Rapport til Heislab

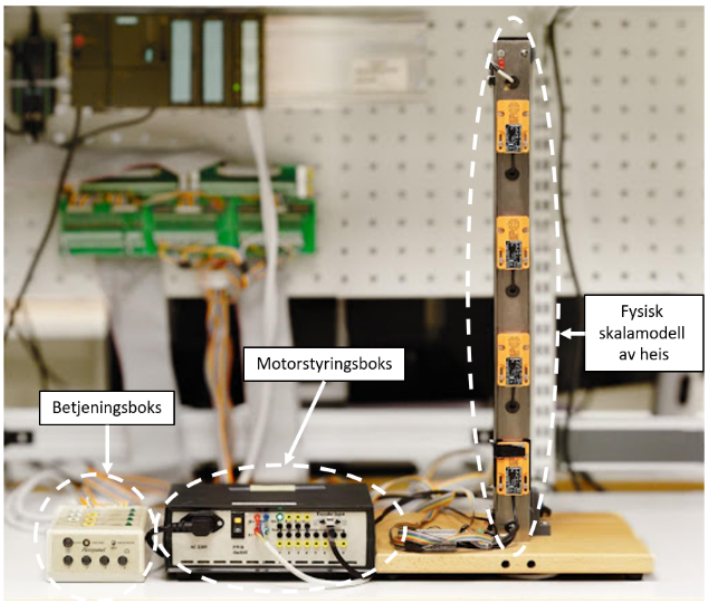
Tittel: Heis_veis	
Forfatter: Erlend Withammer-Ekerhovd og Veronica Kenworthy	
Versjon: 1.0	Dato: 15.03.21

Innhold

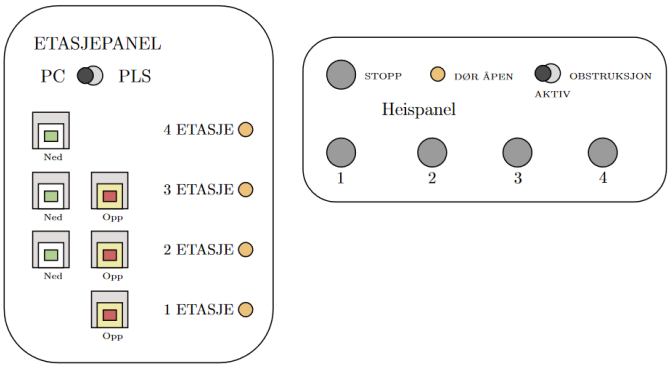
---

1 Problembeskrivelse

Heislab er et prosjekt for studenter ved Kybernetikk og Robotikk i det 4. semesteret. I denne laben ble det jobbet med utvikling av C-kode for styring av en heismodell ved bruk av V-modellen.
Heismodellen består av en betjeningsboks, en motorstyringsboks og den fysiske heisen som vist i Figur 1. Betjeningsboksen består av ulike typer bestillingsknapper og lys som vist i Figur 2.



Figur 1: Laboppsett



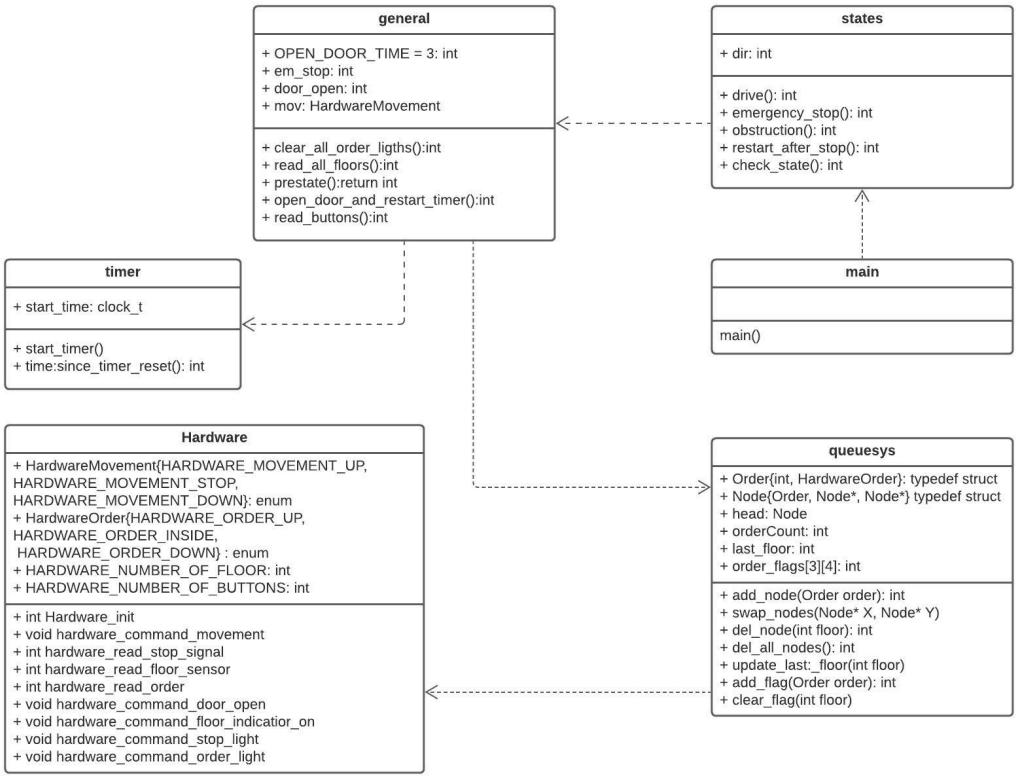
Figur 2: Betjeningsboksens panel

2 Overordnet arkitektur

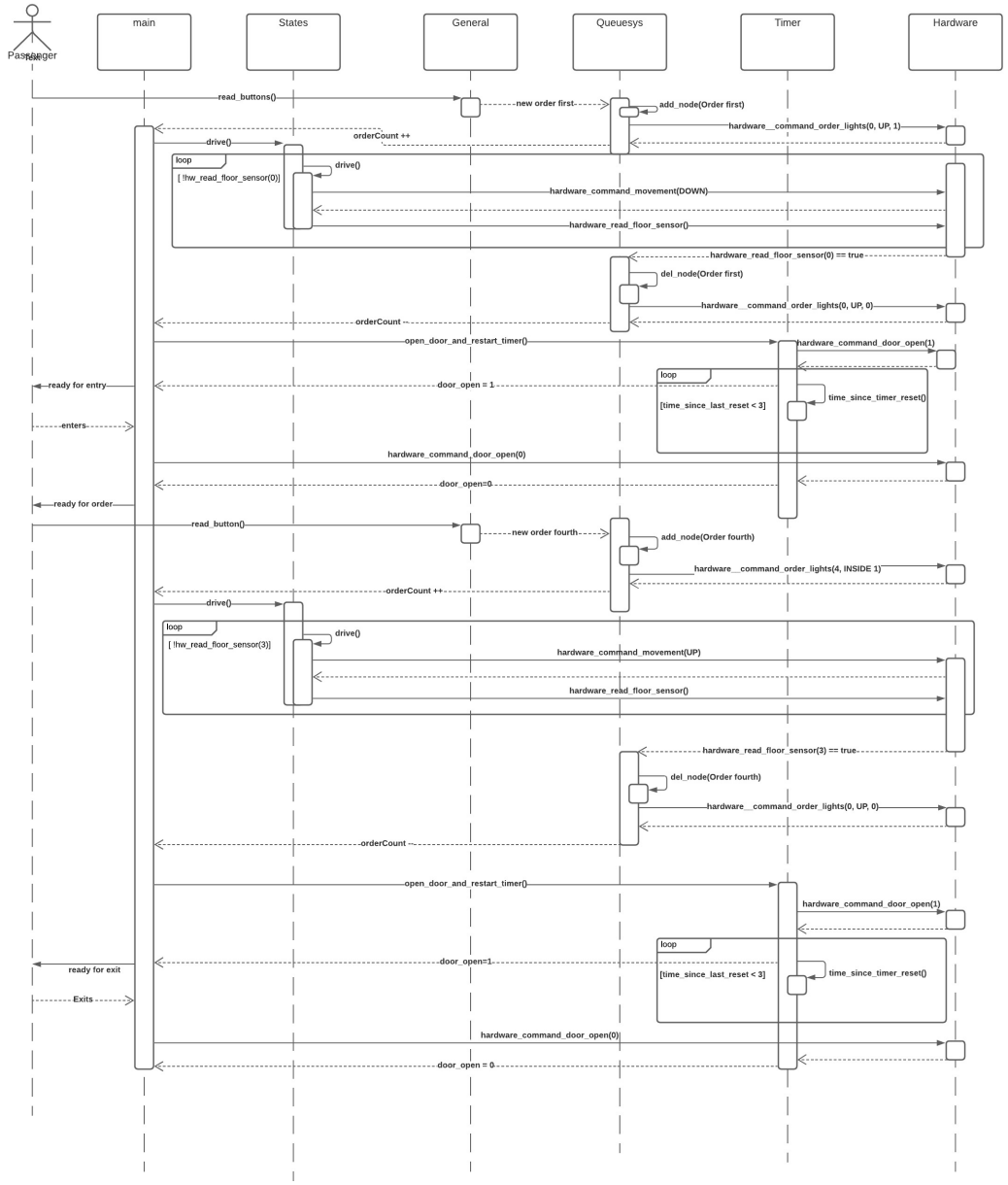
Den overordnete arkitekturen er ment for å gi et overblikk og en grunnleggende forståelse over funksjonaliteten til heisen. For å forstå relasjonen modulene i styringssystemet har til hverandre er klassediagrammet i Figur 3 nyttig. For å videre illustrere hvordan de forskjellige modulene henger sammen og deres samspill, egner det seg godt med et eksempel. I sekvensdiagrammet i Figur 4 presenteres det et eksempel på en person som bestiller heisen fra første, og deretter skal opp til fjerde. Den implementerte koden bak denne prosessen kan kort forklares som en forenklet tilstandsmaskin som looper check_state(). Dette er en funksjon som returnerer ulike tall etter hvilken tilstand koden skal settes i. Main funksjonen switcher mellom de ulike returverdiene til check_state(), og velger om heisen enten skal kjøre, gjennomføre et nødstopp, eller reagere på et obstruksjonssignal. Dette er demostrert i tilstandsdiagrammet i Figur 5.

Heisbestillinger legges i en dobbel-lenket liste om det ikke allerede finnes en bestilling til den etasjen i køen. Uansett om det er en bestilling eller ikke til etasjen så settes et bestillingsflag også i order_flags som er en 3x4 matrise med én mulighet for hver av de 3 ulike bestillingstypene, i hver av de 4 etasjene.

Grunnen til at akkurat denne arkitekturen er gunstig som et heisoppsett er at heisen får en svært enkel og godt modulert kode. At koden er enkel gjør den lettere å vedlikeholde og mer robust. Ettersom koden looper gjennom check_state() kontinuerlig i stedet for å holde seg i en tilstand, vil ikke koden bli fanget i en evig tilstandsløkke om et uforutsett spesialtilfelle oppstår. At koden er modulær med tydelige skiller mellom modulene gjør at det blir lettere å feilsøke problemer som oppstår under utvikling og ved vedlikehold.



Figur 3: Klassediagram

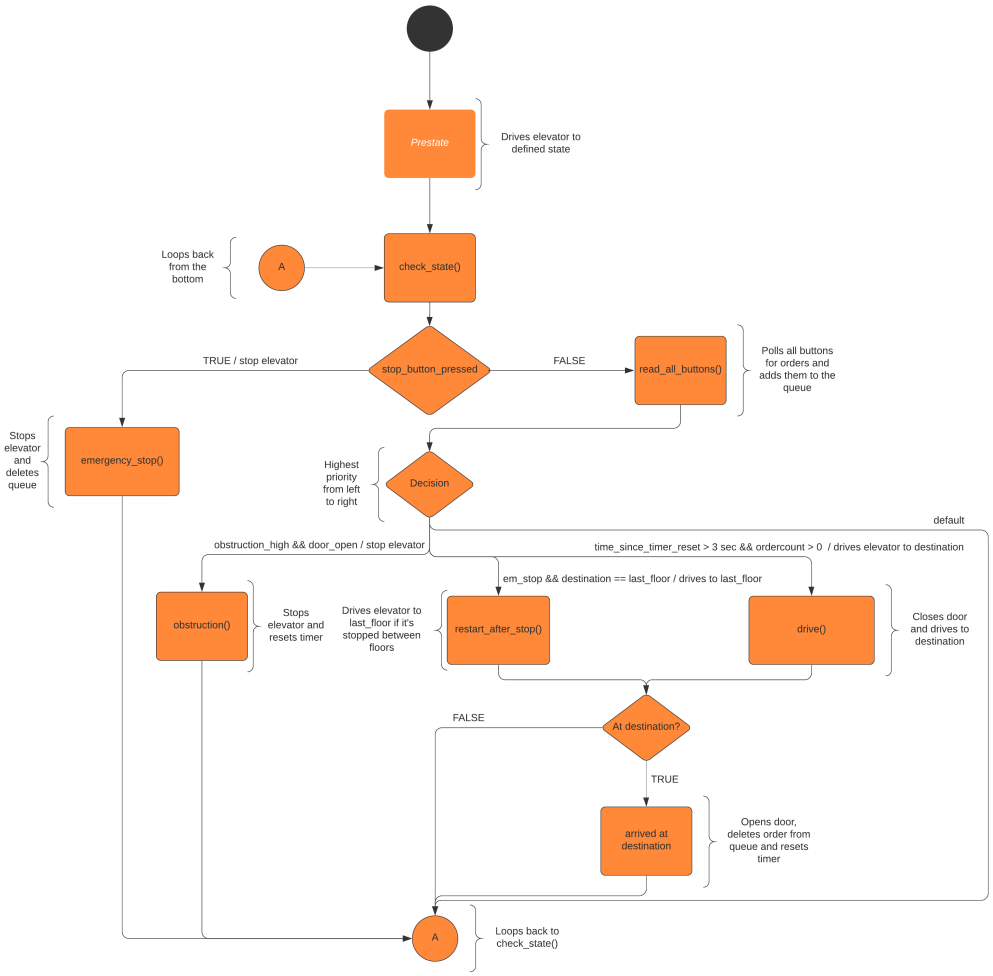


Figur 4: Sekvensdiagram

3 Moduldesign

3.1 Timer

Tidtakingsmodulen består av funksjon som restarter en stoppeklokke og en funskjon som sjekker differansen i sekunder mellom nåværende tid og forrige gang stoppeklokken ble restartet. Grunnen til at det blir brukt en stoppeklokke med sekunders nøyaktighet er at heisen ikke trenger å vite den faktiske tiden, og en nøyaktighet på millisekunder er unødvendig. Ettersom



Figur 5: Kodens arkitektur

å bruke en stoppeklokke ga en enklere implementasjon ble det valgt.

3.2 Queuesys

Køsystemet består av en dobbel-lenket liste og en 3x4 matrise kalt order_flags. Det er mange ulike måter å lage et køsystem, hvor det enkleste vil være en statisk array. I en array vil alle bestillinger ligge etter hverandre i minnet som gjør det lett å gå gjennom listen fra start til slutt.

Problemet med en statisk array er at en på forhånd må definere størrelsen som i dette tilfelle ville vært på 12 bestillinger[1]. Siden det ikke alltid vil være 12 bestillinger i køen må en legge inn en idle-bestilling i resten. Et større problem er at når heisen når en etasje skal alle bestillinger i den etasjen slettes, selv om de ikke nødvendigvis er etterhverandre i køen. Dette kan føre til at en får idle bestillinger i mellom vanlige bestillinger som enten må håndteres med et

[1]Det er 4 etasjer og 3 ulike bestillingstyper i hver etasje

sorteringsalgoritme kall etter hvert stopp eller komplekse hjelpefunksjoner som vil være svært vanskelig å generalisere til n etasjer.

Med en lenket liste blir ikke dette et problem da alle elementene binnes sammen av pekere i stedet for den fysiske lokasjonen i minnet. En kan dermed dynamisk fjerne og legge til elementer fra eller til hvor som helst i listen uten noen form for "idlebestilling, og sortering blir dermed ikke nødvendig. Ulempen er at det er vanskeligere å implementere.

Ettersom alle bestillinger til en etasje skal slettes samtidig kan en legge ulike bestillinger til samme etasje etterhverandre i køen, som vil gjøre at en ikke må itterere gjennom hele listen når en skal fjerne en etasje. Allikevel ble implementeringen lettere med en matrise av flag over de ulike bestillingene som heller kunne sjekkes. Denne matrisen ble kalt order_flags og er en 3x4 matrise, ettersom det er 3 mulige bestillingstyper til hver av de 4 ulike etasjene. Dette skapte en hybridløsning med en dobbel-lenket liste som kø med rekkefølgen heisen skulle kjøre til etasjene, og en matrise med flag som sjekkes hver gang heisen kjører forbi en etasje. For eksempel hvis første element i køen er en bestilling til 3. etasje og heisen kjører oppover forbi 2. sjekkes 2. etasjes oppflag i matrisen for å finne ut om heisen skal ta med noen på veien.

3.3 General

I general ble det laget flere funksjoner som kunne være nyttig for generell heisbruk. Funskjoner som read_all_floors() og read_buttons() implementers i denne modulen og brukes hendholsvis til å finne ut om heisen er i en etasje, og i såfall hvilken, og til å ta bestillinger. Her ligger også funsksjonen prestate() som kjører heisen til en definert tilstand om den begynner mellom to etasjer. For å holde main modulen ryddig ble det valgt å lage en egen modul for slike funskjoner da de alle var nyttige funskjoner som ikke logisk passet inn i de andre modulene.

3.4 States

Implementeringen av heisfunskjonen er som nevnt en forenklet tilstandsmaskin. I State modulen ble det valgt å bruke en funksjon som kontinuerlig kalles fra main for å velge hvilken tilstand heisen skal være i. Fordelen med denne formen for implementasjoen er at den er mer robust mot uforutsette senarioer, mens den beholder liten kompleksitet i koden. Dette ga lettere feilsøking i da få funskjoner ble kalt fra flere steder. De ulike tilstandene implementert i States er forklart under.

3.4.1 Emergency

Tilstanden som aktiveres når den rød stoppknappen trykkes inn er emergency_stop. I emergency_stop stoppes heisen og lyset på stoppknappen slås på. Om heisen står i en etasje vil døren åpnes og timeren på døren vil restartes. Et flag kalt em_stopp blir også satt høyt. Det tas ingen bestillinger, og hele køen blir slettet.

3.4.2 Obstruction

Om både obstruktionsbryteren er høy og døren er åpen vil tilstanden bli satt til obstruction så lenge stoppknappen ikke er trykket. I obstruction holdes døren åpen og heisen i ro. I obstruksjons tilstanden tas det bestillinger, selv om heisen ikke kjører før obstruksjonen fjernes.

3.4.3 Restart after stop

Hvis em_stop flagget er høyt og koden ikke har havnet i en foregående tilstand blir restart_after_stop() kalt. Dette er modulen som behandler spesialtilfellet hvor heisen ble stoppet mellom to etasjer og første bestilling er til etasjen hvor heisen sist var. Da vil heisen egentlig tro at den allerede er i den etasjen, ettersom last_floor variabelen nå er lik ønsket destinasjon. Funksjonen gjør at uansett hvor mange ganger heisen blir stoppet mellom to etasjer, og uansett hvilken den kalles til vil den gå den korteste distansen til destinasjonen. Tilstanden er altså kun til for å ordne opp i uønsket oppførsel forårsaket av nødstopp.

3.4.4 Drive

Drive er tilstanden som håndere all kjøring til destinasjoner når em_stop flagget er lavt. Heisen kjører mot etasjen til det første elementet i køen. Når heisen når en etasje sjekkes det om det er noen inne i heisen som skal til etasjen ved å sjekke order_flags. Dette er en matrise som holder styr på alle de ulike typene av ordre som er ubehandlet, men er ikke det samme som køen. Heisen sjekker også om det er noen utenfor etasjen som skal i samme retning i hver etasje den kjører forbi.
Modulen behandler også specialcasen hvis flere av etasjene under heisen har en opp bestilling så skal heisen kjøre til den nederste av disse for så og stoppe på resten på vei opp. Dette funker også andre veien.
Ellers kjører heisen til den når sin destinasjon.

3.5 Main

Main er en svært ryddig og enkel modul ettersom det meste av hjelpefunkjoner er lagt i general som diskutert i underseksjon 3.3. Det ble valgt å holde main ryddig så det skulle være tydelig hvordan heisen fungerte. I mainfunksjonen kjøres prestate() som sender heisen til en definert tilstand for så å kjøre en enkel løkke med en switch som velger på de ulike returverdiene til check_state().

4 Testing

En av de høyest prioriterte delene av et prosjektet er testingen. For å vite om heisen fungerer på en ønskelig og trygg måte, er det essensielt med gode testerutiner. I starten av prosjektet

ble det derfor laget en tabell med punktene som skulle oppfylles for at heisen skulle fungere optimalt. For å forsikre seg om at kravene ble tilfredstillende nådd, ble det preparert tester som systematisk skulle gjennomføres etterhvert som modulene ble fullført.

4.1 Modultesting

4.1.1 Queuesys

Punkt	Beskrivelse	Test	Resultat
MQ1	Alle bestillinger skal legges inn i køen, uansett hvilken bestillingskanpp som trykkes.	Lagde funksjonen print_queue(), for å se at alle bestillingene gitt ble lagt inn i køsystemet.	Funker. Alle bestillingene blir tatt.
MQ2	Bestillingslys skal lyse helt til bestillingen er fullført.	Gir heisen en rekke bestillinger, og ser om bestillingslyset begynner å lyse, og deretter om det slukker når den kommer til den tilhørende etasjen.	Funker. Lyset skrur seg og på og av til riktig tid.
MQ3	Avhengie av hva slags type bestilling det er, skal det settes et flag i order_flags[ ].	Gir heisen en rekke forskjellige bestillinger, printer deretter arrayen, og observer at antall flag samsvarer med bestilligene gitt i de ulike etasjen.	Funker, flagg og type bestilling er ekvivalente.
MQ4	Når heisen ankommer en etasje med en bestilling, skal alle flagg i denne etasjen fjernes, samt bestillingene fra køen.	Bruker print_queue(), for å være sikker på at alle bestillinger i etasjen heisen stopper slettes.	Funker, alle bestillingene til tilhørende etasje slettes.

4.1.2 Timer

Punkt	Beskrivelse	Test	Resultat
MT1	I den bestemte tiden timeren går skal en tilstand opprettholdes. Når tiden er fullført skal tilstanden endres.	Setter timeren til 3 sekunder, og bruker funksjonen hardware_command_door_open. Måler timingen nøye, og observerer at lyset til døren ikke skrur seg av før tiden er fullført.	Funker, døren lukkes ikke før etter 3 sekunder.

4.1.3 General

Punkt	Beskrivelse	Test	Resultat
MG1	Når heisen staret skal den gå til definert tilstand.	Starter heisen mellom to etasjer, og ser hvordan den håndterer oppstartfasen i denne tilstanden.	Funker. Observerer at heisen går opp til den etasjen den er nærmest.
MG2	Uansett hvilken knapp som trykkes, skal systemet klare å tolke hva slags bestilling det er.	Bruker print_queue(), for å se at bestillingene i køen tilsvarer bestillingen gitt.	Funker. Ordrene og køen er samfallende.
MG3	Heisen skal klare å registrere hvilken etasje den er i, og tenne tilhørende etajselys. Dette skal ikke endres før etasjen kommer til en ny etasje.	Observerer hvordan etasjelysene endrer seg avhengie av hvor heisen er, og med ulike type ordre.	Funker, etasjelyset tennes, og endres ikke før den kommer til en ny etasje.
MG4	Når heisen ankommer en etasje det er gjort bestilling til, skal døren åpnes i 3 sekunder, for deretter å lukkes.	Timer tiden det tar fra heisen når en etasje og døren åpnes, til den lukkes.	Funker. Døren holder seg åpen i 3 sekunder.

4.1.4 States

Punkt	Beskrivelse	Test	Resultat
MS1	Heisen sitt styresystems skal kjøre heisen til etasjen til første order, og ta med ordre, som er i samme retning, på veien	Gir heisen en rekke diverse bestillinger, og ser hvordan heisen tar disse.	Funker, heisen går til første element i køen, og plukker kun opp de bestillingene som er i samme retning.
MS2	Når stoppknappen trykkes, skal heisen stoppe momentant, og slette alle bestillinger i køen.	Tester stoppknappen, når heisen er i en etasje, mellom to etasjer, samt når den har bestillinger og når den ikke har det. Bruker også print_queue(), for å se hvordan køen påvirkes av knappen	Funker. Heisen stoppet momentant i alle fire scenarioene.
MS3	Når obstruksjonssignalet er høyt i en etasje, og døren er åpen, skal døren holdes åpen. Når signalet går lavt, skal døren lukkes etter tre sekunder	Tester at døren holdes kontinuerlig åpen, både med og uten bestillinger i køen, og at dette ikke endres før obstruksjonssignalet går lavt.	Funker. Døren holdes åpen, og endres ikke før signalet går lavt.
MS4	Etter et nødstopp skal heisen vite hvor den er, og ta bestillinger på vanlig måte.	Observerer hvordan heisen vil håndtere tiden etter et nødstopp mellom to etasjer.	Funker. Heisen vil ta bestillinger på vanlig måte.

4.2 Integrasjonstesting

Integrasjonstesten var den avsluttende testen når alle modulene var fullført. Denne testen var ment for å gjøre rede for om modulene samspiller på en slik måte at kravene oppfylles.

Ved oppstart av programmet, skal systemet alltid komme til en definert tilstand, før den har oppnådd dette skal heissystemet ignorere alle forsøk på å ta bestillinger. For å teste dette ble heisen startet mellom to etasjer, mens man prøvde å trykke inn en bestilling. Når denne testen ble gjennomført kjørte heisen til nærmeste etasje, og ved å observere outputen til print_queue(), kunne man se at ingen bestillinger ble lagt inn. Dette forsøket visste altså at krav O1, O2 og O3 ble tilfredstillt, samt R2 og R3.

Måten heisen håndtere bestillinger var også et viktig sjekkpunkt. Prossen bak håndteringen

ble analysert ved å bruke dette scenarioet:

1. Heisen står stille i første etasje.
2. Heisen mottar en bestilling ned i fjerde etasje.
3. Deretter mottar heisen en bestilling ned i andre etasje.
4. Så en besttiling opp, og en ned i tredje etasje.
5. print_queue() printes kontinuerlig.

Når heisen mottok denne sammensetningen av bestillinger, reagerte den ved å kjøre forbi andre etasje. Deretter stoppet den i tredje etasje, åpnet døren i 3 sekunder, lukket den, og kjørte så videre til fjerde etasje. Igjen holdt døren seg åpen i kun 3 sekunder, for å så kjøre ned til andre etajse og stoppe der. Mens dette scenaroiet utspillte seg, kunne man observe fra køen at selv om det var to ulike bestillingstyper i tredje, ble begge slettet. Utifra denne testen kunne man altså observere at flere av kravene for at heisen skulle fungere optimalt var oppfylt. Alle bestillinger blir tatt, heisen betjente ikke bestillinger utenfor heisrommet når den var i bevegelse i motsatt retning, alle personene i tredje etasje gikk på heisen, selv om de ikke skulle i samme retning, og heisen stopper når køen ble tom. Dette tilfredstilt punkt H1, H2, H3 og H4.

I denne eksaminasjonen kunne man også se hvordan lysene ble påvirket. Bestillingslysene ble skrudd på når de tilhørende knappene ble trykket, og skrudde seg ikke av før heisen hadde fullført bestillinge. Dette oppfylte punkt L1 og L2. Det ble videre observert at etasje-lyset ble tent når heisen kom til en etasje, og at det ikke ble endret før heisen nådde en ny etasje. Dette sjekket av L3, L4 og L5. Når heisen befant seg i en etasje ble det i tillegg lagt merke til at døren oppførte seg på ønskelig måte. Den holdt seg åpen i ønsket tid, og heisen bevegde seg aldri mens døren var åpen. D1, D2, S1 og S2 ble dermed også bekreftet som fungerende.

Når testingen av bestillingssystemet, og at kjøringen foregikk på en forsvarlig måte var gjennomført, var det over til testingen av de to gjennstående elementene; stoppknappen og obstruksjonsbryteren. For å teste funksjonaliteten til disse, ble det gjennomgått en rekke sammensetninger av forsøk. For å teste stoppknappen, ble den trykket inn både når den befant seg i en etasje og når den var mellom to etasjer. Straks knappen ble trykket stoppet heisen momentant uansett hvor den befant seg, og ved å se på print_queue() kunne man se at køen ble fullstendig tømt. Det ble videre observert at stoppknappen kun lyste når knappen ble holdt inne, og at den slukket umidellbart den ble sluppet. Hvis heisen stod i en etasje ble døren åpnet, og holdt åpen i hele tidsrommet knappen var aktiv, og lukket tre sekunder etter at den ble sluppet. Denne responsen bekreftet at punkt L6, D3, S4 og S5 var funksjonelle. I tillegg ble alle forsøk på å gjøre bestillinger avvist når stoppknappen var høy, og når knappen gikk lavt beveget ikke heisen seg før den fikk nye bestillinger. Dette oppfylte krav S6 og S7. Et annet sikkerhetsmoment var obstruksjonsbryteren. Dette ble testet ved å aktivere bryteren når:

1. Heisen befant seg i en etasje med døren lukket.
2. Heisen befant seg i en etasje med døren åpen.
3. Heisen befant seg mellom to etasjer.

Det ble da observert at obstruksjonbryteren ikke ville påvirke heisen så lenge døren var lukket, uansett om dette var mellom to etasjer eller i en etasje. Den ville derimot påvirke døren hvis den var i en etasje med døren åpen. Døren ville da forholde seg åpen så lenge bryteren var aktiv, og ikke lukke seg før 3 sekunder etter den ble slått av. Heisen beveget seg da heller ikke. Dette huket av R1 og D4. I alle de gjennomgåtte forsøkene bevgde aldri heisen seg utenfor heisrommet definert av første og fjerde etasje, som dermed bekreftet det endelige punktet S3. Alt i alt sjekket altså heisen av alle punktene som gjør at den kjører optimalt og trygt.

5 Diskusjon

Alle prosjekter inneholder feil, mangler eller ting som kan forbedres og vårt prosjekt er intet unntak. I denne seksjonen trekkes det frem de mest sentrale aspektene ved koden som kunne vært endret.

5.0.1 Globale variabler

Prosjektet bruker mange globale variabler. Globale variabler er generelt en dårlig idé i store prosjekter da flere ting kan ha samme navn som vil skape forvirring, og mangel på get og sett funksjoner gjør at variablene ikke er beskyttet fra feil bruk i c. Det ble valgt å bruke globale variabler da dette forenklet implementeringen noe, og prosjektet ikke er spesielt stort. Allikevel er det viktig å påpeke at om prosjektet skulle blitt videreført kunne de globale variablene ført til mye forvirring og problemer, og er dermed noe en burde forbedre.

5.0.2 Dynaminsk minnealokering og pekere

Lenkede lister er et fantastisk verktøy som gjør det enkelt å legge til og slette elementer fra en liste. De gjør det unødvendig å definere en fast størrelse på listen og å sette av minne som ikke nødvendigvis brukes. Problemet er at de krever en god forståelse for minneallokering og pekere og mangelen på disse kan føre til minnelekasjer og akksesering av feil minne. Bruken av lenkede lister forutsetter altså gode kunnskaper innen tema, om endring av køsystemet skulle være ønskelig. Koden kan derfor bli vanskeligere å vedlikeholde, men fordelen er at sorteringsalgoritmer blir overflødig, og det minsker unødvenig kopiering av minne. Fordelene kan neglisjeres om heisen styres av en rask datamaskin, men vil være viktig om styringselementer er en microkontroller. Om pekere og minneallokering er en fordel blir altså opp til bruksområdet, og er dermed noe en kan vurdere å endre.

5.0.3 Modulering

Et annet problem med koden er at noen av modulene er svært brede og noen funskjoner ikke ligger der de burde. For eksempel ligger update_last_floor() i queuesys, noe som ikke kommer tydelig frem i navnet. Dette gjør vedlikehold vanskligere. Det finnes for eksempel heller ikke en modul som hådterer lys da dette håderes når elementer legges til, og slettes fra, køen. For noen som ikke har skrevet koden vil det trolig være unaturlig å lete etter lysfeil i queuesys, selv om det fra et implementasjonsstandpunkt ga mening. For å øke lesbarheten og enkelheten ved vedlikehold ville en naturlig endring være å flytte flere funkjsoner og øke antall moduler.

6 Oppsumering

Heislab gikk ut på å implementere kode for styring av en heismodell. Den overordnede arkitekturen som ble valgt var en forenklet tilstandsmaskin. Koden ble delt opp i flere moduler som alle håndterte ulike aspekter ved heisen. Gjennom hele prosjektet ble modulene testet i hendhold til V-modellen og det ble også gjennomført grunndig testing i etterkant. Selv om heisen oppfylte de formelle kravene som ble stilt i oppgaven var det flere aspekter ved koden som kunne vært forbedret. Disse var ikke kritiske for virkemåten til heisen, men sentrale i forenklig av vedlikehold og lesbarhet av koden.

See you later elevator!