Project: Specification

1DT059: Model-based Design of Embedded Software, HT20

October 7, 2020

In this project, you will develop and deploy embedded software to control a small (simulated) elevator system. The (simulated) elevator has been constructed by Magnus Lång in September 2020. It has GUI interface animation, programmed in Qt. It is controlled via an API, described in appendix A. Your tasks include to design a control algorithm to control the (simulated) elevator, to develop a Simulink model that mimics the (simulated) elevator system, to generate and deploy code from the control algorithm, and to test and tune your design. Designing a Controller for a similar (not identical) Elevator was part of Assignment 2b. so you should have finished this assignments before starting with the project. You can use and adapt the solution from one of the group members as a basis for the project. Note that the assignment 2b controller needs some extensions to fit into the project. You may also want/need to improve it while carrying out the project.

> This assignment is to be solved by groups of two students. Each group should hand in their solution, and also demonstrate their working design. If there are difficulties to form groups, get back to us, and we will try to resolve this. It is of course not allowed to share or copy (parts of) solutions between groups.

The interface, shown in fig. 1, contains 7 buttons (five, numbered 0, 1, 2, 3, 4, correspond to five floors, the sixth to open the doors, and the seventh is an emergency stop button) that can be pressed by users (for simplicity, we do not distinguish between buttons at floors and buttons inside the elevator, there is just one button for each floor). The (simulated) elevator has a lamp for each floor, and a lamp connected to the emergency stop button. The movement of the (simulated) elevator cart is controlled by a motor, which you can control via the given API. The position of the elevator cart can be read by the Controller via a "sensor", which can be accessed via the supplied API. Note that this position sensor can be unreliable and can sometimes produce strange results (see further below).

Whenever a button for floor n has been pressed, a corresponding light should light, and the elevator must eventually (possibly after passing and stopping at some other floors) arrive to floor n, and the doors open. The elevator then stays at this floor until the control unit orders it to move to some other floor. Once the doors are open, they remain open

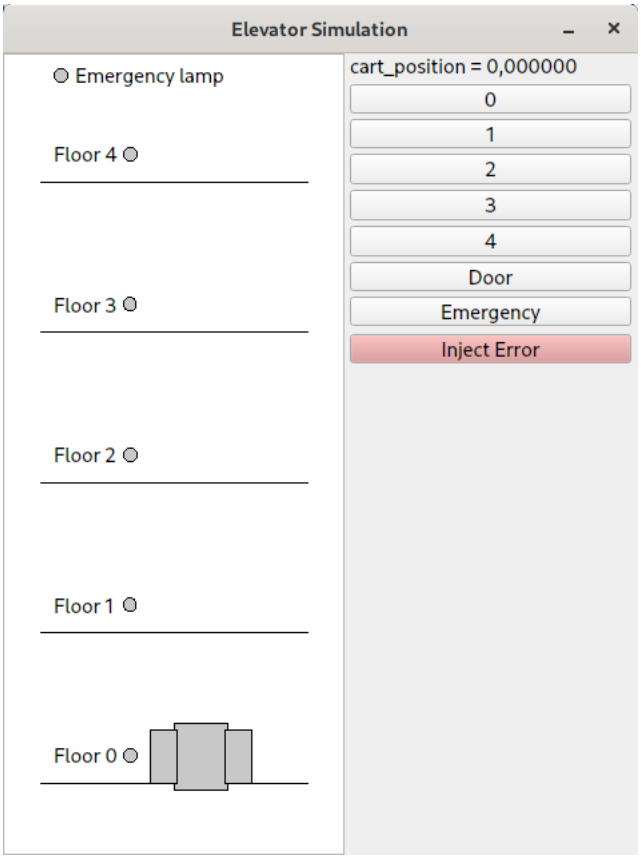

Figure 1: Simulator Graphical Interface

for at least 5 seconds. The elevator must close the doors before leaving a floor. There is a red lamp, intended to represent that the emergency stop has been pressed, and a set of lamps, each corresponding to a floor. Your task is to

1. model the elevator controller in Simulink/Stateflow, and
2. generate code from your model to control the (simulated) elevator system according to the specification given in this document. You should combine your generated code with the (simulated) elevator to produce a working (simulated) elevator which is controlled via the interface.

In parallel with the above you should

1. make a Simulink/Stateflow model of the elevator system, which includes the above controller as a component, and extends it with a Simulink model that behaves approximately like the given (simulated) elevator system. For this, you may (but are not required to) (re)use and adapt the animation and interface that you produced for Assignment 2b. You can keep the same kinds of "lamps" to represent pressed buttons and opened doors as in Assignment 2b. A requirement is that the elevator cart in your Simulink model moves in approximately the same way (similar speed, starting and stopping behavior) as that in the (simulated) elevator.

The requirements are quite similar to those for the last problem in Homework 2b. They include

- Typical "elevator-behavior" requirements, including that: whenever the button for floor n has been pressed, the elevator must eventually arrive to floor n. When this happens, the doors open. Once the doors are open, they remain open for at least 5 seconds. The doors must close before leaving any floor. Whenever the "emergency stop" button is pressed, the elevator is stopped and the red lamp is lit. The lamp is switched off and the elevator resumes to normal when the "emergency stop" button is pressed again.
- The elevator should run smoothly, and make smooth stops and starts at floors. It should thus exhibit a suitable maximum speed and a suitable maximum absolute value of acceleration.
- The elevator should stop only at the precise locations of the floors, as specified by the API.
- The elevator should be robust to faults/disturbances, at least in the following sense: if the distance sensor provides clearly wrong data (e.g., if the **"position"** input shows strange values for some reason, then the elevator should detect that "something is wrong", and perform suitable action (maybe this could be just stopping, and/or possibly lighting up some lamp, etc. Feel free to design this in a suitable way).

You can reuse your floor-planning algorithm from Assignment 2b, but if its performance is not good, you are encouraged to optimize it (this has lower priority than making the elevator functional).

Some of the subtasks that are involved include:

- You should make yourself familiar with the (simulated) elevator and its interface; this is described later in this document.
- Making a Simulink model of the correspondence between the control signals from the controller to the (simulated) elevator, controlling the speed of the elevator cart. This correspondence includes, soft starting and stopping, and approximately similar speed.

What you should hand in You should produce a report (in .pdf), where you describe your overall design effort (what design decisions where applied to the various parts), how your controller is designed (description of its main mechanisms), how your Simulink model was constructed (the main steps of this). Include the models of controller and simulated elevator.

You should also present your design on the day of project presentations. This presentation should explain your controller design, and how you produced the Simulink elevator model. You should also demonstrate a usage scenario on the (simulated) elevator, and thereafter apply the same usage scenario on the Simulink model.

To produce a model that can be used both for code generation and simulation, the block *Environment Controller* (see Simulink documentation) can choose to forward either of two signals, depending on whether code is generated or the model is simulated. You may consider whether you want to use it (using it is not required). An illustration of how the block can be used is given in the file `possible_sketch.slx`.

Submission

Solutions (all files) to this assignment are to be submitted via the Student Portal by **October 21, 2020**. You should also prepare a demonstration. These demonstrations should happen on **October 23, 2020**. A schedule will be put up later, please plan to attend also demonstrations of other groups.

A The Elevator Simulator

The elevator simulator is a program that you run on your computer. You program it by writing a program that interfaces with a C API, very much like if you were programming a micro-controller to drive a real elevator.

You will receive full source code for the elevator simulator, so that it may be compiled for your own computer, but you are not allowed to modify it, and are not intended to inspect it, aside from the `simulation/elevator.h` header file that declares the methods of the API, shown below:

```c
/* Elevator Simulator, version 1 */
#ifndef ELEVATOR_H
#define ELEVATOR_H

#ifdef __cplusplus
extern "C" {
#endif

#include <stdbool.h>

/* Opaque handle type to a simulation */
typedef struct elevator *elevator_h;

/* Bitfield of which buttons are pressed down. */
typedef unsigned button_state;
#define BUTTON_STATE_FLOOR0    0x01
#define BUTTON_STATE_FLOOR1    0x02
#define BUTTON_STATE_FLOOR2    0x04
#define BUTTON_STATE_FLOOR3    0x08
#define BUTTON_STATE_FLOOR4    0x10
#define BUTTON_STATE_DOOR      0x20
#define BUTTON_STATE_EMERGENCY 0x40

/* Bitfield of which lamps should light. */
typedef unsigned lamp_state;
#define LAMP_STATE_FLOOR0    0x01
#define LAMP_STATE_FLOOR1    0x02
#define LAMP_STATE_FLOOR2    0x04
#define LAMP_STATE_FLOOR3    0x08
#define LAMP_STATE_FLOOR4    0x10
#define LAMP_STATE_EMERGENCY 0x40

/* Starts the simulation. Writes a valid handle to *handle_p and returns 0 on
 * success, returns nonzero on failure. */
int start_elevator(elevator_h *handle_p);
/* Stops the simulation */
void stop_elevator(elevator_h);
/* Speeds are between -1 (full speed down) and 1 (full speed up). */
void set_motor_speed(elevator_h, float);
/* Height of the elevator in floors above ground level. Values range between
 * 0.0 (ground floor) and 4.0 (4th floor). */
float get_cart_position(elevator_h);
/* Get the buttons that are currently pressed down */
button_state get_button_state(elevator_h);
/* Opens or closes the doors. */
void set_doors_open(elevator_h, bool);
/* Sets the lamps specified in lamp_state to on, the rest off. */
void set_lamp_state(elevator_h, lamp_state);
```

```
49
50  #ifdef __cplusplus
51  }
52  #endif
53
54  #endif // ELEVATOR_H
```

You will also receive a file `main.c` that demonstrates the use of this API, which you should replace.

```c
/* Don't include anything else from simulation/ */
#include "simulation/elevator.h"

#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    elevator_h e;

    /* Before we start, we must initialise the simulation */
    int ret;
    if ((ret = start_elevator(&e))) {
        fprintf(stderr, "Failed to initialise elevator simulator: %d\n", ret);
        return 1;
    }

    /* This is an example that runs a fixed program. Replace this by hooking up
     * to your own elevator controller. */
    set_doors_open(e, true);
    sleep(2);
    set_doors_open(e, false);
    set_lamp_state(e, LAMP_STATE_FLOOR4);
    set_motor_speed(e, 0.5);
    /* Note, this is a terrible way to control an elevator. */
    while (get_cart_position(e) < 4) usleep(1000);
    set_motor_speed(e, 0);
    usleep(500000);
    set_lamp_state(e, LAMP_STATE_FLOOR0 | LAMP_STATE_FLOOR2);
    set_doors_open(e, true);
    while (!(get_button_state(e) & BUTTON_STATE_FLOOR0)) usleep(1000);
    set_doors_open(e, false);
    set_motor_speed(e, -0.5);
    sleep(5);
    set_motor_speed(e, 0);
    sleep(1);

    /* If we decide to quit, for whatever reason, we should stop the
     * simulation. */
    stop_elevator(e);
```

```
40      return 0;
41  }
```

B Compiling the Elevator Simulator

Please use a Linux environment to work with the simulator. You should have one available from the code generation lab.

In order to build the elevator simulator, you also need Qt and CMake. On Ubuntu, you can install these with

```
$ sudo apt install qtbase5-dev cmake
```

Then, assuming you have the elevator in a directory called `elevator_sim` under the current, you can make a build directory `build` like this:

```
$ mkdir build && cd build
$ cmake ../elevator_sim
```

Then, in the build directory, you can just type `make` to build the simulator. This produces both a dynamic library `libSimulation.so` with the simulator, and an executable `elevator_sim` (from `main.c`). You can then run it in the `build` directory with

```
$ ./elevator_sim
```

You should see the interface (fig. 1). When you're ready, add your generated code and replace `main.c` to hook the simulator up to your controller. You can add extra `.c` files to the build in `CMakeFiles.txt`. Good luck!