UPPSALA
UNIVERSITET

Programming Embedded Systems Hand-in 3

Group 10

Nusrat Hossain Fredrik Tåquist

December 27, 2019

1 Test Case 2

We based our solution heavily on the code for test case 1. First we close the doors and send the elevator to floor 3. Just like in test case 1, we reset the pin for floor 3 after 2 s and then enter the 45 s loop. In the loop we wait for the position to reach 300, which is about 2 s away from floor 2 at 50 cm/s. At this point the elevator should be too close to floor 2. Since we wait 1 s between each position check we need 2 s to ensure that the button is pressed before the elevator reaches floor 2. We don't release the pin to floor 2 (the person who wants to go there is impatient and keeps the button pressed until he arrives -

everyone knows that the elevator goes faster if you keep pressing the button, right?).

There's a boolean value *reached_floor_3_first* which is set to 1 if floor 3 is reached but floor 2 is not. It is set to 0 if floor 2 is reached but floor 3 is not. This way it will be 1 if the test passes. We also check whether floors 2 and 3 have both been reached.

2 Random testing

We implemented random testing in the function *testCaseR(seed)* in *testcaser2.ini*. It takes as a parameter the seed of the random number generator. The script assumes that the doors are initially open but will close them in the first iteration of the loop body. Since the planner waits for two seconds before moving the elevator, if the script is not started before those two seconds have elapsed a safety violation occurs, related to the doors.

The test is run in an infinite loop. There are four separate counters, one for generating random events, one for keeping track of how long the doors have been open, one for keeping track of how long the stop button has been pressed, and finally one that allows the script to check whether a floor has been reached. This latter counter always triggers at 1 second, the others have a degree of randomness.

To get an event we generate a uniformly distributed random value between 0 and 90. An event to a pin, p, is generated according to

$$p = \begin{cases} 0 & \text{if } 0 \leq u \leq 29 \\ 1 & \text{if } 30 \leq u \leq 59 \\ 2 & \text{if } 60 \leq u \leq 89 \\ 3 & \text{if } u = 90 \end{cases},$$

where u is the random value. In this way a stop button event (pin 3) is much less likely than an event for any of the floor buttons (pins 0-2). The uniform distribution also makes all of the floor buttons equally likely to have an event. An event for a button toggles the button pin (this way there's no need to keep track of which buttons have been pushed, they will be released eventually - as a button release isn't handled by the planner it makes no difference for the simulation whether a button is continually pressed or not). Whenever the stop button is pressed, a timer starts. After a certain time after this the stop button will be released. This is a random delay between 1 and 4 seconds.

The code then goes on to check whether a floor has been reached, this happens every second. If it has reached a floor then the doors will open. At this point a counter begins. At a random time between 1 and 3 seconds the doors will be closed again. The trigger for reaching a floor only happens once per floor to ensure that the doors do not open and close again. The fact that the floor sensor is checked every second means that the planner must stay in place for at least two seconds to be sure to catch a door opening.

The delay for random actions is set to 10 seconds, which simulates a lift that is in frequent use as it takes 15 seconds to go between two floors. This way it's likely that a new event will trigger while the elevator is moving while not spamming the planner with events. The random numbers are all drawn from a uniform distribution. It would be more realistic to use a Gaussian random number generator for a normal distribution. However, converting a uniform random number to normal is computationally expensive. We could use the Box-Muller transform, but that requires a square root, a logarithm and a cosine - very expensive. Besides, with a uniform distribution edge cases are more likely to occur.