beta_min THEN
          dx_s:=dx_s+2*pi;
     END_IF;
END_IF;
```

```
al:=0.25*pi*DB4005.d_o_l**2;
phil:=(DB4005.d_o_l**2-DB4005.d_i_l**2)/DB4005.d_o_l**2;
af:=0.25*pi*DB4005.d_o_f**2;
phif:=(DB4005.d_o_f**2-DB4005.d_i_f**2)/DB4005.d_o_f**2;



K1:=0.75; //Vi ønsker å ikke bruke mer enn 75% av maks flow til
    ventilene. Dette vil alltid holde summen av alle flowene under det
     pumpen kan levere.


IF dx_l < 0 THEN
     vmax_l:=K1*DB4005.Q_i_l/(al*phil*0.06);
ELSE
     vmax_l:=K1*DB4005.Q_o_l/(al*0.06);
END_IF;

IF dx_f < 0 THEN
     vmax_f:=K1*DB4005.Q_i_f/(af*phif*0.06);
ELSE
     vmax_f:=K1*DB4005.Q_o_f/(af*0.06);
END_IF;

vmax_s:=K1*DB4005.Q_s*104.7/(DB4005.v_m_s*DB4005.i_s);

Tr_s_1:=vmax_s/amax;
Tr_l_1:=vmax_l/amax;
Tr_f_1:=vmax_f/amax;

dist_s:=ABS(dx_s);
dist_l:=ABS(dx_l);
dist_f:=ABS(dx_f);

T_s:=(dist_s+Tr_s_1*vmax_s)/vmax_s;
T_l:=(dist_l+Tr_l_1*vmax_l)/vmax_l;
T_f:=(dist_f+Tr_f_1*vmax_f)/vmax_f;

IF Tr_s_1 >= 0.5*T_s THEN
     T_s:=((4*dist_s)/amax)**0.5;
     Tr_s_1:=0.5*T_s;
END_IF;

IF Tr_l_1 >= 0.5*T_l THEN
     T_l:=((4*dist_l)/amax)**0.5;
     Tr_l_1:=0.5*T_l;
END_IF;

IF Tr_f_1 >= 0.5*T_f THEN
```

```
    T_f:=((4*dist_f)/amax)**0.5;
    Tr_f_1:=0.5*T_f;
END_IF;

T_max_1:= MAX (IN1:=T_s, IN2:=T_l, IN3:=T_f);
vmax_s_1:=dist_s/(T_max_1-Tr_s_1);
vmax_l_1:=dist_l/(T_max_1-Tr_l_1);
vmax_f_1:=dist_f/(T_max_1-Tr_f_1);



Q_s:=vmax_s_1*DB4005.v_m_s*DB4005.i_s/104.7; //    l/min

IF dx_l< 0 THEN
    Q_l:=vmax_l_1*al*phil*0.06;
ELSE
    Q_l:=vmax_l_1*al*0.06;
END_IF;

IF dx_l< 0 THEN
    Q_f:=vmax_f_1*af*phif*0.06;
ELSE
    Q_f:=vmax_f_1*af*0.06;
END_IF;

//K2 er enda en sikkerhetsfaktor som sikrer at man holder seg under
   maks pumpeflow.
K2:=DB4005.Q_p/(Q_s+Q_l+Q_f);
IF K2 > 1 THEN
    K2:=1.0;
END_IF;
T_max_2:=T_max_1/K2;
Tr_s_2:=Tr_s_1/K2;
Tr_l_2:=Tr_l_1/K2;
Tr_f_2:=Tr_f_1/K2;

vmax_s_2:=dist_s/(T_max_2-Tr_s_2);
vmax_l_2:=dist_l/(T_max_2-Tr_l_2);
vmax_f_2:=dist_f/(T_max_2-Tr_f_2);
xs_prev:=beta_act;
xl_prev:=xl_act;
xf_prev:=xf_act;
first_time:=FALSE;
END_IF;



IF t< Tr_s_2 THEN
   beta_dot_setp:=t*vmax_s_2/Tr_s_2;
ELSIF t<T_max_2-Tr_s_2 THEN
```

```
  beta_dot_setp:=vmax_s_2;
ELSIF t< T_max_2 THEN
    beta_dot_setp:=vmax_s_2-(t-T_max_2+Tr_s_2)*vmax_s_2/Tr_s_2;
ELSE
    beta_dot_setp:=0.0;
END_IF;
IF dx_s < 0.0 THEN
    beta_dot_setp:=-beta_dot_setp;
END_IF;
beta_setp:=xs_prev+beta_dot_setp*steptime;
xs_prev:=beta_setp;

IF t< Tr_l_2 THEN
   xl_dot_setp:=t*vmax_l_2/Tr_l_2;
ELSIF t<T_max_2-Tr_l_2 THEN
  xl_dot_setp:=vmax_l_2;
ELSIF t< T_max_2 THEN
    xl_dot_setp:=vmax_l_2-(t-T_max_2+Tr_l_2)*vmax_l_2/Tr_l_2;
ELSE
    xl_dot_setp:=0.0;
END_IF;
IF dx_l < 0.0 THEN
    xl_dot_setp:=-xl_dot_setp;
END_IF;
xl_setp:=xl_prev+xl_dot_setp*steptime;
xl_prev:=xl_setp;

IF t< Tr_f_2 THEN
   xf_dot_setp:=t*vmax_f_2/Tr_f_2;
ELSIF t<T_max_2-Tr_f_2 THEN
  xf_dot_setp:=vmax_f_2;
ELSIF t< T_max_2 THEN
    xf_dot_setp:=vmax_f_2-(t-T_max_2+Tr_f_2)*vmax_f_2/Tr_f_2;
ELSE
    xf_dot_setp:=0.0;
END_IF;
IF dx_f < 0.0 THEN
    xf_dot_setp:=-xf_dot_setp;
END_IF;
xf_setp:=xf_prev+xf_dot_setp*steptime;
xf_prev:=xf_setp;

t:=t+steptime;
IF t> T_max_2 THEN
    t:=0.0;
    i:=i+1;
    int_reset:=TRUE;
    int_reset_counter:=int_reset_counter+1;
    IF i > tablerows THEN
        i:=tablerows;
        finished:=TRUE;
```

```
    END_IF;
    first_time:=TRUE;
END_IF;

IF finished=TRUE THEN
        beta_dot_setp:=0.0;
        xl_dot_setp:=0.0;
        xf_dot_setp:=0.0;
END_IF;
ELSE
        beta_dot_setp:=0.0;
        xl_dot_setp:=0.0;
        xf_dot_setp:=0.0;
END_IF;


END_FUNCTION_BLOCK
```

C.15 FC3000 - ATAN2

```
FUNCTION ATAN2 : REAL

VERSION : '0.1'
NAME    : 'atan2'
FAMILY  : Crane
AUTHOR  : KJTA

VAR_INPUT
X, Y : REAL;
END_VAR

VAR_TEMP
    pi : REAL;
END_VAR

//pi approximation
pi:= 3.1415927;

//output - http://en.wikipedia.org/wiki/Atan2
IF X>0 THEN
  ATAN2:=ATAN(Y/X);
ELSIF Y>=0 AND X<0 THEN
  ATAN2:=ATAN(Y/X)+pi;
ELSIF Y< 0 AND X < 0 THEN
  ATAN2:=ATAN(Y/X)-pi;
ELSIF Y>0 AND X=0 THEN
  ATAN2:=pi/2;
ELSIF Y<0 AND X=0 THEN
  ATAN2:=-pi/2;
ELSIF Y=0 AND X=0 THEN
  ATAN2:=0;
```

```
END_IF;

END_FUNCTION
```

C.16 FC3001 - Inverse Kin og Forward Kin

```
FUNCTION IK_FK: VOID

VERSION : '0.1'
NAME    : 'Kin_call'
FAMILY  : Crane
AUTHOR  : KJTH

VAR_INPUT
i : INT;
END_VAR

VAR_OUTPUT
xl_ref, xf_ref, beta_ref, xl_actual, xf_actual : REAL;
END_VAR

VAR_TEMP
    alpha_actual, gamma_actual, beta_actual, x_ref, y_ref, z_ref,
       vref_x, vref_y, vref_z : REAL;

// Vinkler som beregnes */
l_b_MB : REAL;
alpha_b_MB : REAL;
l_c_KB : REAL;
gamma_c_KB : REAL;

x_prime_actual, z_prime_actual : REAL;

// Mellomregninger */
radius_ref, alpha_ref, gamma_ref, b_x, b_z, l2_x, l2_z, f1_x, f1_z,
   f2_x, f2_z, sxl_x, sxl_z, sxf_x, sxf_z, b_x_actual : REAL;
 b_z_actual, l2_x_actual, l2_z_actual, f1_x_actual, f1_z_actual,
    f2_x_actual,f2_z_actual, sxl_x_actual, sxl_z_actual, sxf_x_actual
    , sxf_z_actual : REAL;



  y_prime_actual : REAL; //Per definisjon

 pi : REAL;
 a_x : REAL;          // Innfestingspunkt av MB på kongen */
 a_z : REAL;     // Innfestingspunkt av MB på kongen */


b_x_MB : REAL;      //Innfestningspunkt mellom MB og KB, i MB sitt
   oordinatsystem */
```

```
b_z_MB : REAL;       //Innfestningspunkt mellom MB og KB, i MB sitt
    oordinatsystem */


c_x_KB : REAL; //Punkt til krantupp, i KB sitt koordinatsystem */
c_z_KB : REAL;  //Punkt til krantupp, i KB sitt koordinatsystem */


l1_x : REAL;      // Innfestingspunkt av løftesylinder mot kongen */
l1_z : REAL;     // Innfestingspunkt av løftesylinder mot kongen */


l2_x_MB : REAL;    //Innfestingspunkt av løftesylinder mot MB, i MB
    sitt koordinatsystem */
l2_z_MB : REAL;//Innfestingspunkt av løftesylinder mot MB, i MB sitt
    koordinatsystem */

f1_x_MB : REAL; // Innfestingspunkt av foldingssylinder mot MB, i MB
    sitt koordinatsystem */
f1_z_MB : REAL;// Innfestingspunkt av foldingssylinder mot MB, i MB
    sitt koordinatsystem */


f2_x_KB : REAL; // Innfestingspunkt av foldingssylinder mot KB, i KB
    sitt koordinatsystem */
f2_z_KB : REAL; // Innfestingspunkt av foldingssylinder mot KB, i KB
    sitt koordinatsystem */

END_VAR

BEGIN
pi:=DB4002.pi;   //Pi
alpha_actual:=DB4001.Input1;
beta_actual:=DB4001.Input2;
gamma_actual:=DB4001.Input3;
x_ref:=DB4000.Table[i,1];
y_ref:=DB4000.Table[i,2];
z_ref:=DB4000.Table[i,3];

a_x:=DB4004.a_x;
a_z:=DB4004.a_z;
b_x_MB:=DB4004.b_x_MB;
b_z_MB:=DB4004.b_z_MB;
c_x_KB:=DB4004.c_x_KB;
c_z_KB:=DB4004.c_z_KB;
l1_x:=DB4004.l1_x;
l1_z:=DB4004.l1_z;
l2_x_MB:=DB4004.l2_x_MB;
l2_z_MB :=DB4004.l2_z_MB;
f1_x_MB:=DB4004.f1_x_MB;
f1_z_MB:=DB4004.f1_z_MB;
```

```
f2_x_KB:=DB4004.f2_x_KB;
f2_z_KB:=DB4004.f2_z_KB;

//Utregninger av vinkler */

// Lengden mellom pkt. a og pkt. b */
l_b_MB :=
(b_x_MB**2+b_z_MB**2)**0.5;

// Vinkel mellom pkt. a og pkt. b */
alpha_b_MB :=
ATAN2(Y:=b_z_MB,X:=b_x_MB);

// Lengden mellom pkt. b og pkt. c */
l_c_KB :=
(c_x_KB**2+c_z_KB**2)**0.5;

// Vinkel mellom pkt. b og pkt. c */
gamma_c_KB :=
ATAN2(Y:=c_z_KB,X:=c_x_KB);

// Utregning av referanse til sylinderlengde og svingvinkel */
radius_ref :=
(x_ref**2+y_ref**2)**0.5;

 beta_ref :=
 ATAN2(Y:=y_ref,X:=x_ref);

 alpha_ref :=
ACOS((l_b_MB**2+(radius_ref-a_x)**2+(z_ref-a_z)**2-l_c_KB**2)/(2*
    l_b_MB*((radius_ref-a_x)**2+(z_ref-a_z)**2)**0.5))+ATAN2(Y:=(z_ref
    -a_z),X:=(radius_ref-a_x))-alpha_b_MB;

gamma_ref := acos((l_b_MB**2+l_c_KB**2-(radius_ref-a_x)**2-(z_ref-a_z
    )**2)/(2*l_b_MB*l_c_KB))-gamma_c_KB+alpha_b_MB;

 b_x :=
a_x+l_b_MB*cos(alpha_ref+alpha_b_MB);

b_z :=
a_z+l_b_MB*sin(alpha_ref+alpha_b_MB);

l2_x :=
a_x+cos(alpha_ref)*l2_x_MB-sin(alpha_ref)*l2_z_MB;

l2_z :=
a_z+sin(alpha_ref)*l2_x_MB+cos(alpha_ref)*l2_z_MB;

f1_x :=
a_x+cos(alpha_ref)*f1_x_MB-sin(alpha_ref)*f1_z_MB;
```

```
f1_z :=
a_z+sin(alpha_ref)*f1_x_MB+cos(alpha_ref)*f1_z_MB;

f2_x :=
b_x+f2_x_KB*(sin(alpha_ref)*sin(gamma_ref)-cos(alpha_ref)*cos(
    gamma_ref))+f2_z_KB*(cos(alpha_ref)*sin(gamma_ref)+cos(gamma_ref)*
    sin(alpha_ref));

f2_z :=
b_z-f2_x_KB*(cos(alpha_ref)*sin(gamma_ref)+cos(gamma_ref)*sin(
    alpha_ref))+f2_z_KB*(sin(alpha_ref)*sin(gamma_ref)-cos(alpha_ref)*
    cos(gamma_ref));

sxl_x :=
l2_x-l1_x;

sxl_z :=
l2_z-l1_z;

xl_ref :=
(sxl_x**2+sxl_z**2)**0.5;

sxf_x :=
f2_x-f1_x;

sxf_z :=
f2_z-f1_z;

xf_ref :=
(sxf_x**2+sxf_z**2)**0.5;

// omregning fra målt vinkel til sylinderlengde */
b_x_actual :=
a_x+l_b_MB*cos(alpha_actual+alpha_b_MB);

b_z_actual :=
a_z+l_b_MB*sin(alpha_actual+alpha_b_MB);

l2_x_actual :=
a_x+cos(alpha_actual)*l2_x_MB-sin(alpha_actual)*l2_z_MB;

l2_z_actual :=
a_z+sin(alpha_actual)*l2_x_MB+cos(alpha_actual)*l2_z_MB;

f1_x_actual :=
a_x+cos(alpha_actual)*f1_x_MB-sin(alpha_actual)*f1_z_MB;

f1_z_actual :=
a_z+sin(alpha_actual)*f1_x_MB+cos(alpha_actual)*f1_z_MB;

f2_x_actual :=
```

```
b_x_actual+f2_x_KB*(sin(alpha_actual)*sin(gamma_actual)-cos(
    alpha_actual)*cos(gamma_actual))+f2_z_KB*(cos(alpha_actual)*sin(
    gamma_actual)+cos(gamma_actual)*sin(alpha_actual));

f2_z_actual :=
b_z_actual-f2_x_KB*(cos(alpha_actual)*sin(gamma_actual)+cos(
    gamma_actual)*sin(alpha_actual))+f2_z_KB*(sin(alpha_actual)*sin(
    gamma_actual)-cos(alpha_actual)*cos(gamma_actual));

sxl_x_actual :=
l2_x_actual-l1_x;

sxl_z_actual :=
l2_z_actual-l1_z;

xl_actual :=
(sxl_x_actual**2+sxl_z_actual**2)**0.5;

sxf_x_actual :=
f2_x_actual-f1_x_actual;

sxf_z_actual :=
f2_z_actual-f1_z_actual;

xf_actual :=
(sxf_x_actual**2+sxf_z_actual**2)**0.5;
END_FUNCTION
```