Mbdes Project

Fredrik Tåkvist Neethu Joseph Nusrat Hossain

October 2020

1 Design Decisions

We chose which elevator model to use as a baseline on two criteria: graded score, and modifiability. The elevator chosen was designed with extensions in mind, so proved useful for the project. It also already contained a gui with a simulation of an elevator. We chose to first extend the Simulink model to fit the new requirements, including adding a fifth floor and adding acceleration when starting and stopping. Once we had a model, we copied the controller into a new Simulink model, where we adapted it for code generation. We chose this method, rather than incorporating both simulation and code generator into a single model, because it required the least amount of learning and was therefore time-efficient.

2 Elevator Model

The elevator model consists of two parts, the controller and the gui. The controller takes the position of the elevator as input variable. The input events consist of the five floor buttons, the emergency button, and a pulse generator used to activate the controller at regular intervals so it can poll the position. The controller has four outputs - the speed of the elevator, an array containing the states of the floor indicator lights, the emergency indicator light and the door opened indicator light. The emergency and door indicators have only two states - On or Off. The floor lights have four states - Off, On, Target and Here. On means that the elevator has been called to that floor. Target means that the elevator is currently heading towards that floor. And Here means that the elevator is at that floor and the doors at that floor are open.

The control module contains two parallel states, one for controlling the elevator and one for catching floor button presses. When a floor button is pressed, the floor button controller sets the state of the corresponding floor indicator to On if it is currently Off. It also generates a Button event to inform the elevator controller that a floor button has been pressed.

The elevator controller has four states for normal operation - *Idle*, *Plan*, *Moving* and *Doors*. In the *Idle* state, the speed is set to 0. When a Button event is generated, the controller moves to the *Plan* state. At this state a Matlab

function is called to plan the next move. The planning function looks at the floor indicators in the elevator's direction of travel and chooses the closest floor with with an active indicator (i.e. an indicator in the On or Target state) as the target floor, unless the floor is too close for the elevator to stop. If there's no such indicator in the direction of travel, the planner instead chooses the closest floor in the the opposite direction. This ensures that all floors with active indicators will eventually be served by the elevator. If the planner sets a target floor, the controller transitions to the *Moving* state and sets the speed of the elevator to maximum. Otherwise, the controller moves back to the *Idle* state.

In the *Moving* state, the controller polls the position sensor to check whether the elevator has reached its target floor. When the elevator is within 0.4 floors away from the target, it enters a *Breaking* state. At this state, the speed is set to the distance from the floor, ensuring a smooth deceleration. The 0.4 value was chosen as the elevator seems to require slightly less than half a floor to from coming to a stop after moving at full speed. Should a floor button be pressed when the controller is in the *Moving* state, it moves back to the *Plan* state. This way, if a floor button for a closer floor in the direction of travel is pressed, the elevator will set that as the target. When the elevator reaches its target floor, the controller waits for half a second and then transitions to the *Doors* state, where the door indicator is set to On and the floor indicator is set to Here. After 5 s, the controller transitions to the *Plan* state, so that the elevator can continue its journey. At this point, the door indicator and floor indicator are set to Off.

In addition to normal operation, there's also an *Emergency* state. The controller transitions out of normal operation into the *Emergency* state whenever an Emergency button event occurs, setting the speed to 0 and turning on the emergency indicator. When the Emergency button is pressed again, the controller transitions back to normal operation. If the door indicator is on when this happens, the controller transitions back to the *Doors* state. Otherwise, it transitions to the *Idle* state. The elevator also transitions to the *Emergency* state when the difference between the current position and previous position is greater than 0.1 floors. Since the elevator has a time-step of 0.001 and the maximum speed of the elevator is 1 floor/s, if the elevator has moved more than 0.1 floors since last position poll, something must have gone wrong with the position sensor. When entering the *Emergency* state because of a fault in the position sensor, the elevator waits for the position sensor to stabilize before allowing for normal operation to be resumed. It waits until the position sensor has been stable for 5 seconds, before waiting for the Emergency button to be pressed again.

3 GUI

The GUI part of the Simulink simulation consists of three parts - a Gui controller, an acceleration subsystem and an error injection subsystem. The accel-

eration subsystem approximates the acceleration of the elevator. The acceleration seems to depend on the velocity. Each time-step, the model calculates a maximum acceleration depending on the velocity. This acceleration is added to the velocity each time-step (unless the difference between the velocity and the target velocity are smaller than the maximum acceleration, in which case the difference is added instead to prevent the velocity becoming too large).

The error injector uses a uniformly distributed random number generator to set the position to a number between 0 and 5. If the Error Injection button is pressed, the position value from the acceleration subsystem is replaced with the random number. Otherwise, the position from the acceleration subsystem is fed through.

The Gui controller consists of 8 parallel state machines, controlling the 7 buttons and the animated doors. To ensure that the signal for the buttons reaches the elevator controller (since it's not continuously updated) each button has two states - *On* and *Off*. When a button is pressed, the Gui controller sets the corresponding button signal and doesn't reset it until it receives an appropriate response from the elevator controller. For the floor buttons, the appropriate response is that the corresponding floor light is not in the Off state. For the Emergency button, the response is for the emergency light to be toggled. The Gui controller also animates the doors as opened whenever the door light is on. This is also done by moving between an *On* and *Off* state.

All of the buttons share a single callback function. This function differentiates between the buttons by an index parameter. The state of each button is stored in an array and the button callback function sets the element of corresponding index to 1. The Gui controller polls array to observe the state of the button.

4 Code

No optimization was performed on the code generated by Simulink, as the code is fast enough for the elevator to run smoothly. The code in *main.c* first initializes the model and performs the first step, to prepare the state machine. It then enters a while loop where it polls for button presses and sets the inputs of the model accordingly. After this, the motor speed of the elevator is set to the speed output of the model and the door opened status is set to the door output from the model. Finally, another step of the model is executed and the simulator sleeps for 1 ms (as that is the size of the model's time-steps).