Heisprosjekt i TTK4235

Hannah Hansen, Mina Haver

Heisann sveisanntidssalen

Introduksjon

Målet med dette prosjektet har vært å lage en velfungerende heismodell. Til dette er programmeringsspråket C benyttet, og utviklingen er forsøkt i tråd med V-modellen.

1. Overordnet arkitektur

Arkitekturen vi har valgt å implementere består av fire moduler; en dørmodul, en styringsenhet, en modul som håndterer bestillinger og en heismodul. I tillegg kommer de to utdelte modulene som leser og skriver til sensorene og styrer hardware, men vi vil ikke fokusere på disse her.

Dørmodulen åpner og lukker døra, bestillingsmodulen leser nye bestillinger, sletter bestillinger og sjekker om bestillinger finnes, og heismodulen kjører heisen. Mens disse tre modulene tar hånd om hver sin funksjonalitet, knytter kontrollenheten modulene sammen og sørger for at modulene sammen opererer i henhold til kravspesifikasjonene.

1.1. Klassediagram

I klassediagrammet i figur 1 kan den overordnede arkitekturen observeres. Her inngår funksjonene og medlemsvariablene og relasjonene mellom modulene.

1.2. Sekvensdiagram

For å illustrere hvordan modulene opererer sammen for å betjene en bestilling, har vi laget et sekvensdiagram. Et eksempel på dette kan sees i figur 2, og illustrerer når i prosessen de ulike modulene samarbeider for å betjene denne bestillingen.

1.3. Tilstandsdiagram

Heisen er implementert som en endelig tilstandsmaskin der heisen alltid vil befinne seg i en veldefinert tilstand. Disse tilstandene og betingelsene som kan endre tilstanden, er beskrevet i figur 3. Hvis heisen ved oppstart befinner seg utenfor en definert tilstand, vil den initialisere seg ved å bevege seg til den kommer seg til en definert tilstand.

2. Moduldesign

Oversikt over bestillinger og prioriteringskø er implementert i queue_handler-modulen. Bestillingene representeres i en todimensjonalt liste ("array") med én rad pr etasje og tre kollonner som representerer bestillinger opp, ned og inni heisen. Verdien 1 tilsvarer en bestilling og 0 svarer til ingen bestilling. Denne implementasjon gjør at man enkelt kan finne ut om det er en bestilling av en spesifikk type i en gitt etasje. Ved oppstart av programmet trenger man bare opprette én slik tabell, og dermed kan man enkelt kan sette inn og fjerne bestillinger uten å bekymre seg for minnelekkasje.

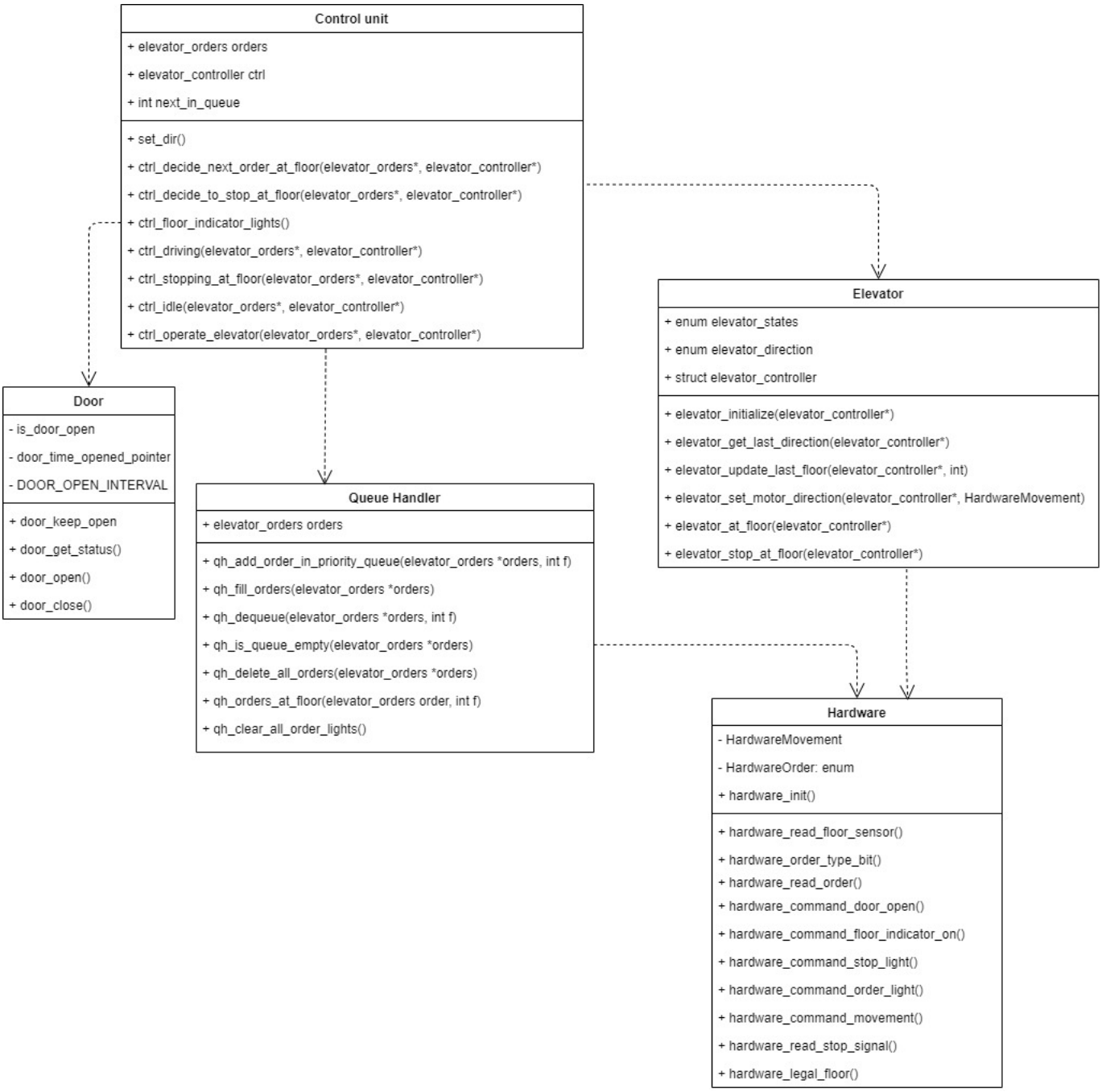
Prioriteringskøen er en liste med lengde lik antall etasjer. En tom plass i køen har verdien -1. Vi antar at heisens etasjer i hardware-modulen alltid representeres med tallene 0 til N, hvor N er antall etasjer, så køen kan lett generaliseres til flere etasjer. Ved en ny bestilling vil den tilsvarende etasjen settes inn på første ledige plass i køen, om den ikke allerede står i køen. Ettersom kravspesifikasjonene sier at alle personer går av eller på når heisen stopper i en etasje, er det ikke nødvendig å legge inn én etasje flere ganger i køen. Når heisen stopper i en etasje, fjernes etasjen i prioriteringskøen, eventuelle bestillinger lengre bak i køen flyttes fram én plass og siste element settes til -1. Dette gjør at alle bestillinger vil bli håndtert. En heis trenger også bare én slik kø, så denne implementasjonen gjør det enkelt å søke gjennom køen og sette inn og fjerne elementer uten fare for minnelekkasje.

Ettersom oversikt over bestillinger og prioriteringskø henger tett sammen med tanke på innsetting, sletting og valg av ordre som skal utføres, har vi valgt å lage en datatype kalt elevator_orders som inneholder disse to listene. Denne er vist i listing 1. Dette gjør programmet mer oversiktlig, og det trengs færre parametere i funksjonene som behandler bestillinger og kø.

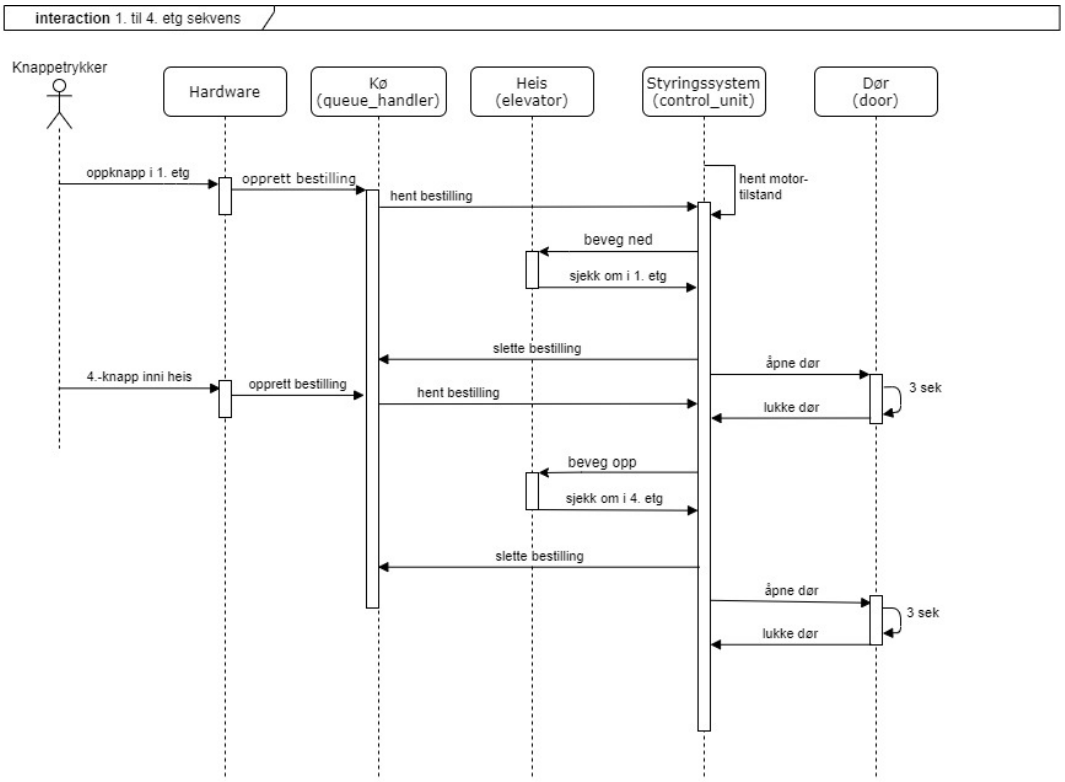
```
1 typedef struct elevator_orders{
2     int order_table[HARDWARE_NUMBER_OF_FLOORS
      ][3];
3     int priority_queue[HARDWARE_NUMBER_OF_FLOORS
      ]; }elevator_orders;
```

Listing 1: Struct for å representere bestillinger

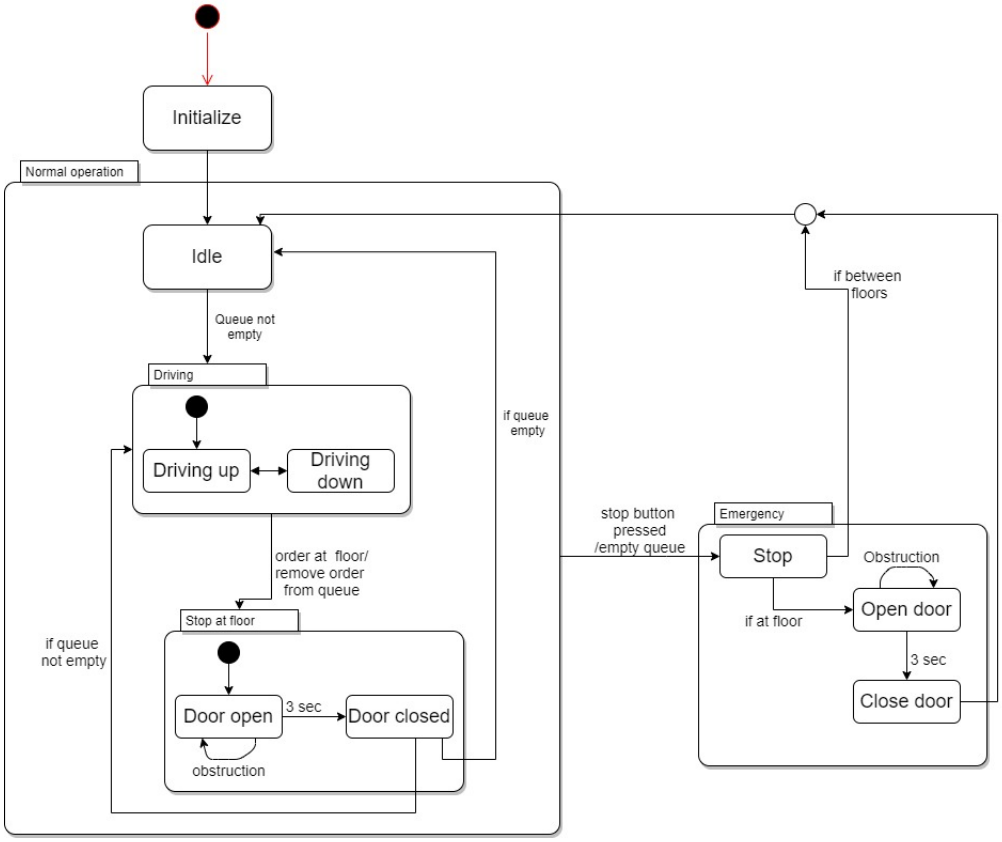
Dørmodulen inneholder funksjoner for å åpne og lukke dør, samt en variabel som holder styr på om døren er åpen eller lukket og en timer-variabel som lagrer tiden døren sist ble åpnet. Disse har vi valgt å gjøre private. I denne modulen finnes også en funksjon som bestemmer om døren skal



Figur 1: Klassediagram. Viser funksjonene inneholdt i de ulike modulene, samt hvordan de henger sammen.



Figur 2: Sekvensdiagram for en bestilling fra første til fjerde etasje. Dette viser hvordan de fire modulene, samt Hardware, jobber sammen for å gjennomføre denne bestillingen



Figur 3: Tilstandsdiagrammet viser de ulike tilstandene heissystemet kan befinne seg i, og hvilke betingelser som kan endre heisens tilstand.

holdes åpen på bakgrunn av timeren og obstruksjonssignalet. Dette er det eneste stedet i koden obstruksjonssignalet sjekkes slik heisen ikke påvirkes av obstruksjonsbryteren med mindre døren er åpen og denne funksjonen kalles.

Heismodulen inneholder en datatype elevator_controller som beskriver heisens tilstand og bevegelse. Denne er vist i listing 2. Den inneholder variabler med den sist kjente etasjen heisen ble registrert i, den sist kjente retningen den kjørte i, samt en enum som beskriver hvilken av de fire tilstandene i tilstandsmaskinen heisen befinner seg i. Dette gjør at man kan samle relevant informasjon om heisens oppførsel i én variabel. Funksjonen i denne modulen initialiserer heisen, styrer motoren, oppdaterer medlemsvariablene i elevator_controller, samt sjekker om heisen befinner seg i en etasje.

```
typedef struct elevator_controller{
  elevator_direction last_dir;
  int last_floor;
  elevator_state state;
}elevator_controller;
```

Listing 2: Datatype som inneholder heisens posisjon, retning og tilstand.

Kontrollenheten knytter sammen disse tre modulene. Med informasjon om heisens tilstand, posisjon og bevegelse i en elevator_controller-datatype, kan kontrollenheten styre heisen på bakgrunn av bestillinger fra etasje- og heispanel i en elevator_orders-datatype. Kontrollenheten er også en tilstandsmaskin. Hvordan heisen skal oppføre seg, er avhengig av hvilken av tilstand heisen befinner seg i, og dermed er det også hensiktsmessig å implementere kontrollenheten som en tilstandsmaskin. De mulige tilstandene vi har valgt er definert i enumen elevator_states vist i listing 3.

```
typedef enum {DRIVING_STATE,
              STOP_AT_FLOOR_STATE,
              EMERGENCY_STOP_STATE,
              IDLE_STATE} elevator_state;
```

Listing 3: Enum for å representere hesiens mulige tilstander

Heisen er i tilstanden DRIVING_STATE når den kjører mellom etasjer og utfører bestillinger. Når heisen når en etasje, må kontrollenheten undersøke om heisens kjøreretning samsvarer med eventuelle bestillinger, eller om denne etasjen er den første i køen. Om dette stemmer, vil heisen stoppe og gå over til neste tilstand, STOP_AT_FLOOR_STATE. Her vil riktige bestillinger slettes og døren åpnes og lukkes. Om det finnes flere bestillinger i køen, vil heisen gå tilbake til DRIVING_STATE. Hvis køen er tom, endres heisens tilstand til IDLE_STATE. Heisen vil da vente i etasjen med døren lukket til den får en ny bestilling. Hvis stoppknappen trykkes, vil tilstanden endres til EMERGENCY_STOP_STATE. Heisen stopper momentant, og alle bestillinger slettes. Heisen går så over i IDLE_STATE, hvor den er klar til å håndtere nye bestillinger.

3. Testing

Modulene ble først testet enkeltvis for å se om de fungerte alene, før de ble satt sammen til et større sammenhengende program.
Dette ble så testet i randtilfeller og med utgangspunkt i kravspesifikasjonene i oppgaveteksten. Ved avvik fra kravspesifikasjonene ble disse feilsøkt, og modulene gått over for å finne de logiske og syntaktiske feil.

I testingen av spesifikasjonene i forbindelse med oppstart av heisen ble programmet kjørt mens heisen stod stille mellom to etasjer. Ved oppstart flyttet heisen seg da til etasjen over, og **O1** ble dermed ansett oppfylt. Ved forsøk på å gjøre bestillinger i denne oppstartfasen observerte vi heisen fremdeles stod i ro etter å ha funnet definert tilstand, og følgelig er **O2** ivaretatt. **O3** ansees ivaretatt ved at det ikke er tatt hensyn til at heisen kan befinne seg under første eller over fjerde etasje ved oppstart.

Under testingen av bestillingshåndteringen var utgangspunktet logikken bak priotiteringskøen og virkemåten til ordretabellen. Disse tar inn bestillinger fortløpende. Som følge av at alle ordrer i en gitt etasje blir ansett betjente når heisen stopper i etasjen og at heisen vil fortsette å betjene bestillinger til køen er tom, anser vi det umulig at heisen kommer i en situasjon hvor en bestilling ikke blir betjent. Dette ble også testet ved å gi heisen en stor strøm av bestillinger, i kombinasjon med å observere at bestillingene faktisk blir tatt. Krav **H1** er altså oppfylt. Når heisen ble ferdig med å betjene bestillingene den hadde fått, observerte vi at den forble i ro i den siste etasjen den hadde bestillinger til. Dette, i kombinasjon med en gjennomgående stillstand ved fravær av bestillinger ellers i testingen, bekrefter **H4**.

For å se hvorvidt heisen betjener bestillinger utenfor heisrommet i motsatt retning av bestillingsretningen, prøvde vi å bestille heisen nedover mens den kjørte oppover. Denne bestillingen ble ikke utført før etter at den originale bestillingen var utført. Observasjon av heisen etter gjentatt trykking på mange knapper både i og utenfor heisen resulterer etter endt trykking fremdeles i maks fire stopp før heisen igjen står stille i påvente av nye bestillinger. Det er derfor rimelig å anta at alle bestillingene i en etasje betjenes når heisen stopper i gitt etasje.

Testingen av bestillings- og etasjelysene bestod hovedsakelig i å observere de respektive lysene i kombinasjon med trykking på korrelerende knapper. Etasjelyset er også observert til å lyse så lenge gitt etasje var den foregående etasjen heisen befant seg i. Etter observasjon og testing diktert av kravspesifikasjonen ble det konkludert med at alle spesifikasjonene kyttet til lysene er ivaretatt.

Døra ansees åpen når dør åpen-lyset lyser. Gjennomgående observeres dør åpen-lyset når heisen ankommer en

etasje den skal stoppe i, og punkt **D1** ansees ivaretatt. Dørlyset slukkes også etter tre sekundene ved fravær av bestillinger, og står dermed stille og venter med døra igjen. Dette oppfyller **D2**. Med heisen inaktiv, ble stoppknappen trykket. Dette resulterte i åpne dører så lenge stoppknappen var holdt inne, og videre tre sekunder etter det, i samsvar med **D3**. Dette ble forsøkt i flere etasjer, og fungerte der og. Med åpen dør, observerte vi at døra fortsetter å holde seg åpen så lenge obstruksjonsbryteren er aktiv. Etter dette holdt døra seg åpen i videre tre sekunder. Dette ble forsøkt i flere etasjer, og fungerte der og. Dermed kan vi anta at **D 4** stemmer.

Under testing av sikkerheten til heisen, gikk store deler av testingen ut på å se hvorvidt de forskjellige kravene var tilfredsstilt i randtilfellene beskrevet i kravspesifikasjonen. Vi har ikke klart å åpne døra mens heisen er i bevegelse, da dette bare er mulig gjennom funksjoner i dørmodulen som bare kalles i tilstander hvor heisen står i ro. Vi har heller ikke observert at dør åpen-lyset har lyst når heisen har vært i bevegelse. Dermed har vi konkludert med at **S1** og **S2** er ivaretatt. Under testingen av heisen ble det heller ikke observert noen tilfeller hvor heisen forsøkte å kjøre utenfor endeetasjene. Det eksisterer ingen kommandoer som vil klare å sende heisen utenfor første og fjerde etasje, og vi synes det dermed er rimelig å anta at dette er umulig. Heisen er forsøkt stoppet med stoppknapp i alle etasjer og i de tre strekkene mellom etasjene både i retning oppover og nedover. Etter at heisen ble stoppet av stoppknappen i disse tilfellene, ble heisen stående i ro etter at stoppknappen ble sluppet. I tillegg ble kø og ordretabell printet, og her så vi at begge disse var tomme. Det er derfor rimelig å anta at køen slettes simultant med at stoppknappen trykkes inn. Dette tilfredsstiller **S4** og **S5**. For å sjekke hvorvidt heisen kan motta bestillinger mens stoppknappen er trykket inn, forsøkte vi å holde den inne samtidig som vi trykket på knapper både på heispanelet og på etasjepanelet. Når stoppknappen da ble sluppet opp, forble heisen i ro, og vi antar dermed at den har ignorert forsøkene våre på å gi den bestillinger. Utskrift av kø og ordretabell etter at stoppknappen var sluppet bekrefter også dette. Vi anser følgelig **S6** og **S7** oppfylt.

Ved forsøk på å aktivere obstruksjonsbryteren ved vanlig heiskjøring ble dette ignorert av heisen, så **R1** er oppfylt. Det har heller ikke vært behov for å starte programmet på nytt under testing av den ferdigstilte heisen, og vi synes derfor det er rimelig å anta at krav **R2** er oppfylt. Ved oppsart av heisen vil den kjøre til den kommer til en etasje og oppdatere last_floor-variabelen. Denne blir oppdatert kontinuerlig så lenge heisen er i bevegelse. Dette sørger for at heisen alltid vet hvor den etter initialiseringen, i tråd med **R3**. Dette har blitt testet ved å hente ut verdien av denne variablelen mens heisen er i bevegelse og etter stoppknappen har blitt aktivert.

Generelt oppfører heisen seg som en normal heis. På bakgrunn av testene beskrevet over, mener vi at **Y1** i stor grad er oppfylt. Vi innser likevel at kjøreruten heisen velger for å utføre ordrene i ikke er optimal. Dette utdypes i neste seksjon.

Gjennomgående i testingen ser vi heisoppførsel i tråd med kravspesifikajonene. Med forankring i dette vil vi si at heisen er testet grundig nok til å påstå at den oppfyller kravspesifikasjonene.

4. Diskusjon

4.1. Prioritering

Heisens prioritering av rekkefølgen bestillinger blir utført i er ikke optimal. Køsystemet er basert på når bestillingene opprettes og ikke hvilken kjørerute som er den mest effektive. Med en heis med de fire etasjene vi har operert med her, fungerer prioriteringssystemet rimelig. Hvis man derimot skulle oppskalert heisen til å operere over enda flere etasjer ville forsinkelser knyttet til knotete prioriteringssystem økt kraftig. Med en heis som går over enda flere etasjer, og dermed flere kombinasjoner av etasjer som kan bestilles til, vil det bli nødvendig med en bedre prioriteringsalgoritme, som i større grad tar hensyn til hvilke retninger bestillingene som tas, er i. Da heisen i denne oppgaven kun går over fire etasjer, har vi sagt oss fornøyde med vår enkle prioriteringsalgoritme, da vi tolker denne i tråd med de oppgitte kravspesifikasjonene.

4.2. Arbeidsprosessen

Sammenlignet med tidligere prosjekter var dette en mer omfattende oppgave. Det var derfor svært nyttig å kunne ta utgangspunkt i V-modellen, samt UML-diagrammene vi lagde i forkant av selve programmeringen. Som følge av mye tid brukt på modultesting og godt gjennomarbeidede moduler, var det gjennomgående lite behov for debugging og feilsøking da modulene ble satt sammen til et fullstendig program. Prosessen beskrevet i V-modellen har fungert godt og supplert med gode verktøy for å skape velfungerende moduler.

5. Konklusjon

Arbeidet med dette prosjektet har ført fram en heis som funker og opererer i tråd med oppgitte kravspesifikasjoner. Arkitekturen består av en modul som drifer heisen, en modul som håndterer bestillinger og kø, en modul som håndterer døra og en kontrollmodul som betjener heisen. Rekkefølgen heisen velger å utføre bestillinger er ikke optimal, men fungerer med få etasjer.