NTNU

TTK 4235 - Tilpassede Datasystemer

Gruppe 19

Institutt for teknisk kybernetikk

# Rapport
# Heislab

***Forfattere:***
Ole Andreas Wammer
Vemund Berg

1. mars 2020

# Sammendrag

Vi har i denne laboppgaven programmert en heis ved hjelp av V-modellen til å oppføre seg på en realistisk måte. Etter en grunnleggende planleggingsfase, ble de ulike funksjonene implementert og testet enkeltvis, dette kalles enhetstesting. Når hele systemet var implementert utførte vi flere implementasjonstester, som tester hele systemet ut i fra kravspesifikasjonene. Resultatet ble et fullverdig system i henhold til oppgaven, men som fortsatt har forbedringspotensiale i kode og logikk.

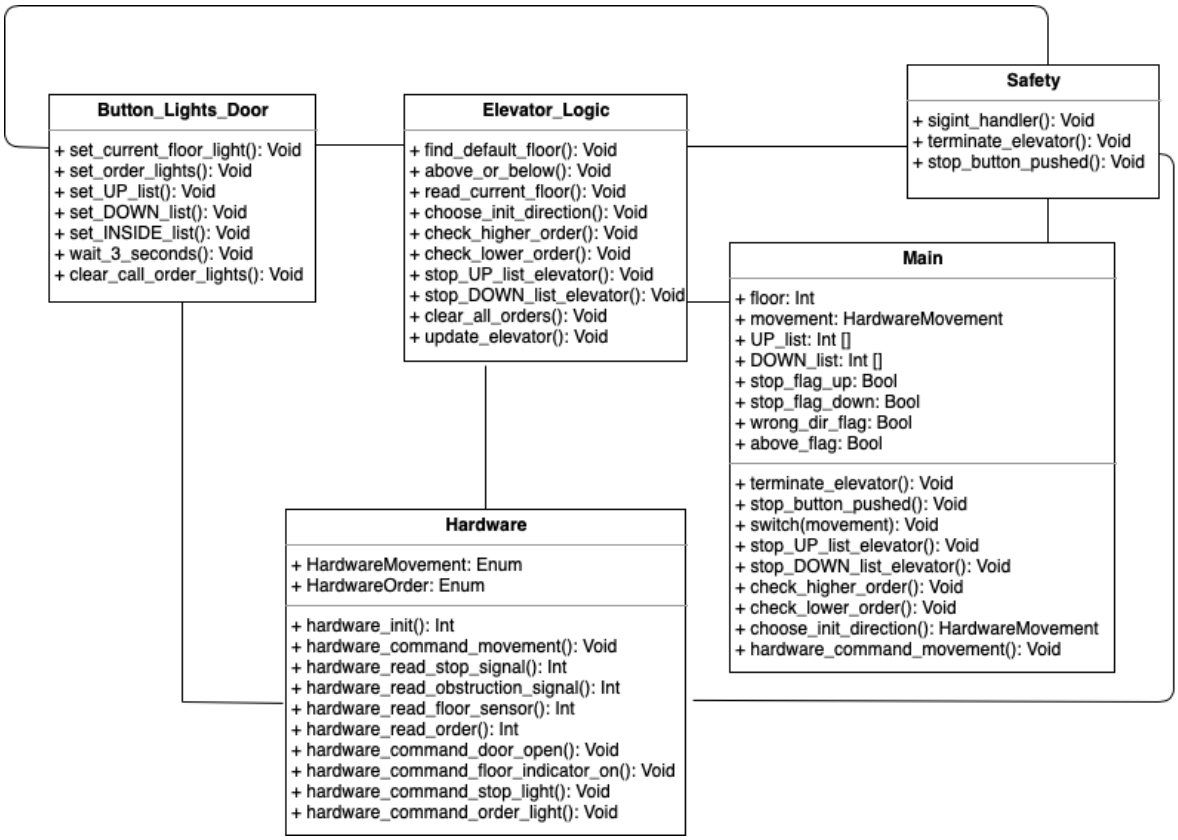
# Innhold



# Figurer

# Introduksjon

Heisprosjektet i faget tilpassede datasystemer er et lab prosjekt med lang historie på Kybernetikk studiet. Her får studentene testet seg på planlegging, programmering og testing av en heismodell. Laboppgaven dekker et bredt spekter med ny kunnskap studentene skal lære seg, blandt annet git, doxygen og operativsystemet Ubuntu. Oppgaven består av tre deler, som hver teller 10 %. Disse delene er dokumentasjon, kodekvalitet og funksjonalitet. Det siste punktet vurderes gjennom en Factory Acceptance Test (FAT) som beskriver hvorvidt studentene har implementert et fullverdig system ut i fra kravene.

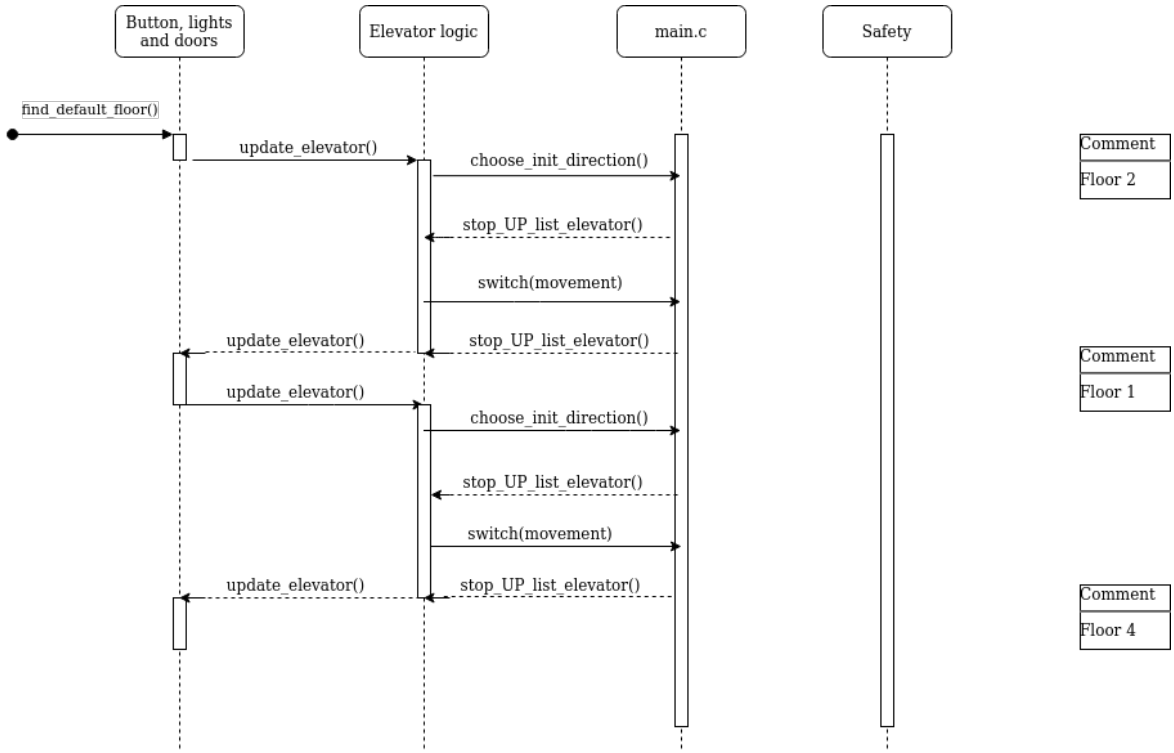



**Figur 1.1:** Klassediagram over arkitekturen til systemet

# 1 Overordnet arkitektur

Arkitekturen er delt inn i fire logiske blokker. En modul dekker knappetrykk, lys og dør, en dekker alle sikkerhetsfunksjoner, en annen dekker heisfunksjoner og main filen kjører alt dette kontinuerlig for å styre heisen. Dette kan sees i Figure 1.1. Bestillinger blir håndtert ved hjelp av to arrays. Bestillinger som skal opp legges i UP_list og bestillinger som skal ned legges i DOWN_list. Når noen inne i heisen trykker på en knapp, legges denne bestillingen i begge arrays, slik at de alltid blir håndtert. Dette er en relativt enkel metode å implementere en heislogikk på, som også er intuitivt grei å forstå.

Figure 1.2 viser et sekvensdiagram som visualiserer scenarioet beskrevet i oppgaveteksten. Her bestiller en person heisen, som er i etasje 2, til etasje 1. Personen trykker altså på knappen UP 1. While løkken i main vil kjøre helt til heisen er i etasje 1, da vil heisen stoppe og plukke opp personen. Når personen trykker etasje 4, vil heisen gå oppover helt til den er framme, og kan slippe av personen. While løkken i main vil kjøre kontinuerlig og sjekke om den er framme, det samme vil sikkerhetsfunksjonene i

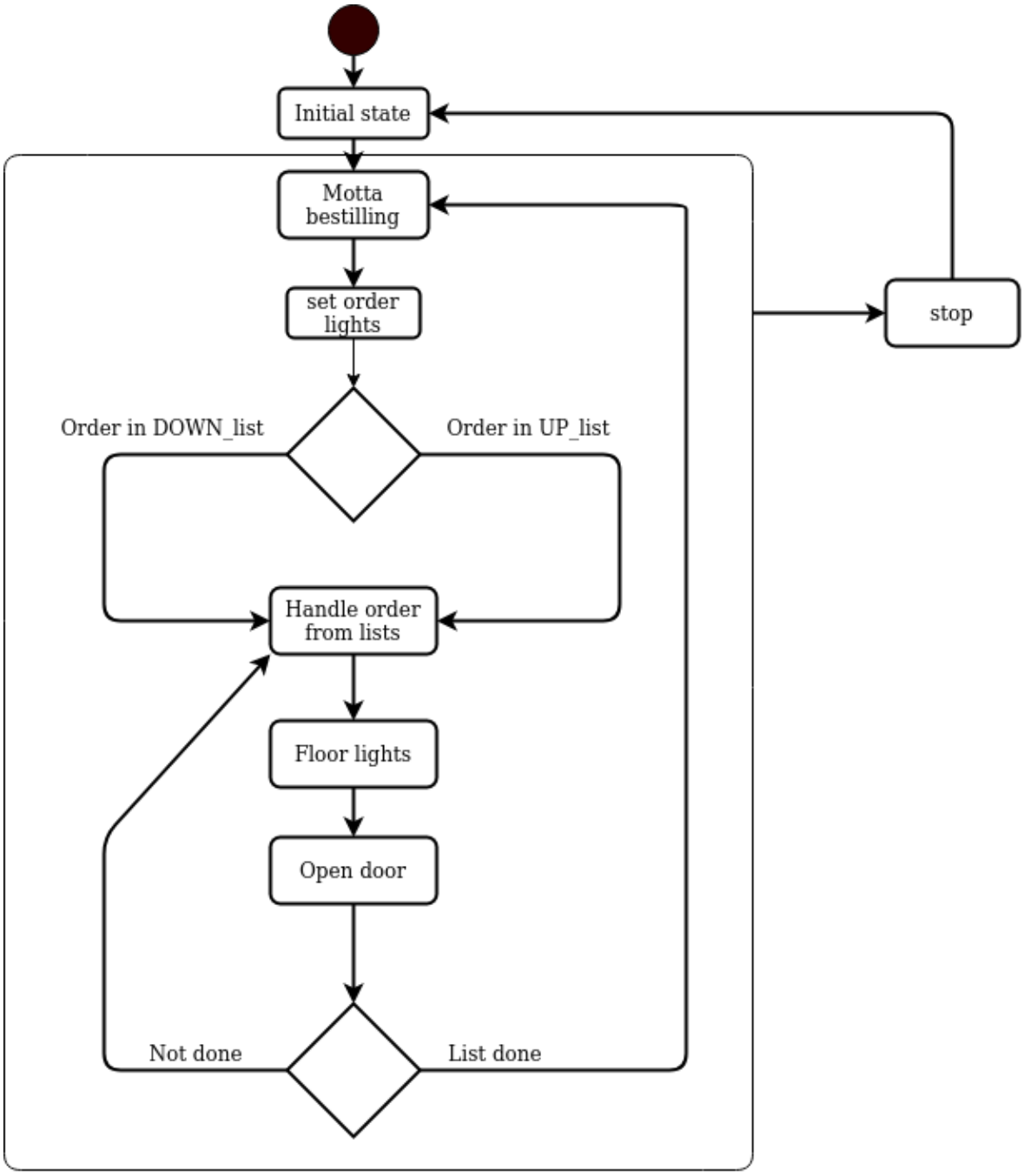



**Figur 1.2:** Sekvensdiagram av scenario gitt i oppgaveteksten

safety. Dette er ikke illustrert for å forenkle diagrammet.

Figure 1.3 viser et tilstandsdiagram over systemet. Fra start går heisen inn i en initital state, før den tar i mot bestillinger. Når den mottar bestillinger legges de enten inn i UP_list eller DOWN_list, før de håndteres av heislogikken. Denne sekvensen går kontinuerlig, med mindre stoppknappen er trykt inn. Stoppknappen er illustrert slik at det vises at den kan kalles hele tiden, før den sender heisen tilbake til initial state. Når heisen håndterer bestillinger, vil alltid lys og dør sjekkes og oppdateres om nødvendig. Når det er tomt for bestillinger går heisen tilbake til start.

Arkitekturen er designet for å være oversiktelig slik at du enkelt kan finne fram til det du ønsker. Det er dermed ikke for mange filer, men heller flere funksjoner som greit forklarer hva de gjør. Løsningen med to arrays er også brukt på grunn av simplisitet. For hver etasje trenger kun heisen sjekke om en bestilling er høy i arrayen for å vite om den skal stoppe eller ikke. I den kontinuerlig kjørende while løkkene er det lagt inn en switch statement som velger handling ut i fra nåværende bevegelse til heisen.




**Figur 1.3:** Tilstandsdiagram til systemet

# 2 Moduldesign

Main-filen er en evig løkke med kontinuerlige iterasjoner over bestillingslistene. En for om bestillingen var opp, en for ned. Når første bestilling har kommet, settes heisen i bevegelse basert på hvor bestillingen kom fra, og hvilken liste den ble lagt inn i. Heisen er nå i en state, kjører opp eller ned, helt til den treffer en etasje sensor og Elevator_logic avgjør om heisen skal stoppe eller ikke. Vi sjekker nå om det er flere bestillinger i listen vi behandler, i samme retning vi kom i, eventuelt bytter retning eller stopper og venter på en ny bestilling.

Denne måten å implementere et køsystem på gir en veldig intuitiv tilnærming til heisen, der vi løpende sjekker om vi skal stoppe, kjøre videre i listen vi er i, eller endre state. Det gjør det lett å lese og mye enklere å implementere enn å på forhånd lage en optimalisert rute heisen skal kjøre.

Buttons_lights_doors er en enkel modul der medlemsfunksjonene kalles gjennom hele programmet, der alt av bestillinger legges inn i listene, alle lys i forhold til spesifikasjonen skal settes og fjernes. Her er også funksjonen wait_3_seconds som kalles når heisen skal stoppes. Dette er en enkel while løkke som fortsetter å ta i mot bestillinger, åpner døren, og bryter ut av løkken etter 3 sekunder.

Til slutt har vi en modul som kun tar seg av sikkerhet. Her har vi mulighet for å terminere koden, samt håndtere obstruksjons- og stoppknappen.

# 3 Testing

Systemet ble planlagt og implementert ved hjelp av V-modellen. Testingen ble delt opp i to deler: Enhetstesting og implementasjonstesting. Disse er beskrevet under.

## 3.1 Enhetstesting

Enhetstesting vil si å teste spesifike subdeler av systemet. Vi utførte en enhetstest for hver gang en ny funksjon ble skrevet, for å forsikre oss om at koden fungerer. Et eksempel på dette er å prøve å stoppe heisen i 3 sekunder med wait_3_seconds() funksjonen vår. Modulene krever mange funksjonskall med hverandre, og GDB ble brukt flittig for å identifisere hvilke funksjoner som ble kalt, fra hvor og eventuelt de som feilet.

### 3.2 Implementasjonstesting

Etter all kode er implementert er det tid for å implementasjonsteste systemet. Det vil si å teste alle mulige situasjoner på hele systemet for å sjekke om den er godkjent ut i fra kravspesifikasjonene gitt i oppgaven. Vi går gjennom hele listen av spesifikasjoner, der vi setter opp ulike tester heisen må håndtere. Dette gir oss konkrete feil vi må rette opp i, eller en bestått test. Et eksempel på en test vi utførte var at heisen var i toppetasjen, og alle opp-knappene ble trykket. Da skal heisen gå helt ned og stoppe på de andre på vei opp. Vi testet også om den prioriterte å gå opp til toppetasje ved bestilling innvendig eller ned til andre hvor noen hadde trykt ned, om forrige bestilling var opp på tredje etasje. Slike vanskelige situasjoner er avgjørende for om heisen er implementert på en logisk måte.

## 4 Diskusjon

Vår implementasjon av køsystemet ser vi i etterkant var for svak. Isteden for å ha en helt separat modul som tok seg av bestillinger og lagde en optimal rute, endte vi med en main-loop som bar preg av at heisen ikke hadde en rute den skulle følge, men mer bevegelse etter mange if-else statements og situasjoner vi måtte ta hånd om. Etter hvert stopp, havner heisen alltid i samme state, hvor den skal velge neste bevegelse. Dette førte til unødvendig vanskelig kode, og kunne vært løst ved å huske forrige bestillingstype og bevegelse.

Et annet problem var at heisen ikke husket hvor den var om vi stoppet den midt mellom to etasjer. Vi løste dette ved å gjøre en rimelig antagelse om at heisens retning før stopp-knappen var trykket, ga oss svar på om den er under eller over etasjen den viser. Var den på vei oppover, og viser etasje to, er den mellom to og tre. Omvendt er den mellom første og andre. Dette er en enkel og grei antagelse, men en løsning man kan manøvrere seg rundt. For eksempel, om man kjører nedover igjen etter stans mellom andre og tredje etasje, og deretter trykker stopp igjen før den passerer andre, vil heisen tro at den er under. Dette gir følgelig problemer med nye bestillinger som kommer inn. Dette mener vi fortsatt er en god løsning på problemet, når måten man manøvrerer seg rundt problemet er søkt og urealistisk for en fysisk heis.

Punkt S 3 i sikkerthetsspesifikasjonen sier at heisen aldri skal kjøre utenfor området definert av første- og fjerde etasje. Vi har ingen sikkerhetsfunksjon som passer på dette, men i vår implementasjon er det ingen logisk grunn til at den nettopp skal kjøre utenfor området. Det finnes ingen bestillinger som kan tas over fjerde- eller under første etasje, dermed vil den aldri kunne kjøre utenfor. Og hvis den kjører til ytteretasjene, er det fordi en bestilling er tatt der, og heisen skal stoppe.

# 5 Konklusjon

Heissystemet vi har implementert er enkelt og oversiktelig, men til tider med unødvendig vanskelig kode. Resultatet av testingen viste at systemet fungerer på ønsket måte, og vil bestå testene gjort i FAT-prøven. Det finnes andre måter å programmere blant annet while løkken på, som ville gjort koden bedre, selv om det fungerer i henhold til kravspesifikasjonene. Vi føler dermed vi har implementert en fullverdig kode i henhold til oppgaveteksten, men det finnes fortsatt forbedringer å gjøre på systemet.