Heisprosjekt i TTK4235

Nils Reed
Selsebil Radi
Gruppe 58

Vår 2020

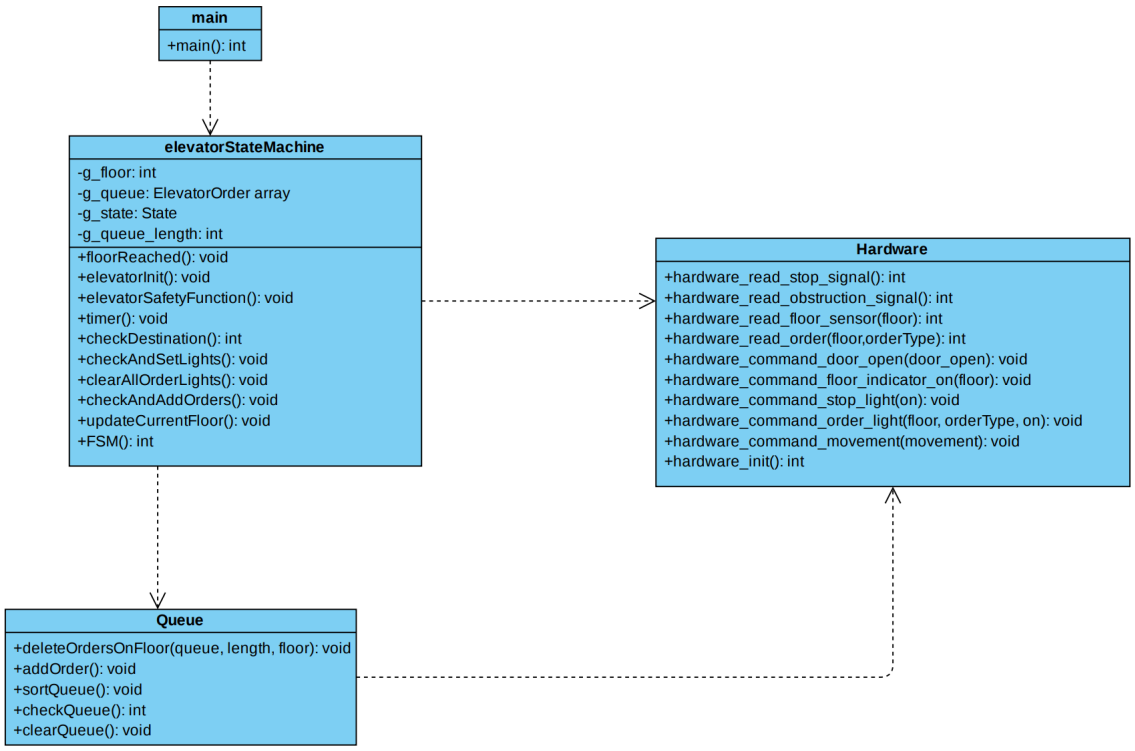
Figur 1: Ungdomar i heis på elektro B (1920, colorized).

1 Introduksjon

Denne rapporten tar for seg heisprosjektet i Tilpassede datasystemer. Heisprosjekt er programmert i C. Rapporten gjenspeilar kvifor me har tatt dei vala me har tatt i implementasjonen av heisen, og det me ser i retrospekt at me kunne gjort betre.

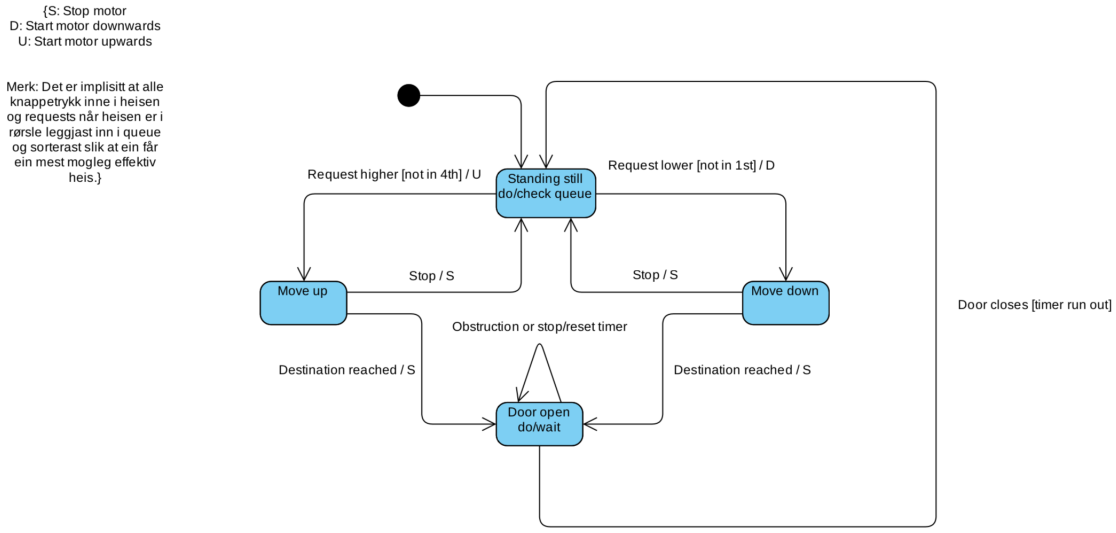
2 Overordna arkitektur

2.1 Klassediagram



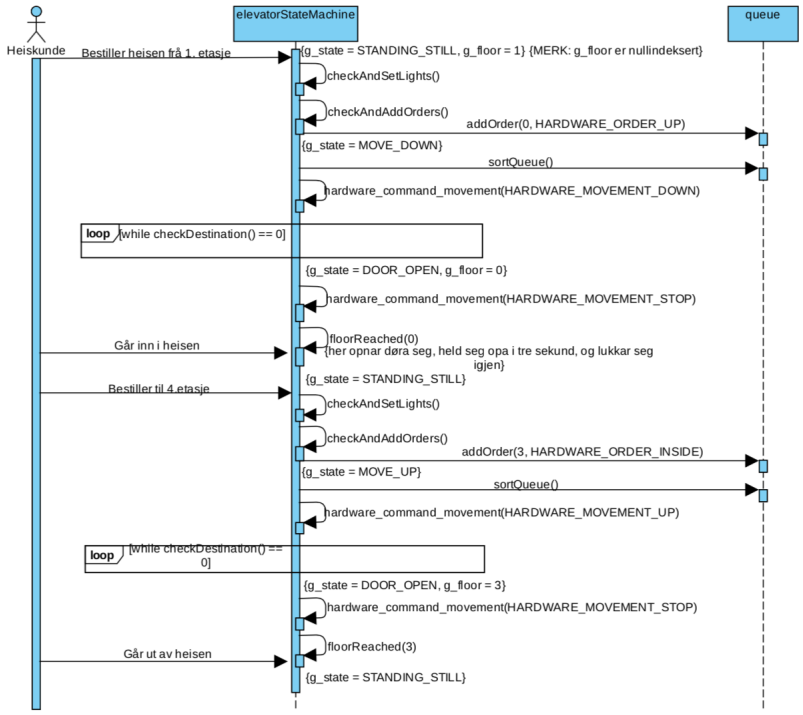
Figur 2: Klassediagram for heissystemet.

2.2 Tilstandsdiagram



Figur 3: Eit generelt tilstandsdiagram for logikk implementert i tilstandsmaskinen i elevatorStateMachine.

2.3 Sekvensdiagram



Figur 4: Sekvensdiagram for ein bestemt funksjonalitet av systemet oppgitt i oppgåveteksten.

2.4 Val av arkitektur

Me har valt å ha to tilstandar MOVE_UP og MOVE_DOWN i systemet vårt, slik som vist i 3. Dette er for å sleppe å lagre retninga til heisen i ein eigen variabel, og det gjer også slik at logikken i systemet ikkje er avhengig av å vite førre og neste posisjon, og i liten grad avhengig av å vite førre tilstand. I tillegg så oppfører systemet seg ulikt basert på kva for ein retning den er på veg. Utover dette har me òg tilstandane DOOR_OPEN og STANDING_STILL. Systemet si tilstand og tidlegare tilstand lagrar me i dei globale variablane g_state og g_prev_state.

I tillegg har me òg oppretta ein eigen modul main. Dette har me gjort for at dokumentasjonen gjennom Doxygen skal gje meir mening. Me har valt å bruke globale variablar for å gjere systemet meir generelt. Dette gjer det enkelt å leggje til fleire etasjer og gjere køen lenger ved behov. Dette gjer òg at me kun treng éin kø-peikar, noko som gjer minnebruken til programmet og dermed faren for segmenteringsfeil mindre.

3 Moduldesign

Me har implementert ein timer med time.h-biblioteket i elevatorStateMachine og ikkje som ein eigen modul slik at me unngår å ha mange sterkt avhengige modular. Ein eigen modul for timer ville medført ein modul avhengig av elevatorStateMachine og hardware.

Funksjonalitet for å manipulere køen har me lagt i queue-modulen. I tillegg ligg funksjonen checkQueue() der. Dette fordi me meinar dette er ei logisk separering av funksjonalitet frå elevatorStateMachine.

Vidare har me lagt alt av funksjoner som krev polling av sensorar og som gjer tilstandsmaskinfunksjonaliteten fullstendig i elevatorStateMachine.

4 Testing

4.1 Einingstesting

For queue-modulen har me testa funksjonane deleteOrdersOnFloor(), addOrder() og sortQueue() kvar for seg.

Testinga av addOrder() vart gjort på følgjande måte:
For å ikkje vere avhengig av å trykkje på knappar for å sjå om funksjonen fungerer, tok funksjonen inn ein tilfeldig kø, kor ein sjølv kunne fylle arrayet ved hjelp av bibliotekfunksjonen scanf(). Desse vart lagt til i første tomme element i arrayet, og me vart sikra at funksjonen addOrder fungerte ved å skrive ut køen med ein printf()-setning på slutten av køyringa.

Dersom ein køyrde alle funksjonane samtidig, vart arrayet frå addOrder() sortert av sortQueue() slik at det var stigande rekkjefølje på vei opp, og synkande rekkefølje på vei nedover. Dei ulike retningene vart testa med to ulike array.

Testinga av deleteOrdersOnFloor() vart gjort ved å generere tilfeldige køar av elevatorOrders, printe desse til konsoll, deretter køyre deleteOrdersOnFloor med dei

og ein tilfeldig etasje, og printe dei til konsoll på nytt. Dette vart gjort fleire gonger med ulike antal bestillingar som skulle slettast for å sikre at funksjonen fungerte.

For einingstesting av elevatorStateMachine-modulen vart funksjonen timer() testa.

For å teste timer(), vart lyset satt på i main og deretter skrudd av etter bruk av timer for å sjå om den heldt seg på i ønsket tid. Det vart gjort slik:

```
hardware_command_door_open(1);
timer(3);
hardware_command_door_open(0);
```

Me jamførte med stoppeklokke.

Resten av funksjonane er blitt testa i integrasjonstestinga.

4.2 Integrasjonstesting

For å teste queue-modulen med elevatorStateMachine-modulen, har me gjort følgjande:

Før heisen er i stand til å ta i mot bestillingar, køyrer den til ein bestemt tilstand, som me har valt til å vere etasje 1. Det er ingenting som skjer dersom du trykkjer på bestillingsknappane før den er initialisert i etasje 1.

Dersom me trykkjer på ei bestilling i same etasje som heisen befinn seg i, vil dørene opne seg, og vere opne i tre sekund. Om det kjem ei bestilling til medan døra er opa, vil dørene vere opne i tre sekund frå den nye bestillinga vart plassert. Altså vil timeren resettast.

For å teste køen køyrde me systemet i GDB. Me køyrde systemet som normalt, men hadde eit display av g_queue. Dersom heisen står i etasje 1, og heisen får ei bestilling i etasje 4, vil den køyre oppover mot etasje 4. Hvis den på vei opp får ei bestilling til etasje 3, vil g_queue sorterast på nytt, og etasje 3 vil leggjast fyrst i køen dersom det var ei bestilling oppover eller frå innsida av heisen. Dersom det var ei bestilling nedover frå etasje 3, leggjast den etter etasje 4. Heisen vil altså betjene etasje 4 fyrst.

Viss heisen vert stoppa mellom to etasjer, vil heisen vite kva for ein etasje den står i mellom, sjølv om me ikkje har ein definert etasje for mellomstega. Me brukar i stade eintydige kombinasjonar av variabelen stopped og variabelen g_floor. Det er greit å merke seg at desse ikkje er unike for kvart mellomsteg mellom to etasjer. Derfor vil heisen alltid vite kor den er dersom stoppknappen vert trykt, og heisen kan fungere som normalt videre. Sidan heile køen vert sletta, har det ikkje noko å seie kva for ein retning heisen beveget seg i, da den vil prioritere fyrste bestilling uavhengig av tidlegare state.

Elles har me òg testa “normal heisframferd”ved å trykkja inn ulike kombinasjonar av bestillingar når heisen står i ulike tilstandar og jamført med det ein vil forvente av ein “normal heis”.

5 Diskusjon

Sjølv om systemet, etter vår fatteevne, tilfredsstiller kravspesifikasjonen, er det nokre ting me kunne ha implementert annleis.

Me har mellom anna valt å bruke bubble sort for å sortere køen til heisen. Dette har me valt fordi det er ein algoritme som er enkel å implementere og forstå. Bubble sort er også ein god løysing sidan arrayet/køen ikkje har mange element.

Ei anna ulempe er at bubble sort bruker mange kodelinjer i systemet vårt grunna dei to ulike tilstandane MOVE_UP og MOVE_DOWN, sidan køen sorterast ulikt avhengig av kva for ein tilstand heisen er i. Dette kan minke lesbarheita til koden sjølv om implementasjonen er relativt enkel. Det krev tid å setje seg inn i, og ei betre løysing hadde vore å bruke ein metode som ikkje krev så mange kodelinjer. Blant annet kunne me hatt ein rekursiv bubble sort. Det vil ha samme funksjonalitet, men krevje langt færre kodelinjer. Eventuelt kunne me og unngått å sortere køen ved å implementere all logikk i addOrder(), eller laga ein funksjon som vel kva for ein bestilling som er neste utifrå posisjonen og retninga til heisen.

Me kunne òg implementert funksjonaliteten til korleis heisen tolkar kvar den står når den vert stoppa mellom etasjer annleis. Sidan me har eintydige kombinasjonar for heisen til å tolke kvar den står, og ikkje unike kombinasjonar, vil dette minke lesbarheita til koden.

Ei annan endring som kunne gjort lesbarheita betre, ville vore å hatt fleire uavhengige modular. Dette kunne vert betre gjennomtenkt under planleggingsfasen. Modulen elevatorStateMachine tar for seg mange funksjoner som kunne vært implementert i ein eigen modul. Fordelen med det hadde vore at heisen kunne betre lesbarheit, og det ville vore enklare å bytte implementasjon av funksjoner i kvar modul. Heile systemet hadde vore meir oversynleg.

I systemet vårt har me valt å definere ElevatorOrder-en {floor = -1, orderType = HARDWARE_ORDER_INSIDE} som ein none-type. Dette slik at me kan allokere ein kø med statisk lengd i starten av programmet og bruke den som global variabel. Me er klare over at det går ut over lesbarheita til koden. I byrjinga av implementasjonsfasen la me til eit alternativ NONE i HardwareOrder-enumeratoren i hardware.h, men etter kvart støytte me på problem med resten av den utdelte koden, og tok det difor vekk. Det at orderType er HARDWARE_ORDER_INSIDE er berre ein arbitrær konvensjon som er vald og brukt konsekvent gjennom heile programmet.

Me har konsekvent brukt camelCase til å skrive ulike funksjonsnamn for å ty-

deleggjere skilnaden på eigenskriven og utdelt kode. Me har i tillegg skrive globale variablar med snake_case for å skilje dei frå lokale som er skrivne med camelCase. Dette kan gjere at koden ser litt ustrukturert og rotete ut, og for å forbetre dette, kunne me ha skrive alt ved hjelp av å kun bruke snake_case.

Det at me lagrar både noverande og førre tilstand til systemet, kjem av at det i siste liten vart oppdaga ein bug som ikkje vart tatt omsyn til i hovuddelen av implementasjonen. Bugen var at køen ikkje vart sortert når heisen stod stille, altså i tilstandene DOOR_OPEN og STANDING_STILL. Som ei siste liten-løysing, bestemte me oss for å kompensere for denne feilen ved å leggje inn ein if-setning i addOrder(), og å lagre førre tilstand. Dette fiksa problemet, men gjer igjen koden litt mindre leseleg då me har innført ein ny variabel, som me ikkje bruker i veldig stor grad.

Argumentasjonen for moduldesignet vårt er litt tynn. Dette kjem av at me tenkte litt for lite gjennom korleis me faktisk skulle designe modulane i praksis under planleggjing. Difor har ein del blitt som det har blittlitt spontant ut frå vår kodingsprogresjon. Om me heller hadde planlagt betre, og tenkt meir gjennom verkemåten til modulane og deira plass i systemet tidleg i prosjektet, ville dette stått fram som ein sterkare del av vårt system.

Me kunne også unngått å ha fleire avhengige modular ved at me ikkje brukte globale variablar. Dei globale variablane gjer at me er nødt til å inkludere elevatorStateMachine.h i queue.c, noko me kunne unngått. Hadde vi i staden laga funksjonar som berre tek inn argument og ikkje bruker dei globale variablane aktivt, kunne me latt vere i å ha maksimal avhengignad i systemet.

Alt i alt tykkjer me at me har programmert ein rimeleg god heis, med tanke på funksjon. Sjølve implementasjonen tykkjer me kunne vore litt ryddigare og betre planlagt tidlegare.