NTNU

Kunnskap for en bedre verden

Institutt for teknisk kybernetikk

TTK4235 - Tilpassede Datasystemer

Heis-prosjekt

Gruppe: 78

Av : Alper Yarenbasi

Mars, 2025

Table of Contents

1 Innledning

Formålet med heisprosjektet er å anvende kunnskap fra tidligere øvinger i TTK4235 for å utvikle et styresystem for en fysisk heismodell med fire etasjer. Systemet er implementert i programmeringsspråket C og styrer en heis via en Arduino, inkludert motorbevegelser, etasjeindikatorer, knappelamper, og dørlogikk. Prosjektet simulerer en realistisk heis ved hjelp av sensorer og aktuatorer på Sanntidssalen.

For å sikre en strukturert utviklingsprosess har vi brukt den pragmatiske V-modellen, som guider arbeidet fra kravanalyse til design, implementering og testing (Factory Acceptance Test - FAT). UML-diagrammer (state- og klasse-diagrammer) er brukt for å visualisere systemets arkitektur og oppførsel før kodingen startet. Dette har hjulpet oss med å unngå uforutsette problemer og sikre at systemet oppfyller kravene i *main.pdf*

Rapporten dokumenterer hele prosessen, inkludert krav, design, kodeimplementering, og en refleksjon over arbeidet. Til tross for tidspress og utfordringer ble et fungerende system levert, som oppfyller de fleste kravene i FAT-testen.

2 Krav og spesifikasjoner

I *main.pdf* er det listet en rekke krav til heissystemets funksjon og robusthet. Disse kravene ligger til grunn for både design, implementasjon og testing (inkludert en slutt-test, *Factory Acceptance Test* – FAT). Kravene kan deles inn i følgende hovedkategorier:

1. **Oppstart (O1–O3):**
   - Systemet skal alltid komme i en definert tilstand ved oppstart.
   - Bestillinger skal ignoreres frem til heisen kjenner sin etasje.
2. **Håndtering av bestillinger (H1–H5):**
   - Alle bestillinger skal tas, og ingen skal «henge igjen».
   - Heisen skal kjøre i riktig retning, og bestillinger skal ekspederes i ankomstetasjen.
3. **Bestillingslys og etasjelys (L1–L6):**
   - Knappelys tennes når bestilling registreres, og slukkes når bestillingen er utført.
   - Etasjeindikator lyser for gjeldende (eller sist kjente) etasje.
4. **Sikkerhet (S1–S7):**
   - Heisen skal stå stille dersom stoppknappen trykkes inn, og bestillinger skal slettes.
   - Heisen skal ikke kjøre utenfor definert område (1–4 etasje).
5. **Robusthet (R1–R3):**
   - Systemet skal tåle «uventet» bruk av obstruksjon og stoppknapp uten å henge seg opp.
   - Etter første kalibrering skal heisen ikke måtte kalibreres på nytt.

Disse kravene er styrende for hvordan vi har designet og implementert heissystemet.

Oppsummert danner kravspesifikasjonen rammen for hele utviklingsprosessen. Vi starter med kravene i design- og arkitekturfasen (gjennom UML-diagrammer), og bygger deretter implementasjonen slik at hver enkelt funksjon, modul og test skal reflektere og oppfylle de listede kravene. Til slutt bekrefter *FAT*-testen i hvilken grad vi faktisk har møtt kravene.

3 Systemdesign og Arkitektur (UML)

Denne seksjonen presenterer systemdesignet og arkitekturen for heissystemet, med fokus på bruken av pragmatiske V-modellen som utviklingsmetodikk og UML-diagrammer (tilstandsmaskin og klasse-diagram) for å definere systemets struktur og oppførsel.

3.1 V-modellen i prosjektet

V-modellen er en metodikk for systemutvikling som legger vekt på en strukturert prosess med fokus på verifikasjon og validering gjennom hele utviklingen. Modellen, som er vist i Figur 1, starter med å definere krav, går videre til arkitektur og design, deretter implementasjon, og avsluttes med testing og validering for å bekrefte at systemet oppfyller de opprinnelige kravene. Dette sikret at hver fase bygget på den forrige og at systemet møtte spesifikasjonene.

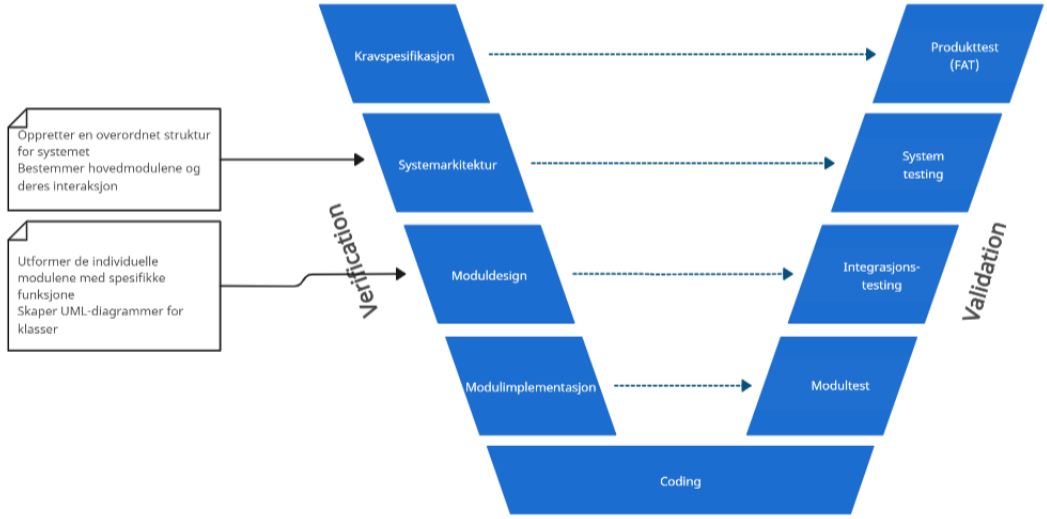

Figure 1: Den pragmatiske V-modellen benyttet i prosjektet.

I dette heisprosjektet ble V-modellen brukt for å sikre en systematisk utvikling av heissystemet. Prosessen begynte med en analyse av kravene fra spesifikasjonen, som er dokumentert i Appendiks B.1 i *main.pdf*. Disse kravene omfatter funksjoner som oppstart av systemet, håndtering av bestillinger, kontroll av lys og dører, samt sikkerhets- og robusthetsaspekter.

Deretter fulgte arkitektur- og designfasen, der UML-diagrammer ble brukt for å visualisere og definere systemets struktur og oppførsel. For eksempel ble et klasse-diagram (Figur 3) utviklet for å beskrive systemets klasser, deres attributter, metoder og relasjoner. Tilstandsdiagramet ble også brukt til å illustrere heisen oppførsel

Etter at designet var ferdigstilt, ble systemet implementert i programmeringsspråket C. Implementasjonen fokuserte på å følge det definerte designet, samtidig som koden ble holdt modulær og enkel å vedlikeholde.

3.2 Tilstandsdiagram

Forklaring av tilstandsdiagrammet:
Diagrammet viser heisens tilstandsmaskin, som styrer systemets oppførsel basert på ulike tilstander

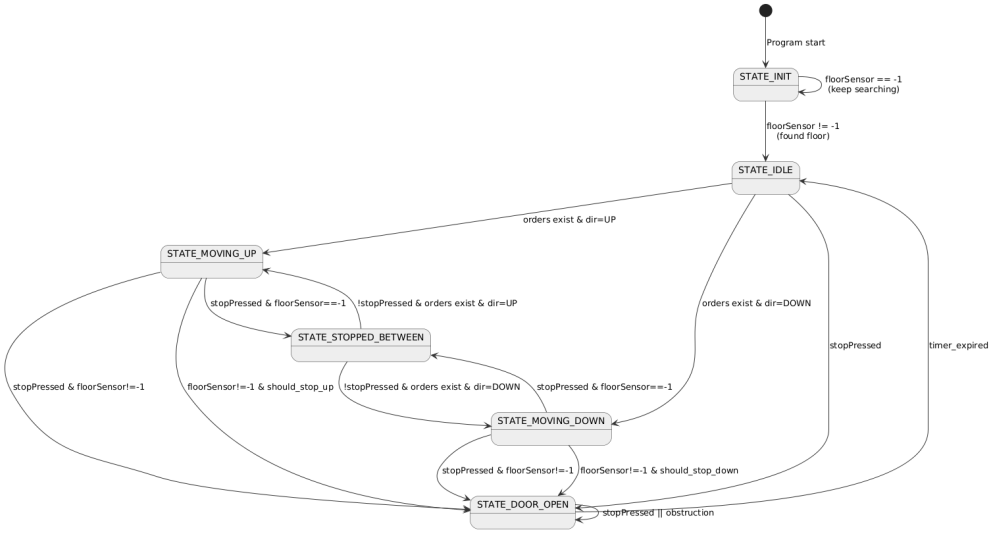

Figure 2: Tilstandsdiagrammet for heisen (tilstandsmaskin

og overgangsbetingelser. Heisen starter i `STATE_INIT`, hvor den beveger seg nedover til den finner en etasje, og går deretter til `STATE_IDLE`. Fra `STATE_IDLE` kan heisen bevege seg til `STATE_MOVING_UP` eller `STATE_MOVING_DOWN` avhengig av bestillinger. Hvis stoppknappen trykkes mens heisen er i bevegelse, går den til `STATE_STOPPED_BETWEEN`. Når heisen ankommer en etasje, går den til `STATE_DOOR_OPEN`, hvor dørene holdes åpne i tre sekunder før den returnerer til `STATE_IDLE`. Overgangene styres av sensorer, knapper og tidsutløp, og sikrer at heisen reagerer korrekt på brukerinput og systemtilstander.

3.3 Klasse-diagram

Klasse-diagrammet (vist i Figur 3) gir en oversikt over systemets arkitektur og illustrerer de sentrale klassene i heissystemet, deres attributter, metoder og relasjoner. Diagrammet er utviklet i tråd med kravspesifikasjonene og reflekterer den modulære strukturen som er implementert i systemet. Nedenfor følger en detaljert beskrivelse av hovedkomponentene:

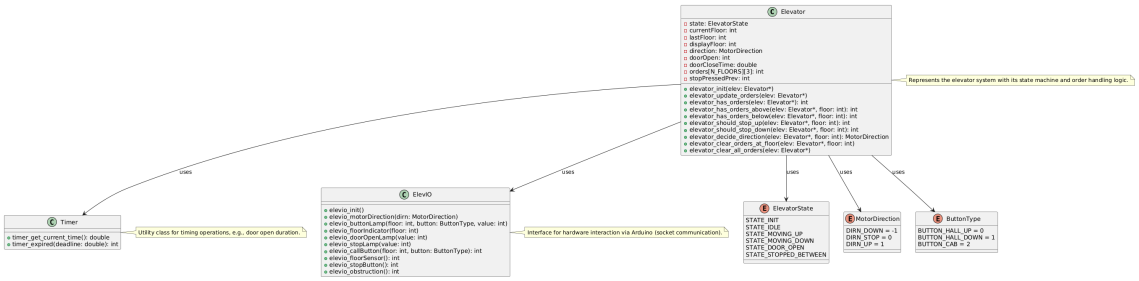

Figure 3: UML klasse-diagram for heissystemet.

3.3.1 Elevator-klassen

Beskrivelse: `Elevator`-klassen er kjernen i heissystemet og håndterer logikken for tilstandsmaskinen samt bestillingshåndtering.

- **Attributter**:
  - `state: ElevatorState` – Heisens nåværende tilstand (f.eks. `STATE_IDLE`, `STATE_DOOR_OPEN`).
  - `currentFloor: int` – Gjeldende etasje heisen befinner seg i (-1 hvis mellom etasjer).

- `lastFloor: int` – Sist besøkte etasje, brukt for å håndtere stopp mellom etasjer.
- `displayFloor: int` – Etasje vist på indikatoren.
- `direction: MotorDirection` – Bevegelsesretning (`DIRN_UP`, `DIRN_DOWN`, `DIRN_STOP`).
- `doorOpen: int` – Indikerer om døren er åpen (1) eller lukket (0).
- `doorCloseTime: double` – Tidspunktet når døren skal lukkes.
- `orders[N_FLOORS][3]: int` – Matrise som lagrer bestillinger for hver etasje og knappetype.
- `stopPressedPrev: int` – Forrige tilstand av stoppknappen for å detektere trykk.

- **Metoder**:
  - `elevator_init(elev: Elevator*)` – Initialiserer heisen til `STATE_INIT`.
  - `elevator_update_orders(elev: Elevator*)` – Oppdaterer bestillinger fra knappene.
  - `elevator_has_orders(elev: Elevator*): int` – Sjekker om det finnes ubetjente bestillinger.
  - `elevator_has_orders_above(elev: Elevator*, floor: int): int` – Sjekker om det er bestillinger over en gitt etasje.
  - `elevator_has_orders_below(elev: Elevator*, floor: int): int` – Sjekker om det er bestillinger under en gitt etasje.
  - `elevator_should_stop_up(elev: Elevator*, floor: int): int` – Avgjører om heisen skal stoppe mens den beveger seg oppover.
  - `elevator_should_stop_down(elev: Elevator*, floor: int): int` – Avgjører om heisen skal stoppe mens den beveger seg nedover.
  - `elevator_decide_direction(elev: Elevator*, floor: int): MotorDirection` – Bestemmer kjøreretning basert på bestillinger.
  - `elevator_clear_orders_at_floor(elev: Elevator*, floor: int)` – Fjerner bestillinger for en spesifikk etasje.
  - `elevator_clear_all_orders(elev: Elevator*)` – Fjerner alle bestillinger (f.eks. ved nødstopp).
- **Formål**: Denne klassen støtter kravene for bestillingshåndtering og sikkerhet ved å koordinere heisens bevegelser og tilstander.

3.3.2 Timer-klassen

Beskrivelse: En hjelpeklasse som håndterer tidsrelaterte operasjoner.

- **Metoder**:
  - `timer_get_current_time(): double` – Returnerer nåværende tid.
  - `timer_expired(deadline: double): int` – Sjekker om en tidsfrist er utløpt.
- **Formål**: Støtter krav som krever at døren holdes åpen i et spesifikt tidsintervall (f.eks. 3 sekunder).

3.3.3 ElevIO-klassen

Beskrivelse: Fungerer som et grensesnitt mellom programvaren og heisens fysiske komponenter via Arduino. Dette er en tildelt modul.

3.3.4 Enums

- `ElevatorState`: Definerer tilstandene i tilstandsmaskinen, som `STATE_INIT`, `STATE_MOVING_UP`, osv.
- `MotorDirection`: Definerer motorretninger: `DIRN_DOWN = -1`, `DIRN_STOP = 0`, `DIRN_UP = 1`.
- `ButtonType`: Definerer knappetyper: `BUTTON_HALL_UP`, `BUTTON_HALL_DOWN`, `BUTTON_CAB`.

3.3.5 Relasjoner

- `Elevator` **bruker** `ElevatorState`, `MotorDirection` og `ButtonType` for å definere tilstand, retning og bestillinger.
- `Elevator` **bruker** `Timer` for å håndtere tidslogikk, som døråpningstider.
- `Elevator` **bruker** `ElevIO` for å kommunisere med hardware.

Klasse-diagrammet illustrerer et modulært design der `Elevator`-klassen styrer kjernefunksjonaliteten, `Timer` håndterer tid, og `ElevIO` abstraherer hardware. Dette reflekterer systemets struktur og støtter videre implementasjon og testing.

4 Implementering

Denne seksjonen beskriver implementasjonen til heissystemet som er skrevet i programmeringsspråket C og tar i utgangspunkt designene fra UML-diagrammene og kravene oppgitt i *main.pdf*. Koden er delt inn i flere moduler, der tilstandsmaskinen for heisen finnes i `main.c`, mens logikken for bestillinger, tidtaking og maskinvarekommunikasjon er lagt i henholdsvis `elevator.c/h`, `timer.c/h` og `elevio.c/h`.

4.1 Overordnet Struktur

Heissystemet er delt inn i følgende hovedmoduler:

- `main.c`: Inneholder en kontinuerlig hovedløkke. Denne løkken leser inn sensordata og håndterer tilstandsmaskinen via en `switch-case`.
- `elevator.c/h`: Definerer `Elevator`-strukturen og funksjoner for bestillingshåndtering og retning.
- `timer.c/h`: Enkle funksjoner for tidsmåling, eksempelvis for å kontrollere hvor lenge døren skal stå åpen.
- `elevio.c/h`: ≪Driveren≫ for heismodellen. Her leses og skrives det til knapper, etasjesensorer, motor, etc.

4.2 Tilstandsmaskinen i `main.c`

Hovedløkken i main.c er systemets hjerte og kjører en uendelig løkke som:

1. **Leser inn sensordata**: Stoppknappen, obstruksjonsbryteren og etasjesensoren. I tillegg hentes et tidsstempel `now` for bruk i tidsrelaterte avgjørelser, som dørlukking.

2. **Oppdaterer bestillinger**: Dersom `stopPressed` ikke er aktiv, kalles `elevator_update_orders()`, som fanger opp nye trykk på etasje- og heisknapper.
3. **Håndterer nødstopp**: Hvis `stopPressed` registreres, ryddes alle bestillinger via `elevator_clear_all_orders()`
4. **Bytter tilstand** via en `switch-case` på `elev.state`. Eksempel: Fra `STATE_MOVING_UP` til `STATE_DOOR_OPEN` hvis heisen har ankommet riktig etasje.
5. **Setter utganger**: Basert på den nye tilstanden og heisens interne data oppdateres motorretning, dørlys, stoppknapplys og etasjeindikatoren.

Et forenklet utdrag av tilstandsmaskinen i `main.c` vises nedenfor:

```
/* main.c */

int main() {
    elevio_init();
    Elevator elev;
    elevator_init(&elev);

    while (1) {
        // 1) Les input
        int stopPressed = elevio_stopButton();
        int obstruction = elevio_obstruction();
        int floorSensor = elevio_floorSensor();
        double now = timer_get_current_time();

        // 5) Tilstandsmaskinen
        switch (elev.state) {
            case STATE_INIT:
                // ...
                break;

            case STATE_IDLE:
                // ...
                break;

            case STATE_MOVING_UP:
                // ...
                break;

            case STATE_MOVING_DOWN:
                // ...
                break;

            case STATE_DOOR_OPEN:
                // ...
                break;

            case STATE_STOPPED_BETWEEN:
                // ...
                break;
        }

        // 6) Sett motorretning, dør og stopp-lys
        elevio_motorDirection(elev.direction);
        elevio_doorOpenLamp(elev.doorOpen);
        elevio_stopLamp(stopPressed);
```

```
        if (elev.displayFloor != -1) {
            elevio_floorIndicator(elev.displayFloor);
        }

        usleep(1000);
    }
    return 0;
}
```

I hver iterasjon av løkken ser du altså at programmet først henter inn informasjon om hvilke knapper som er trykket, hvilken etasje heisen eventuelt befinner seg i, og om nødstopp eller obstruksjon er aktivert. Deretter brukes denne informasjonen til å avgjøre om heisen skal endre tilstand, for eksempel fra `STATE_MOVING_UP` til `STATE_DOOR_OPEN` når `floorSensor` registrerer en etasje som det er bestilling til. Når state-byttet er gjort, kalles de nødvendige funksjonene for å sette motor- og lysutganger korrekt.

Denne inndelingen gjør at det blir enkelt å følge «flyten» gjennom koden: Den ene delen (inndata) sørger for å innhente all informasjon, mens selve `switch-case`-blokken (`elev.state`) bestemmer neste steg i kjøreprosessen. Til slutt oppdateres selve maskinvaren, slik at heisen beveger seg eller døren åpnes/lukkes i henhold til hva tilstandsmaskinen foreskriver.

4.3 `elevator.c/h`- Logikk og Bestillinger

Elevator-modulen, definert i `elevator.h` og `elevator.c`, utgjør kjernen i heissystemet. I denne modulen finner vi:

- `Elevator`**-struct** som lagrer tilstand, etasjer, dørstatus, etc.
- Funksjoner for å oppdage nye bestillinger (`elevator_update_orders()`) og for å tømme bestillinger (`elevator_clear_orders_at_floor()`/`elevator_clear_all_orders()`).
- Hjelpefunksjoner for å avgjøre om heisen bør stoppe i en gitt etasje, avhengig av retning.

4.3.1 Elevator struct

Hovedstrukturen for heisen er `Elevator`-strukturen. Denne holder oversikt over gjeldende tilstand, posisjon, dørstatus, ordrematrisen og mer.

```
typedef struct {
    ElevatorState state;           // Gjeldende tilstand (init, idle, bevegelse, etc.)
    int currentFloor;              // -1 hvis mellom etasjer
    int lastFloor;                 // Siste etasje heisen var i
    int displayFloor;              // Etasjeindikatoren som vises
    MotorDirection direction;      // DIRN_UP, DIRN_DOWN eller DIRN_STOP
    int doorOpen;                  // 1 hvis døren er åpen
    double doorCloseTime;          // Tidspunkt for når døren skal lukkes
    int orders[N_FLOORS][3];       // Ordrematrise: [etasje][BUTTON_HALL_UP, BUTTON_HALL_DOWN, BUT
    int stopPressedPrev;           // Forrige status av stoppknappen
} Elevator;
```

4.3.2 ElevatorState-enumet

En sentral del av implementasjonen er enumet **ElevatorState**, som definerer de mulige tilstandene heisen kan befinne seg i:

```
/* elevator.h */

typedef enum {
    STATE_INIT,
    STATE_INIT,
    STATE_IDLE,
    STATE_MOVING_UP,
    STATE_MOVING_DOWN,
    STATE_DOOR_OPEN,
    STATE_STOPPED_BETWEEN
} ElevatorState;
```

Dette enumet er fundamentet for tilstandsmaskinen og bestemmer hvordan heisen reagerer på input. Hver tilstand har en spesifikk rolle:

1. **STATE_INIT**: Heisen initialiseres og beveger seg til nærmeste etasje.
2. **STATE_IDLE**: Heisen står stille og venter på nye bestillinger.
3. **STATE_MOVING_UP**: Heisen beveger seg oppover mot en bestilling.
4. **STATE_MOVING_DOWN**: Heisen beveger seg nedover mot en bestilling.
5. **STATE_DOOR_OPEN**: Heisen har stoppet, og dørene er åpne i tre sekunder.
6. **STATE_STOPPED_BETWEEN**: Heisen har stoppet mellom etasjer, f.eks. ved bruk av stoppknappen.

4.3.3 Bestillingshåndtering

Heissystemets bestillingshåndtering er implementert ved hjelp av en todimensjonal matrise, `orders[N_FLOORS][3]`. Hver rad i matrisen representerer en etasje, mens de tre kolonnene tilsvarer knappene:

- `BUTTON_HALL_UP` – bestilling for oppreisning fra en etasje,
- `BUTTON_HALL_DOWN` – bestilling for nedreisning fra en etasje, og
- `BUTTON_CAB` – bestilling fra innsiden av heisen (kabin).

Funksjonen `elevator_update_orders()` er ansvarlig for å oppdatere denne matrisen basert på brukerens knappetrykk. Den benytter funksjonen `elevio_callButton()` for å sjekke status på hver knapp i alle etasjer, og dersom en knapp er aktiv, settes det tilsvarende elementet i matrisen til 1. Kode:

```
void elevator_update_orders(Elevator* elev) {
    for (int f = 0; f < N_FLOORS; f++) {
        if (elevio_callButton(f, BUTTON_HALL_UP)) {
            elev->orders[f][BUTTON_HALL_UP] = 1;
        }
        if (elevio_callButton(f, BUTTON_HALL_DOWN)) {
            elev->orders[f][BUTTON_HALL_DOWN] = 1;
        }
        if (elevio_callButton(f, BUTTON_CAB)) {
            elev->orders[f][BUTTON_CAB] = 1;
        }
    }
}
```

Når heisen stopper ved en etasje, kalles `elevator_clear_orders_at_floor()` for å fjerne alle bestillinger knyttet til den etasjen og for å slukke de tilhørende knappelysene. I tillegg benyttes funksjonene `elevator_should_stop_up()` og `elevator_should_stop_down()` under heisens bevegelse for å avgjøre om heisen skal stoppe i den aktuelle etasjen, avhengig av bevegelsesretningen og de aktive bestillingene.

Denne strukturen sikrer at alle bestillinger blir nøyaktig registrert og håndtert, slik at heisen stopper riktig og at knappelysene oppdateres korrekt etter at bestillingen er utført.

4.4 `timer.c/h` - Tidshåndtering

`timer`-modulen gir to essensielle funksjoner:

```
double timer_get_current_time(void);
int timer_expired(double deadline);
```

- `timer_get_current_time()` henter et tidsstempel (i sekunder).
- `timer_expired(deadline)` returnerer `1` dersom systemklokken har passert `deadline`.

Dette er nyttig for at døren kun holdes åpen i et definert tidsrom (for eksempel 3 sekunder). I `STATE_DOOR_OPEN` brukes dette til å styre dørens åpningstid, med unntak hvis stoppknappen (`elevio_stopButton()`) eller hindringsbryteren (`elevio_obstruction()`) er aktivert, da holdes døren åpen lenger.

4.5 `elevio.c/h` - Kommunikasjon med Heismodellen

Modulen kommmuniserer mot maskinvaren via Arduino. Den tilbyr funksjoner for motorstyring, lampehåndtering og sensoravlesning. Her er noen viktige fuksjoner:

- Lese inn knappetrykk (`elevio_callButton()`),
- Finne nåværende etasje (`elevio_floorSensor()`),
- Sette motorretning (`elevio_motorDirection()`),
- Tenne og slukke lys (`elevio_doorOpenLamp()`, `elevio_stopLamp()`, `elevio_floorIndicator()`).

Ved å kapsle dette i en egen modul, unngår resten av koden å håndtere detaljene ved nettverkskommunikasjon eller fysisk I/O-håndtering. Dette er en kode vi fikk tildelt, dermed ikke skrevet av oss selv. Uansett, så var det viktig å forstå hvordan denne koden funket