NTNU

Kunnskap for en bedre verden

Institutt for Teknisk Kybernetikk

TTK4235 - Tilpassede datasystemer

Heis prosjekt

Forfattere:
Sivarasa, S. og Øren, A. E.

September, 2020

Contents

List of Figures

1 Introduksjon

Til faget TTK4235 ble det utviklet en implementasjon for en heis i programmeringsspråket C. Denne rapporten skal det ta for seg hvilke moduler som ble utviklet, hvorfor modulene er satt sammen slik de er, implementasjonsvalg av modulene, testing av modulene og den fullstendige implementasjonen, samt en diskusjon rundt problemer og valg tatt under utviklingen. Altså tar rapporten i bruk samme modell som ble brukt under utviklingen av prosjektet; V-modellen.

2 Arkitekturvalg

2.1 Tilstandsmaskin

For den overordnede arkitekturen av applikasjonen ble det valgt å bruke en tilstandsmaskin med undertilstander. Grunnen til at en tilstandsmaskin ble valgt er at den gjør implementasjonen lettere og mer oversiktlig å lese gjennom, samt at det blir lettere å utvikle videre. Undertilstandene er også med på å fremme dette. Dette er videre diskutert i 3.3.

De forskjellige tilstandene samt deres interaksjon er vist i tilstandsdiagrammet i figur 1. Her har det blitt gjort et valg om å separere kjøre opp og kjøre ned i to tilstander; drive_up og drive_down henholdsvis. Dette ble gjort slik at den overordnede applikasjonen med tilstandsmaskinen ikke trenger å beholde retning i en variabel, men at den derimot kun trengte å sjekke modulen ansvarlig for dette før den endret tilstand. Dette medfører også et enklere grensesnitt til moduler, som køsystemet, siden en alltid vet hvilken retning heisen beveger seg i en gitt tilstand. I tillegg er det kun mulig å komme i tilstand door_open ved å først besøke tilstanden waiting. Dette er gjort med tanke på sikkerhet siden det er kun ønskelig at heisen skal åpne dørene sine når den står stille. Dette er garantert ettersom hver gang heisen går inn i tilstanden waiting stoppes motoren. Det er blir også gjort sjekker for å forsikre seg om at heisen er på en gyldig etasje før den kommer seg inn i door_open etasjen. Dette er for å hindre at den åpner dørene utenfor en gyldig etasje, men også slik at den åpner dørene når den er i en gyldig etasje og stoppknappen trykkes inn.

2.2 Valg av moduler

Funksjonene i modulene samt sammenhengen mellom dem er vist i klassediagramet i figur 2. Grunnen til at det er mange moduler er for å gjøre det lettere å videreutvikle applikasjon. Dette kommer av at modulene er ikke så tett tilkoblet som om de hadde vært færre, men større moduler. Dessuten er mindre moduler også lettere å videreutvikle siden de fremstår mer oversiktlige. Det ble valg å implementere følgende moduler: queue, linked_list, hardware, button, light, timer, floor og fsm. Disse modulene er beskrevet mer i detalj i seksjon 3.

I tillegg er et par av modulene lett å gjenbruke, for eksempel timer eller linked_list modulene. Med tanke på både gjenbrukbarhet og lettere videreutvikling har en høyere grad av funksjonpekere blitt brukt enn det som er nødvendig for å få en fungerende applikasjon. Dette skyldes av at funksjonspekere er satt til funksjoner utenfor modulen slik at det ikke trengs å inkludere modulen. Totalt sett er dette med på å redusere antall avhengigheter i applikasjonen.

2.3 Programflyt

Programflyt for en bevegende heis som både betjener interne og eksterne bestillinger er vist i sekvensdiagrammet i figur 3. Grunnen til at det er mange interaksjoner kommer av valget om å ha flere, men mindre moduler. Dette er diskutert i seksjonen over, seksjon 2.2. Som diskutert tidligere ble det valgt å bruke en del funksjonspekere. Det kommer da naturlig at en funksjon i en modul vil ta inn en funksjonspeker som et argument eller allerede ha denne lagret lokalt. Dette medfører at modulen kan kalle på funksjoner utenfra uten å måtte inkludere andre moduler. Et

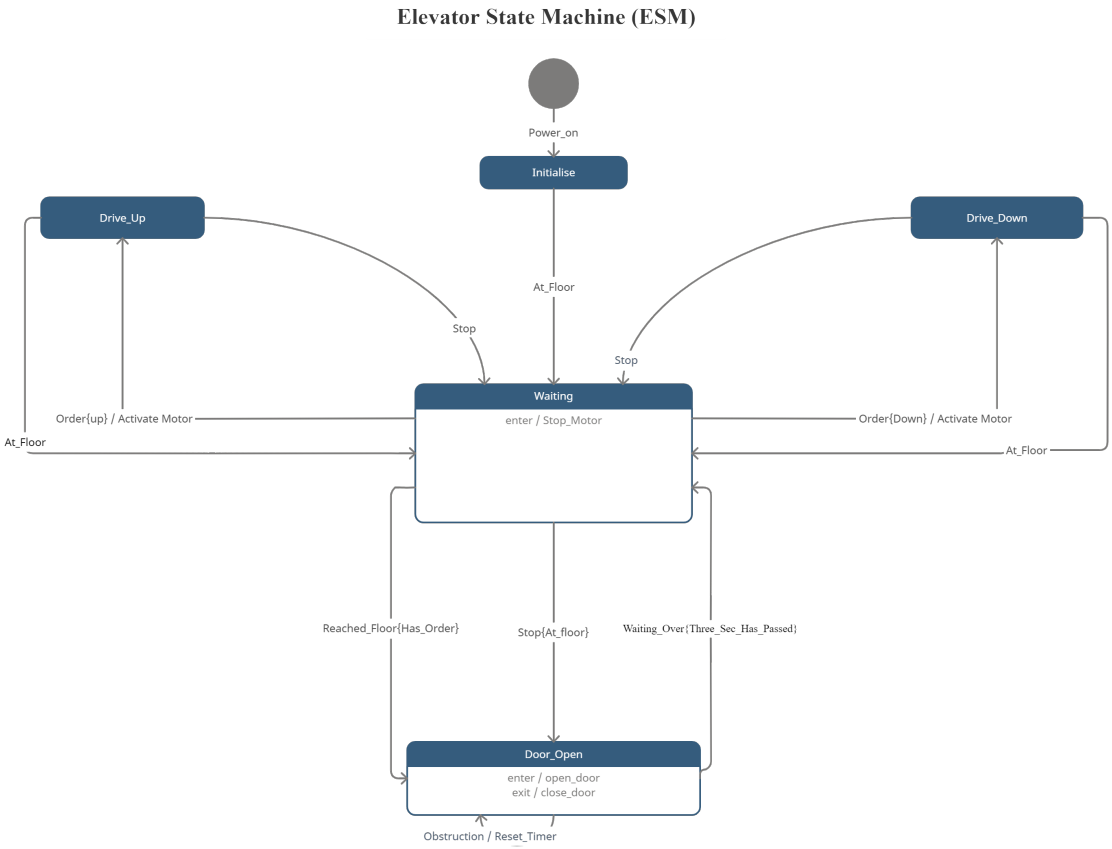

Figure 1: Tilstandsdiagram som viser de forskjellige tilstandene i tilstandsmaskinen samt transisjonene mellom dem.

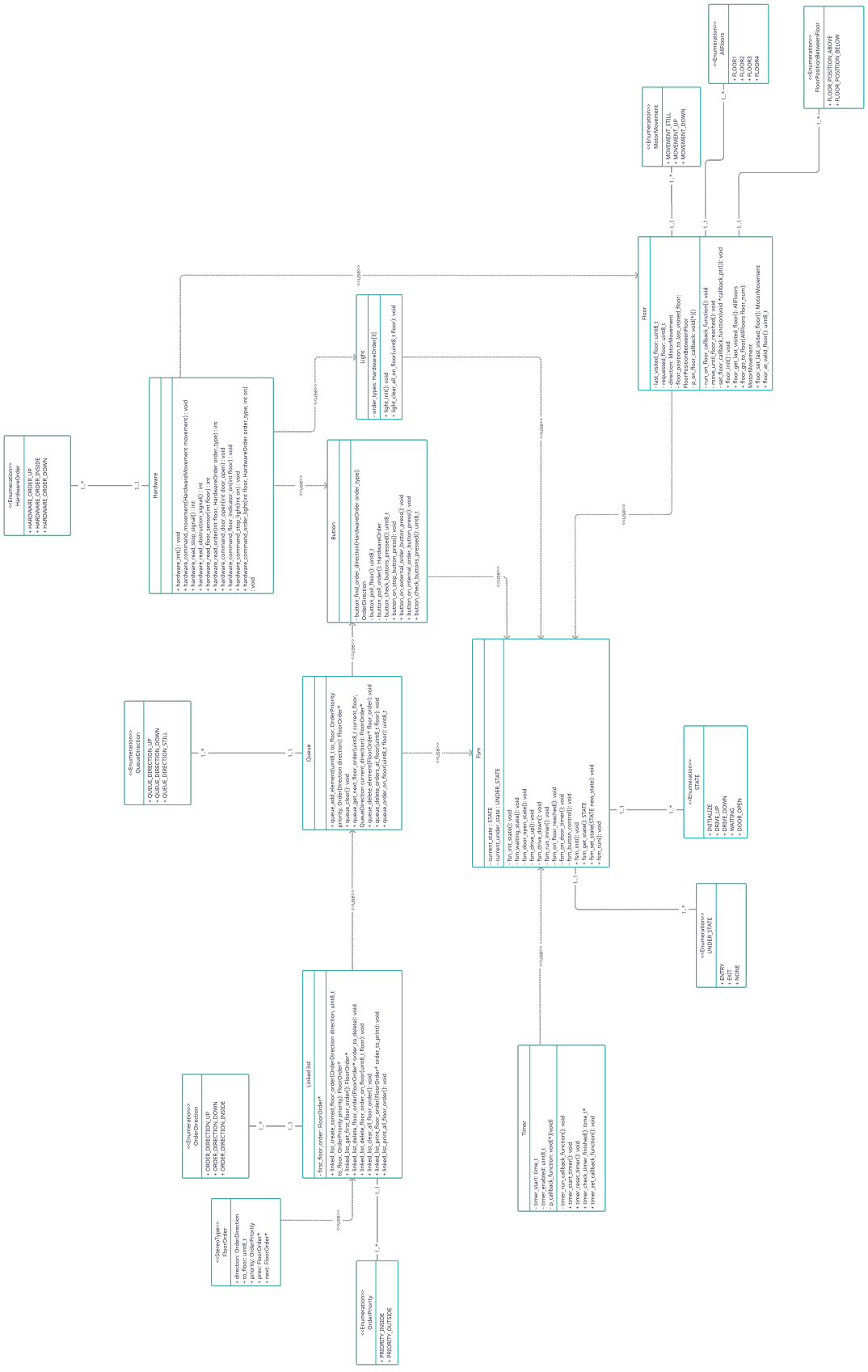

Figure 2: Klassediagram som viser modulene i applikasjonen samt deres avhengigheter

godt eksempel på dette; vist i sekvensdiagrammet, figur 3, er interaksjonen mellom fsm og timer. Det ble bestemt i planleggingsfasen at fsm ikke skulle bli inkludert av andre moduler enn selve main filen. Dette var for å unngå at moduler som blir inkludert av fsm også inkluderer fsm. For å kalle på funksjoner i fsm i timer modulen ble det derfor bestemt å ta inn fsm funksjonene som funksjonspekere i oppstarten av programmet. Dette medførte at timer ikke trengte å inkludere fsm, samt et enklere grensesnitt for timer modulen. En sidekommentar til timer-modulen er at for enkelhetsskyld har det bare blitt vist i dette sekvensdiagrammet én gang den kalles - mot slutten. Dette er gjort for å signalisere at den også kommer til å holde seg åpen i tre sekunder uten at det ligger en spesifikk ordre til grunn for det. I realiteten ville modulen også blitt kalt i det den ankommer første etasje, men tro til vanlig heisoppførsel som spesifisert i kravspesifikasjon Y1 vil tiden potensielt variere litt når en person går inn i heisen - de kan for eksempel være litt trege og resette timeren, eller trykke at døren skal lukke seg, en vanlig funksjon for heiser som ikke er representert i dette prosjektet. Altså har det blitt tatt et valg om å utelate denne biten fra sekvensdiagrammet.

3 Moduldesign

3.1 Linked list

For modulen linked_list.h har det blitt implementert en lenket liste. Grunnen til at lenket liste har blitt brukt er for å gjøre det enkelt å sortere listen når et nytt element blir lagt til. Når et nytt element blir lagt til vil det bli sjekket om det finnes en duplikat, hvis ikke vil den først sortere etter prioritet, så retning og deretter stigende etasje. Knapper innvendig har høyest prioritet og deretter kommer knappene utenpå med samme prioritet. Det vil si at hvis noen trykker på 1 etasje innenfra, 2 etasje innenfra og opp i 2 etasje og opp i 1 etasje vil elementene i den lenkede listen være på følgene rekkefølge; 1 etasje innenfra, 2 etasje innenfra, opp i 1 etasje og tilslutt opp i 2 etasje.

En fordel med å sortere den lenkede listen på denne måten er at neste etasje man skal gå til for en stillestående heis vil alltid være første element i den lenkede listen, derimot vil dette ikke stemme for en heis i bevegelse og en må derfor ha implementasjonen i queue.h, se seksjon 3.2.

Dessuten er linked list modulen fullstendig uavhengig av de andre modulene, det vil si at den ikke inkluderer noen andre selvlagde moduler. Dette gjør det enklere å utvikle, teste og gjenbruke denne modulen, siden utviklerene ikke trenger å tenke på andre interaksjoner enn kun den innad i modulen. Hvordan dette påvirker testing blir beskrivet mer i detalj i 4.1.

3.2 Queue

Queue er en modul som bygger på linked list modulen. De fleste funksjonene i queue modulen kaller direkte på en korresponderende funksjon i linked list modulen. Dette blir gjort for å redusere koblingen mellom linked list modulen og resten av applikasjonen. Dessuten er det en fordel å ikke trenge å også inkludere linked list modulen i andre deler av applikasjonen, men kun inkludere queue.

Majoritet av logikken i queue ligger i funksjonene queue_get_next_floor_order og queue_order_on_floor. Den siste nevnte funksjonen vil iterere gjennom den lenkede listen og sjekker om den gitte etasjen eksisterer i køen. Derimot er den førstnevnte funksjonen der mesteparten av logikken til queue modulen ligger. Denne funksjonen blir brukt til å bestemme hvilken etasje som blir neste å gå til. Så hvis heisen står stille vil den bare returnere første element i den lenkede listen, dette er grunnet av sortering av den lenkede listen beskrevet i 3.1. Hvis den derimot er i bevegelse vil den traversere gjennom den lenkede listen og finne bestillingen som er mest egnet både i henhold til etasjen heisen befinner seg i, og retningen. Den vil ignorere bestillinger i motsatt retning av bevegelsen.

Figure 3: Detaljert sekvensdiagram for en bestilling opp i 1 etasje, deretter en innvendig bestilling til 4 etasje. Dette diagrammet viser interaksjonen mellom de forskjellige modulene.

3.3 Finite State Machine

Hovedprogrammet kjører i fsm modulen, modulen for tilstandsmaskinen. Valget om at hovedprogrammet skulle kjøre her og ikke i main ble gjort med tanke på at dette ville medføre at man kunne abstrahere bort en del intern logikk. Blant annet gjorde det mulig å ha static variabler og funksjoner som ikke trenger å være tilgjengelig utenfor tilstandsmaskinen. Dermed ville dette valget medføre en kode som er mer brukervennlig.

Dessuten har det blitt implementert en tilstandsmaskin i hver av tilstandene; en hierarchical state machine. Disse tilstandsmaskinene, referert som under_state i koden, har kun tre standardiserte tilstander; Entry, Exit og None. Ved å ha slike under tilstander kan man ha en mye enklere hovedtilstandsmaskin som kun tar hensyn til tilstandene vist tidligere i denne rapporten, og disse undertilstandene kan brukes til å variere oppførselen basert på om en går inn, ut av eller forblir i hovedtilstanden. Dette medfører igjen mer lesbar kode og kode som er lettere å ekspandere, spesielt med tanke på å legge til nye tilstander. Det blir også lettere å modifisere eksisterende tilstander siden en kun trenger å ta hensyn til den undertilstanden man redigerer. Hvis man eksempelvis redigerer entry i waiting tilstanden trenger man verken å tenke på exit og none undertilstanden, så man kan kun utvikle for hva applikasjonen skal gjøre når den kommer inn i waiting tilstanden.

3.4 Timer

Det ble tatt et valg om å implementere en modul med kun en timer. Dette valget ble gjort siden det kun trengs én timer. En mulighet for å unngå dette er å ta inn en struct som parameter i alle funksjonene, dette vil derimot føre til tettere kobling mellom modulene, og var derfor ønsket å unngås.

Av samme grunn ble det bestemt å bruke en funksjonspeker som callbackfunksjon når timeren utløper, altså den funksjonen som blir kalt når tiden er ute. Det kan argumenteres for at valget av funksjonspekere gjør koden mindre lesbar og mer komplisert, i den forstand at funksjonspekere ikke er like intuitiv som vanlige funksjonskall. Derimot kan en også argumentere for at det blir lettere å bruke og ekspandere siden en ikke trenger å ta hensyn til en dyp kobling mellom timer og den nye modulen, men heller kun bruker timer_set_callback_function for å sette en callback, som blir automatisk kalt.

3.5 Floor

Floor modulen er modulen ansvarlig for håndering av bevegelse til en gitt etasje samt deteksjon av hvilken etasje heisen befinner seg i.

For å komme til en gyldig tilstand ble det valgt å bevege seg oppover til den når en etasje. Dette ble gjort siden det å bevege seg opp eller ned er klart den enkleste måten å løse denne oppgaven på.

For å orientere seg om sin lokasjon i henhold til etasje leseren den sensorene over og under sin posisjon. Dette funker fint, derimot ønskes det at heisen kan rette opp feil hvis den kommer utenfor disse områdene. Derfor er det i tillegg implementert en sjekk som går gjennom alle sensorer og retter opp heisen basert på dette. Det er for å unngå ulykker der heisen treffer taket eller bunnen.

3.6 Button

Button modulen er ansvarlig for logikken bak både å sjekke hvorvidt knapper er trykket inne, samt håndtere logikken ved knappetrykk.

Først sjekkes det hvorvidt en knapp er trykket inn, her prioriteres stoppknappen, deretter ordre og til slutt obstruksjon. Uten denne rekkefølgen ville enten det blitt tatt bestillinger mens stoppknappen var trykket inne, eller så ville obstruksjonsknappen også hindret nye bestillinger. Begge

disse situasjonene er i strid med kravspesifikasjonene. Utover dette er den generelle button logikken splittet opp mest mulig, med individuelle funksjoner for indre og ytre ordre, samt separate funksjoner for å bestemme retning og etasje. Her er det også innebygd logikk for å ta hånd om eventuelle ulovlige etasjer, eller "fantombestillinger", altså bestillinger som slår ut fra fsm, men som ikke eksisterer når knappene blir pollet på nytt igjen.

Button styrer altså køsystemet vårt. Den bestemmer når vi kan ta imot nye bestillinger, er ansvarlig for å kalle på funksjonene som legger inn nye bestillinger i køen, samt slette alle bestillinger når stoppknappen trykkes på. I tillegg må den kunne stoppe motoren. For enkelhets skyld er ikke button ansvarlig for logikken etter at obstruksjonsknappen aktiveres, bare for å gi beskjed om den er aktivert. Dette fører til færre inkluderinger, og en mindre kompleks modul totalt sett.

3.7 Light

Light modulen har to essensielle funksjonaliteter; slukke alle lys, eller slukke lys på en spesifikk etasje. Grunnen til at light modulen ikke omfatter det å tenne lys, er siden disse nye funksjonene kun vil kalle på funksjoner i hardware.h som allerede er importert og brukes på andre måter der denne nye light modulen skulle ha blitt brukt. Dette vil bare ført til mer inkludering, som vil gjøre det vanskeligere å lese og videreutvikle koden.

4 Testing

4.1 Testing av moduler uavhengig av hardware

Moduler som queue, linked-list og timer er uavhengig av hardware i den forstand at disse ikke trenger å inkludere hardware for å blir brukt eller testet. Dessuten er verken linked-list eller timer avhengig av andre selvutviklede moduler. Dette er en stor fordel når det kommer til testing av disse modulene. De har blitt laget slik at en utvikler uten tilgang til heis eller simulator kan teste dem, noe som reduserer kompleksitet og gjør det enklere å videreutvikle applikasjonen. Dette kom til fordel når forfatterne av rapporten utviklet modulene utenfor et miljø avhengig av heisen. Da ble *repl.it* brukt, som gjorde det enklere å teste og feilsøke.

Repl.it ble brukt for å gjøre enhetstester, testing av kun modulen. I *repl.it* ble forskjellige bestillinger lagt inn i køsystemet. Disse bestillingene ble printet til skjermen i riktig rekkefølge til en hver tid. Dette antyder at køsystemet fungerer som den skulle, samt at den ble godt testet.

Timeren ble også først testet i utviklermiljøet *repl.it*. Her ble det satt en enkel callbackfunksjon med printf inni og ulike tidsintervaller for å teste om timeren fungerte som den skulle. Den viste seg å fungere uavhengig av callback-funksjon og tidsintervall, noe som antyder at denne virker godt.

Deretter ble disse modulene separat testet på selve heisen og i simulatoren for å forsikre seg at de fungerte med resten av implementasjonen. Dette var en form for implementasjonstest i motsetning til det som ble gjort tidligere, enhetstester.

4.2 Implementasjonstesting av timer

I implementasjonstestingen av timeren kom det blant annet frem at dørene ville holde seg åpen i 3 sekunder selv etter en reset. Dette forsikrer at heisen fyller krav D1, D4 og timer delen av D3. I tillegg ble det testet at timeren ikke påvirker systemet når heisen er i bevegelse. Dette stemte så dermed er R1 også oppfylt.

4.3 Implementasjonstesting av køsystemet

Ved å teste bestillinger både i *repl.it*, simulatoren og i den ferdig implementerte heisen kan en forsikre at køsystemet alltid vil ta imot en bestilling. Dermed oppfyller den krav H1. Dessuten fører lenkede liste implementasjonen til at neste etasje er NULL hvis det ikke finnes noen bestilling, dermed vil heisen stå stille og H4 er også oppfylt. I tillegg viste køimplentasjonen å kunne ta bestillinger i samme retning som den beveger seg. Dette ble også godt testet i *repl.it*, simulatoren og tilslutt på heisimplementsjonen. Derfor er H2 også oppfylt. Dessuten hadde køsystemet en funksjon for å fjerne alle bestillinger på en etasje. Dette fungerte stabilt både under enhetstesten og under implementasjonstesten. Dermed er kravspesifikasjon H3 oppfylt.

4.4 Testing av betjening av bestillinger

Enhetstesten av floor ble gjort på selve heisen ved å legge til en rekke bestillinger manuelt, ved hjelp av kode, inn i køsystemet for å deretter observere oppførselen til heisen. Modulen greide å betjene alle bestillinger som ble lagt inn i køsystemet på denne måten. Dessuten ville den alltid bevege seg opp til en gyldig etasje i oppstartfase uavhengig om den startet på en etasje eller ikke. Dette antyder at kravspesifikasjonen O1 var oppfylt

For implementasjonstesten ble knapper implementert i en separat branch i git. Heisen ble testet med en rekke knappetrykk, og den oppførte seg som forventet uavhengig av antall knapper som ble trykket inn og rekkefølgen. Når heisen var i bevegelse ville kun lys fra forrige passerte etasje lyse, i samtråd med spesifikasjon L3, L4 og L5. Under denne testen kom det også frem at heisen ikke betjente bestillinger før den var i en gyldig tilstand. Dette er i tråd med kravspesifikasjon O2.

4.5 Testing av sikkerhet

Kravspesifikasjon S1 tilsier at heisen aldri skal åpne dørene sine i bevegelse. Det ble testet at tilstandsmaskinen ikke kunne komme seg til tilstanden door-open uten å først gå gjennom waiting tilstanden. Her er det en funksjon som stopper motoren. Dessuten er det ikke mulig å starte motoren mens tilstanden er door-open. Dette sikrer at heisen ikke kan åpne dørene i bevegelse. I tillegg er det ikke mulig å endre tilstand fra waiting til door-open uten at heisen befinner seg i en gyldig etasje. Dette ble testet grundig ved å se på hvilke tilstander som besøkes før døren åpnes. Dermed er det heller ikke mulig å åpne dørene utenfor en gyldig etasje, og S2 er oppfylt.

Stoppknappen ble også implementert og testet separat fra de andre knappene til tross for at de er i samme modul. Dette grunnes av at dette er en sentral del av sikkerheten og ble også derfor testet mer grundig. Som nevnt i 3.6 har den også høyere prioritet enn de andre knappene, så det vil si at man ikke kan bruke de andre knappene mens denne er i bruk, altså krav S6. Dessuten vil trykking av stoppknappen slette alle bestillingene i køsystemet. Det ble prøvd å teste stoppknappen mens heisen var stillestående, men også mens den bevegde seg i begge retninger. Det ble også testet om en kunne trykke stoppknappen gjentatte ganger. Alle disse fungerte som det skulle og dermed var S5og S4 oppfylt.

4.6 Testing av robusthet

For enhver kritisk embedded løsning som skal kjøre over en lenger periode er det viktig å teste robusthet. For å forsikre seg om heisen var robust og ikke ville oppføre seg uforventet ble heisen satt i tilfeldig valgte posisjon, deretter ble mange knapper trykket inn og testet. Det ble også testet tilfeller som man ikke forventer vanlige passasjerer av en heis ville gjøre, som for eksempel trykke stoppknappen og bestille en tilfeldig valgt etasje gjentatte ganger. Her var heisens oppførsel som forventet, noe som antyder at den er robust. For å forsikre om at den også håndterte tilfeller utviklerne ikke hadde tenkt på ble blant annet andre studenter og læringsassistenter spurt om å teste heis med hensikt å "prøve å ødelegge den". Før FAT testen ble dette forsøket repetert, men

de andre testerne av systemet greide ikke å fremkalle uforventet oppførsel. Dette antyder at heisen kan kalles robust.

5 Diskusjon

5.1 Modulstørrelser

Ved generell gjennomførelse av prosjektet var det viktig finne ut av hvor lite det var hensiktsmessig å gjøre hver enkelt modul. Dette prosjektet benytter seg av en serie med mindre moduler. Mindre moduler tilsvarer at hver enkelt modul har lettere for å være selvstendig, som igjen tilsvarer lettere testing, samt bedre gjenbruk ved en senere anledning. I tillegg var det ønskelig å holde moduler separerte med unntak av i fsm for både lesbarhetens skyld, samt å unngå overlapp. Gjennom mindre moduler oppnår man enklere klare skiller mellom modulene i hva deres tiltenkte jobb er, dermed blir det lettere å feilsøke senere, samt finne igjen spesifikke funksjoner.

På den andre siden krever store moduler færre statiske funksjoner, og kan dermed være mindre jobb å utarbeide. For enkelte programmer vil det også være meningsfylt å ikke splitte det opp for mye ettersom man da ikke klarer å finne dedikerte oppgaver til hver modul. Større moduler vil da tilfredsstille de samme kravene som mindre moduler, men på sin måte være mer lesbare, og håndterbare. Dersom modulstørrelsen hadde drastisk endret seg ville det betydd en helt annerledes arkitektur. For eksempel ville ikke lenger fsm nødvendigvis vært knytepunktet. I tillegg ville pekerlogikken vært unødvendig ettersom én av hovedgrunnene til å arbeide med pekere er at det er lett å sende variabler som inputs på tvers av moduler.

For dette prosjektet var konklusjonen at mange små moduler var mer hensiktsmessig. Hver modul blir da som nevnt tydelig adskilt fra de andre, og gjør alt fra testing, vedlikehold og gjenbruksbarhet lettere i fremtiden.

5.2 Linked list versus matrise

Et av de første store designvalgene som ble gjort var å basere prosjektet på en implementasjon av lenkede lister, dynamisk allokert minne og pekere, fremfor en mer standard matriseløsning. Beslutningen ble tatt på bakgrunn av tanken om at lenkede lister har en naturlig progresjon, de er laget for å binde sammen et ledd til en annen, altså vil det alltid være enkelt å komme seg videre til den neste orderen, så sant man har en god sorteringsalgoritme. Altså vil store deler av logikken forenkles etter å ha løst dette problemet. Desverre viste det seg at løsningen var overkomplisert. Gevinsten oppnådd ved å alltid kunne ha neste ordre klar gikk tapt når det sammenlignes med den ekstra tiden feilsøkingen tok. I tillegg åpner dynamisk allokering av minne opp for flere run-time errors dersom en ikke er påpasselig med å frigjøre minnet - en problematikk som igjen kan koste tid for eksempel ved introduksjon av feil som double free.

Når dette er sagt har også designet positive sider. Som tidligere diskutert i 3 gir pekere en mulighet til flere statiske funksjoner, og dermed kan modulene splittes mer opp til å være uavhengige av hverandre. Resultatet her er som nevnt at koden lettere kan gjenbrukes og vedlikeholdes ved senere prosjekter. Generelt er også koden mer lesbar ettersom den ikke krever masse nøstede løkker, men heller baserer seg på pekerlogikk, dermed kan man ved hjelp av gode variabel og funksjonsnavn gi gode beskrivelser av funksjonaliteten. En bivirkning av lett gjenbrukbarhet er at det også er enkelt å bytte ut logikken i fremtiden til fordel for en enklere løsning. Som vist i både 3 og 2 er det kun queue som bruker linked list, og generelt er det lav kobling mellom funksjonaliteten til begge disse modulene og resten av programmet. Altså er det fullt mulig å endre innmaten uten å forårsake store skader på programmet som en helhet.

Totalt sett var trolig linked list ikke den optimale løsningen, den skaper en barriere for å forstå koden - som vist i at uten kommentarer lagt inn i implementasjonen av modulene ville det vært vanskelig å forstå dem. Om kunnskapen som kom ut av prosjektet hadde vært besittet i starten av prosjektet hadde rapportskriverne altså sannsynligvis bestemt seg for den enklere løsningen

med matriser. Likevel er det verdt å nevne at læringsutbyttet ved å utforske denne metoden var særdeles god, så fra et pedagogisk synspunkt gjorde det oppgaven mer spennende og læringsrik.

5.3 Funksjonspekere

Under designet av programmet ble det også besluttet at det var nyttig med funksjonspekere. Funksjonspekere viste seg å være nyttige grunnet behovet for callbackfunksjoner som blir kalt eksempelvis når heisen kommer til en ny etasje, eller når timeren er over. Disse blir da statiske funksjoner som man kan endre utifra behovet, altså slipper man å for eksempel lage en switchcase basert på hvor i programmet man er. I stedet har man muligheten til å sette funksjonspekeren til en ny funksjon i forkant av kallet, for deretter å ha en generell funksjon til å kalle på callback-funksjonene. Resultatet er at hver enkelt modul ikke er avhengig av like mange andre moduler. Dermed kan man si at funksjonspekerne ikke endrer funksjonaliteten på en større måte, men heller tillater simplifisering av logikken innad i de forskjellige modulene.

6 Konklusjon

Totalt sett er altså hele prosjektet knyttet opp mot tilstandmaskinen, som er samlingspunktet for alle modulene. Alle moduler er valgt med hensikt å takle et par spesifikke punkter i kravspesifikasjonene, og jobber mest mulig uavhengig fra hverandre. Dette er gjort for å dra nytte av fordeler som; enklere vedlikehold, muligheter for kodegjenbruk og lesbarhet. Modulene er testet via sekvenser på *repl.it*, simulatoren, samt funksjonstesting på heisen selv, og tilfredsstiller alle kravspesifikasjoner. Til slutt er det verdt å nevne at enkelte designvalg, slik som pekerbasert logikk, funksjonspekere og nøye planlagte modulstørrelser har vært gunstig for gjennomføringen av prosjektet, og har bidratt til å gjøre koden lettere å sette seg inn i. Andre valg, slik som bruk av lenkede lister i stedet for matriser, var ikke optimale, men bidro til et bedre læringsutbytte. Med dette i bakhodet konkluderer vi med at prosjektet er godt gjennomført, men har rom for forbedring til senere iterasjoner.