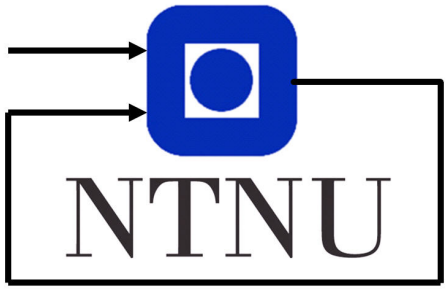

Institutt for teknisk kybernetikk

TTK4235 - Tilpassede datasystemer

Heisraport - Gruppe 2

Av :
Martin Hillestad, Henrik Østvik

Mars, 2025

Table of Contents

1 Introduksjon

I denne rapporten presenterer vi gjennomføringen av vårt heisprosjekt, der vi har utviklet en heisstyring i C. Vi har planlagt prosjektet ved hjelp av UML-diagrammer og fulgt V-modellen for å sikre en strukturert utviklingsprosess. Først beskriver vi planleggingsfasen, inkludert designvalg og heisens ønskede oppførsel. Deretter går vi gjennom implementeringen og de ulike modulene vi utviklet. Til slutt reflekterer vi over erfaringene våre, inkludert bruken av kunstig intelligens i prosjektet.

2 Forarbeid

2.1 Tilstandsdiagram

Det første vi gjorde var å analysere produktkravene for å forstå hva oppgaven innebar. Deretter designet vi et tilstandsdiagram Figur 1 basert på disse kravene.

Når systemet skrus på, returnerer heisen alltid til første etasje før den går over i tilstanden "vent på signal". Derfra kan den enten gå til "kjør oppover" eller "kjør nedover"-modulen. Prinsippet bak dette er at heisen skal fortsette i samme retning helt til siste bestilling i den retningen er behandlet, før den eventuelt bytter retning. For å ivareta sikkerheten la vi til en listener for stoppknappen, slik at heisen kan stanses under hele kjøretiden.

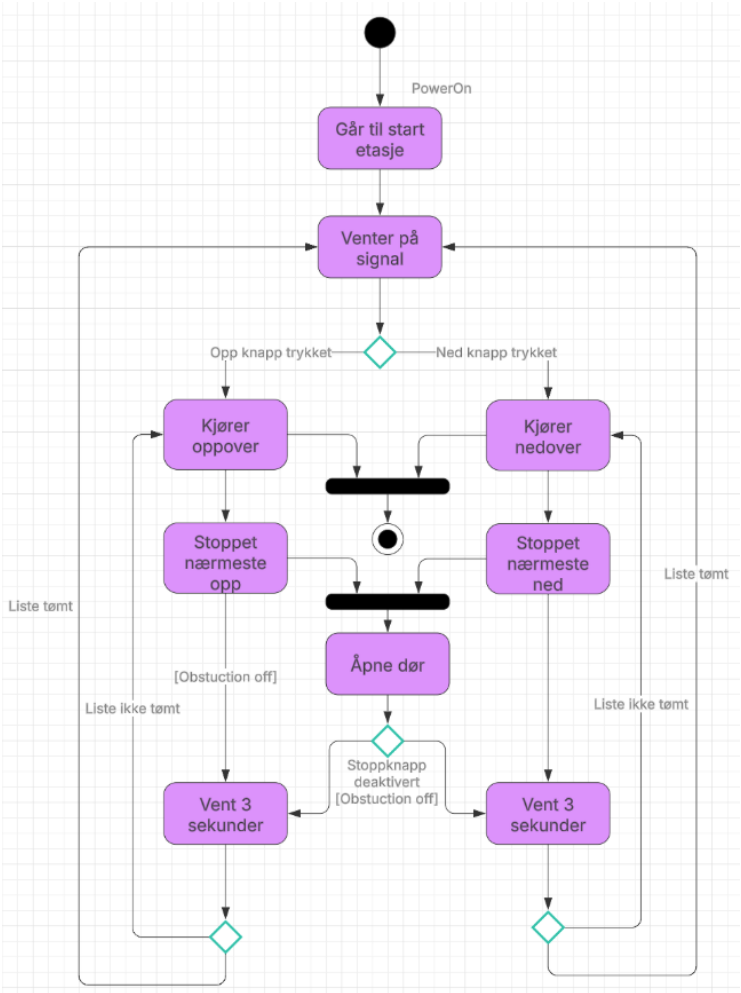

Figure 1: Tilstandsdiagrammet av heisen

Når heisen ankommer ønsket etasje, starter den med å åpne dørene. Deretter sjekker systemet om stoppknappen er deaktivert og at det ikke er noen obstruksjoner til stede. Dersom begge disse kriteriene er oppfylt, startes en timer på tre sekunder før dørene lukkes og heisen fortsetter i samme retning. Når alle bestillinger i en retning er behandlet, eller heisen har nådd sin endepunkt, bytter den retning og gjennomfører samme prosess på nytt.

2.2 Klassediagram

Etter at tilstandsdiagrammet var designet, begynte vi med klassediagrammene Figur 2. Her forsøkte vi å definere hvilke klasser og funksjoner som var nødvendige for systemet.

Den første klassen vi utviklet var StartPhase. Denne klassen sikrer at heisen følger oppstartsekvensen definert i oppgaven. StartPhase kommuniserer med Elevator-klassen, der all nødvendig informasjon om heisens status er samlet. Dette inkluderer posisjon, retning og eventuelle obstruksjoner. Ved å samle denne informasjonen på ett sted, blir koden mer oversiktlig og strukturert.

Videre var det viktig å implementere sikkerhetssystemer for heisen. Derfor la vi til Security-klassen, som håndterer informasjon om hindringer og om stoppknappen er aktivert. Klassen inneholder også to funksjoner som aktiveres ved eventuelle sikkerhetsbrudd.

For at heisen skal fungere optimalt, må brukeren kunne se hvor heisen befinner seg og hvilke knapper som er trykket inn. Dette løste vi ved å legge til LightPanel-klassen, som sørger for at alle lysindikatorer er oppdatert til enhver tid.

Det siste elementet for å bygge en fullstendig heis var et køsystem. Dette løste vi med Queue-klassen, som inneholder to lister: én for oppovergående bestillinger og én for nedovergående. Queue kommuniserer med CabinButton- og FloorButton-klassene, som registrerer om knapper i henholdsvis kabinen eller etasjene er trykket inn, og legger disse bestillingene til i riktig liste.

Resultatet er et klassediagram der hovedklassen Elevator fungerer som systemets kjerne. Fra denne klassen forgrener LightPanel, Security, StartPhase og hele køsystemet seg, noe som gir en logisk og strukturert oppbygning av systemet.

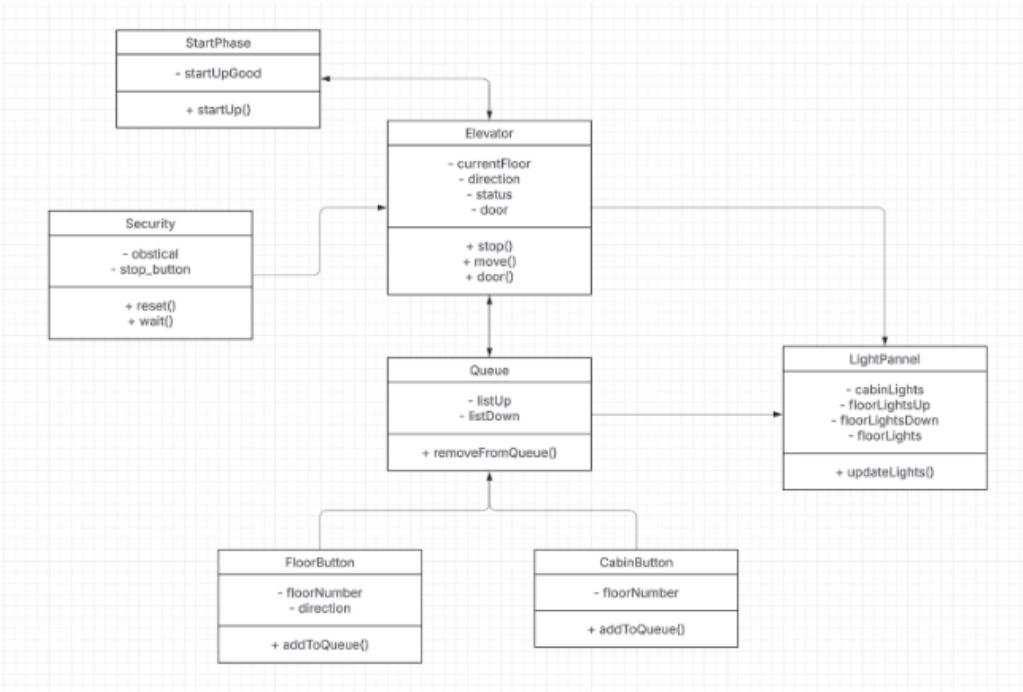

Figure 2: Klassediagrammet av heisen

2.3 Tanker bak opprinnelig design

Gjennom utviklingen av disse to diagrammene konkluderte vi med at heisen skulle baseres på en tilstandsmaskin. Dette innebærer at heisen opererer med to hovedtilstander: opp og ned.

Når heisen er i opp-tilstand, behandler den alle forespørsler som befinner seg over dens nåværende posisjon. Dersom det ikke finnes flere forespørsler over heisen, endrer den tilstand til ned og begynner å søke etter forespørsler under seg. Hvis det heller ikke er noen forespørsler nedenfor, vil heisen forbli i sin nåværende posisjon og kontinuerlig søke etter nye bestillinger både over og under, ved å bytte tilstand fortløpende.

3 Underveis

3.1 Bruk av UML og V-modellen

UML-diagrammene som ble utarbeidet i forarbeidet var til stor hjelp for å definere hvor vi skulle starte. Vi begynte med klassediagrammet, som vi brukte til å opprette de ulike header- og C-filene. Dette sikret en god struktur i koden allerede fra start. Deretter gikk vi videre til å utvikle heisens logikk. For å implementere denne brukte vi tilstandsdiagrammet aktivt, ved å følge diagrammet fra start til slutt og implementere logikken i de klassene som var mest hensiktsmessige basert på tilstandsdiagrammet.

Under implementasjonen gjennomførte vi en rekke tester for å verifisere at hvert aspekt av systemet fungerte som forventet. Dette avvek fra V-modellens standard, men vi opplevde det som en naturlig og nødvendig fremgangsmåte for å unngå tidkrevende feilsøking senere i prosessen. Vi var også bevisste på å ikke være for bundet til UML-diagrammene vi hadde laget. Dersom vi fant bedre løsninger underveis, valgte vi heller disse. Hovedmålet med diagrammene var å sikre at gruppen hadde en felles forståelse av heisens oppførsel og at alle vesentlige aspekter var gjennomtenkt før programmeringen startet.

I sluttfasen hadde verken UML-diagrammene eller V-modellen stor påvirkningskraft. På dette tidspunktet hadde hele gruppen en grundig forståelse av hvordan heisen skulle fungere. Vi hadde en delvis fungerende, men ikke optimal heis. Derfor fokuserte vi i denne fasen på å identifisere og rette opp logiske feil i heisens oppførsel.

3.2 Endringer fra opprinnelig design

Selv om vi i stor grad fulgte det opprinnelige designet, tok det tid før vi landet på den endelige løsningen. Innledningsvis hadde vi en svært kompleks tilnærming til køsystemet, hvor heisen ikke skulle plukke opp passasjerer som skulle i motsatt retning, selv om den stoppet i etasjen på grunn av kabinkøen. Denne tilnærmingen førte imidlertid til komplisert logikk i koden og en uforutsigbar oppførsel av heisen.

Etter hvert kom vi tilbake til punkt H3 i FAT-testen Natlandsmyr and Jacobssen 2025: "Når heisen først stopper i en etasje, skal det antas at alle som venter i etasjen går på, og at alle som skal av i etasjen går av. Dermed skal alle ordre i etasjen være regnet som ekspedert." På bakgrunn av dette valgte vi å slukke bestillingsknappene i begge retninger når heisen stoppet i en etasje. Selv om denne løsningen ikke nødvendigvis er optimal i et større system med flere etasjer eller heiser, vurderte vi den som tilstrekkelig for dette prosjektet, hvor heisen kun opererte i fire etasjer og var alene i systemet.

3.3 Bruken av KI

I løpet av prosjektet har vi benyttet KI til flere formål. Den mest omfattende bruken var knyttet til opprettelse og forståelse av Git, slik at vi kunne lagre og administrere koden på en trygg og effektiv måte. Vi stilte hovedsakelig spørsmål om branching, push og merge, noe som ga oss en solid forståelse av GitHub og hvordan vi best kan bruke det i fremtidige programmeringsprosjekter.

Vi brukte også KI til generell kodeforståelse underveis i prosjektet. Spesielt fikk vi hjelp med bruken av pekere og informasjon om pause-funksjonalitet i C. Vi opplever at denne bruken av KI har vært svært nyttig for å oppnå rask og god forståelse av deler av C-programmering som var ukjente for oss. Samtidig mener vi at det ikke har gått på bekostning av læringen vår, men snarere bidratt til en mer effektiv læringsprosess.

4 Kode

Gjennom denne delen av rapporten vil vi forklare koden, del for del. Vi går gjennom hver enkelt C-fil og beskriver hvilke funksjoner de inneholder og hva disse gjør.

4.1 elevator.c

I elevator.c finner vi funksjonen floorFinderUp, som leter etter forespørsler i etasjene over heisens nåværende posisjon. På tilsvarende måte søker floorFinderDown etter forespørsler i etasjene under. Filen inneholder også funksjonen emptyList, som fjerner nåværende etasje fra alle bestillingslister og slukker tilhørende lys. Til slutt finner vi setCurrentFloorLight, som tenner indikatoren for den etasjen heisen befinner seg i.

4.2 queue.c

I queue.c finner vi funksjonen listenForInput, som kontinuerlig sjekker om noen har trykket på en knapp, enten i en etasje eller inne i heiskabinen. Dersom en knapp aktiveres, kalles en av funksjonene addToUpList, addToDownList eller addToInsideList, som legger til den aktuelle etasjen i henholdsvis opp-listen, ned-listen eller kabin-listen.

4.3 security.c

security.c har ansvar for sikkerhetsfunksjonalitet i heisen. Her finnes blant annet funksjonen resetElevator, som starter opp systemet på nytt ved å kjøre oppstartssekvensen. wait-funksjonen får systemet til å vente i et gitt antall millisekunder. deleteOrders tømmer alle bestillingslister og slukker alle lys. Til slutt har vi stopElevator, som aktiveres når stoppknappen trykkes. Denne funksjonen kaller deleteOrders, tenner stopplyset og får heisen til å vente til stoppknappen slippes. Dersom heisen befinner seg i en etasje når knappen trykkes, åpnes døren automatisk.

4.4 startPhase.c

Denne filen inneholder én funksjon: startup. Funksjonen starter opp systemet, slukker alle aktive lys og sende heisen ned til første etasje, i tråd med kravene til oppstart.

5 Erfaringer

5.1 Effekten av UML og V-modellen

Når vi ser tilbake på prosjektet, bidro UML-diagrammene til en mer strukturert og effektiv gjennomføring. De gjorde det mulig for oss å forstå hvordan heisen var forventet å operere før vi begynte å kode. I tillegg hjalp diagrammene oss med å kartlegge ulike scenarier en heis kan møte, samt hvordan den bør respondere. Dette ga hele gruppen en felles forståelse av heisens logikk og hvordan den skulle implementeres.

Da vi startet med koding, oppdaget vi tidlig at V-modellen ikke var den mest optimale arbeidsmetoden for dette prosjektet. V-modellen er utviklet for prosjekter der en prototype bygges før testing kan begynne. I vårt tilfelle hadde vi en fungerende heis fra starten av, noe som gjorde det mulig å teste heisens oppførsel kontinuerlig. Dermed ble det unødvendig å vente med testing til all logikk var implementert, ettersom vi kunne verifisere hver enkelt klasse eller funksjon fortløpende.

I ettertid ser vi at prosjektet kunne ha blitt mer effektivt dersom vi hadde utarbeidet en rekke tester allerede i forarbeidet. Vi forsøkte å identifisere unike situasjoner som kunne skape utfordringer for heisens oppførsel, men valgte å løse dem ved å endre UML-diagrammene. Gjentatte endringer førte imidlertid til at noen tidligere løste situasjoner ble forstyrret. Vi innser nå at vi ble for opptatt av detaljer, noe som gjorde at vi til tider mistet fokuset på heisens grunnleggende logikk. Ved å ha utarbeidet vår egen FAT-test ville vi alltid hatt en referanse, som kunne hjelpe oss med å sikre at vi ikke overså eller glemte scenarier vi allerede hadde identifisert.

5.2 Robusthet, skalarbarhet og vedlikehold

UML-diagrammene har hatt en betydelig innvirkning på systemets robusthet. De har gitt gruppen en felles forståelse av hvordan heisen skal operere, noe som har gjort det enklere å identifisere logiske feil og unike scenarier systemet kan støte på. Dette har bidratt til å styrke heisens robusthet, da vi kan være sikre på at den håndterer de fleste situasjoner effektivt.

Klassediagrammet har forbedret systemets vedlikeholdsmuligheter ved å gi en tydelig struktur og oversikt over koden. Dette gjør det enklere for en tredjepart å sette seg inn i systemet, enten for å rette feil eller for å implementere oppgraderinger.

Når det gjelder skalerbarhet, har verken UML-diagrammene eller V-modellen hatt noen vesentlig påvirkning. En økning i antall heiser eller etasjer anses ikke som ønskelig, da dette vil føre til en betydelig reduksjon i systemets effektivitet.

En mer systematisk anvendelse av V-modellen underveis ville ha forbedret både vedlikeholdsmuligheter og robusthet. Ved å følge modellens struktur kunne man ha sikret at hver utviklingsfase ble etterfulgt av en tilsvarende testfase, der eventuelle feil ble identifisert og rettet opp tidlig. Dette ville særlig vært fordelaktig for større prosjekter, der grundig verifikasjon og validering er avgjørende for kvaliteten på sluttproduktet.

References

Natlandsmyr, Tord and Terje H. Jacobssen (2025). *main.pdf - Oppgavebeskrivelse*. GitHub repository. URL: https://github.com/ITK-TTK4235/lab_2/releases/tag/1.1.3/main.pdf.

Denne oppgaven er skrevet av forfatterene, men deler av teksten har KUN blitt språkvasket ved bruk av kunstig intelligens for å forbedre lesbarhet og grammatikk. Innholdet ikke produsert av KI.