POLITECNICO

MILANO 1863

Automation of an Elevator Plant

Report for the Automation and Control Laboratory a.y. 2019/2020

Prof. Alan Facchinetti

Federico Ciotti, 905961

Alex Miguel Franco Martínez, 928019

David Estebán Luna Sanchez, 913300

Giuseppe Matteo Daniele Savino, 920028

Summary

List of figures

List of tables

[1]: from CS 134: Operating Systems: File System Implementation, available at https://www.cs.hmc.edu/~geoff/classes/hmc.cs134.201209/slides/class20_disks3_beamer.pdf

1. Introduction

1.1 Project Description

Within the control theory, whether in the industrial field or not, one of the most used and accepted ways to control different types of systems, is mainly based on the implementation of discrete events; this project aims to demonstrate the usage of this type of control through the execution of different logics in the specific case of an elevator plant.

To carry out this control, a PLC (ABB) was used, which was programmed through the platform called CoDeSys, where the control logic was implemented through different blocks and Structured Text functions.

This project is divided into two main parts; The first focuses on the logic used to manage elevator calls, where algorithms based on the movement flow are applied to handle different calls at the same time. In addition, different cases that are simplifications of the main algorithm are analysed (one single call at a time).

Moreover, the second main part is related to the basic security measures that must be provided in an operation without affecting the call management logic (Stop in case of emergency due to any source of misfunction and management of the door opening/closure process when an obstacle its detected impeding the entrance).

1.2 Approach and goals

This report is a guide to describe our approach to the work, the environment on which we based and tested our project and, following, going into the details of the different logics used, the code we developed and some proposals that can be taken into account to improve this work if applied to a real case.

To have an optimized approach to the work, first we focused on the assignments and their deadline that we should have respected. For this reason, we differentiated the work in three big sections:

- First, the decision of the logics that our elevator should have operated and the design of all the schemes needed, and on which we would have based the program coding;
- Second, the proper coding on CoDeSys, its debug and testing, on the simulation and later on the real elevator model;
- Third, the report writing, with an overview about the conclusions and all the future possible improvements for our project.

For all the work on the project and in particular to define the solution of the problems we followed a «Waterfall model». The Waterfall model is a sequential approach, where each fundamental activity of a process is represented as a separate phase, arranged in linear order.

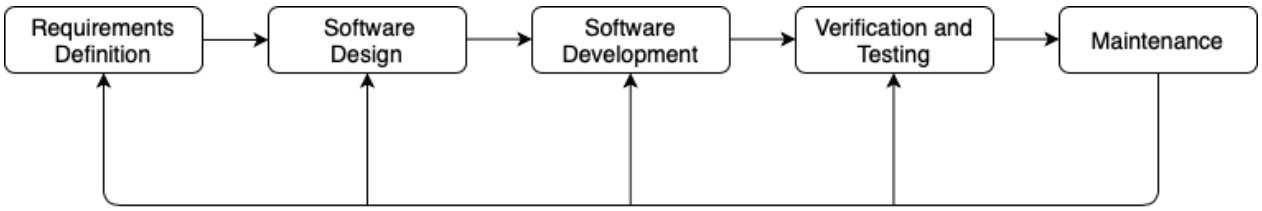


Figure 1.1: Implementation phases

The phases of the waterfall model are: Requirements, Design, Development, Testing and Maintenance. We have planned all the activities before start working on them. But the software development is not a simple linear process, it involves feedback from one phase to another one. The definition of macro-functions helped us to speed the debugging process up when some errors were detected.

We did all the phases of the project in parallel, working each one on one assigned function; then, in the end of each stage, we assembled all the work together, to find errors and for a best final result. Our goal for this project has been surely to develop a robust and correct behaviour for the elevator, without forgetting about the safety point of view, applying all the knowledge acquired during previous years.

2. Hardware and software

2.1 The plant: the elevator model

Our plant is made up by a didactic elevator “Didactic lift ASC89”.

Figure 2.1: Didactic lift

The educational lift used for this project, which represents a real but small-scale system, consists of a main cabin connecting the ground floor to the third floor.

The elevator is equipped with one electric motor that moves the cabin among the four different floors. The elevator presents four floors, with a total of four doors that can be opened and closed thanks to four different electric motors, four cabin presence sensors to detect the presence of the cabin at the corresponding floor, and other eight optical sensors to detect the complete opening or closing of the door.

In the model there are two main input panels, one of them represents the cabin-call panel and the other represents the floor calls.

Figure 2.2: Cabin buttons and leds

In the cabin panel we can find the four buttons to request a *call* to reach each floor, a *stop* button to stop the movement of the elevator at any time instant and a *switch* to simulate the presence of an obstacle obstructing the door at any floor. A red cabin presence *led* can be found next to each *call* button, while a yellow *led* at the top represents the cabin lights.

Figure 2.3: Floor buttons and leds

At each floor we find a button for a "*upcall*" and a button for a "*downcall*", which should be pressed by the user according to his intentions. At the 3rd and ground floors only a *downcall* button and an *upcall* button are present respectively. Moreover, a red cabin presence *led* is located at each floor, together with two yellow *leds* representing every pending *upcall* or *downcall*.

Figure 2.4: Safety key

In order to avoid any uncontrolled situations, a safety key activates some added safety mechanical sensors to provide a wider radius of control to verify the presence of movements outside the allowed range, in this case defined by physical constraints.
Safety sensors are placed at the maximum/minimum reachable point of the cabin. the same is done to limit out-of-range movement of the doors (one sensor for the closure and one for the opening).
If a safety sensor is activated, the entire system is locked down to avoid mechanical damage.
usually, when the system is blocked due to the activation of a safety sensor, the whole system needs to be reset (turn the key) due to the fact that the sensors and therefore the variables are locked in an intermediate state, probably not managed by the program because out-of-range operation.
This safety feature is not reachable through the program.

2.2 PLC and I/O modules

A Programmable Logic Controller (PLC) is an industrial computer control system that continuously monitors the state of input devices and makes decisions based upon a custom program to control the state of output devices.

Depending on the inputs and outputs, a PLC can monitor and record run-time data related to manufacturing machines, chemical plants and so on, automatically start and stop processes, generate alarms if a machine malfunctions, and more.

The PLC receives information from connected sensor or input devices, processes the data and triggers outputs based on pre-programmed parameters.

Figure 2.5: ABB PLC PM554 128kB

The whole project aimed at the implementation of functionalities devoted to manage in proper way elevator calls in different scenarios.

The ABB PLC module is usually programmed with a dedicated software that allow exchange of information between PC and PLC module through a wired connection.

Through the user's written program, the PLC is able to manipulate and read inputs and outputs according to the problem.

In addition, two I/O modules (DC532) connected to the PLC module were used, each one composed by 16 inputs and 16 outputs.

Figure 2.6: 2X DC532, Digital I/O Mod 16DI/16DC

2.2.1 I/O variables

The system is equipped with many sensors. some of them indicate the presence of the cabin on the floor, others to check if doors are completely opened or closed.

In order to have an easy overview of the inputs and outputs of the system.

A table is shown below

No.	Variables	Input/Output	Description
1	CabGoUp	O	Active cabin motor up
2	CabGoDown	O	Active cabin motor down
3	CabLight	O	Active cabin light
4	CabLed0	O	Indicator of cabin presence ground floor
5	CabLed1	O	Indicator of cabin presence 1st floor
6	CabLed2	O	Indicator of cabin presence 2nd floor
7	CabLed3	O	Indicator of cabin presence 3rd floor
8	Floor1LedDw	O	Indicator of call up 1st floor
9	Floor2LedDw	O	Indicator of call down 2nd floor
10	Floor3LedDw	O	Indicator of call down 3rd floor
11	Floor0LedUp	O	Indicator of call up ground floor
12	Floor1LedUp	O	Indicator of call down 1st floor
13	Floor2LedUp	O	Indicator of call up 2nd floor
14	Floor0DoorOpen	O	Command to open ground floor door
15	Floor1DoorOpen	O	Command to open 1st floor door
16	Floor2DoorOpen	O	Command to open 2nd floor door
17	Floor3DoorOpen	O	Command to open 3rd floor door
18	Floor0DoorClose	O	Command to close ground floor door
19	Floor1DoorClose	O	Command to close 1st floor door
20	Floor2DoorClose	O	Command to close 2nd floor door
21	Floor3DoorClose	O	Command to close 3rd floor door
22	CabStop	I	Cabine internal Emergency Stop
23	CabObstacle	I	Simulation of an obstacle
24	CabCallP0	I	Cabine internal call to ground floor
25	CabCallP1	I	Cabine internal call to first floor
26	CabCallP2	I	Cabine internal call to second floor
27	CabCallP3	I	Cabine internal call to third floor
28	Floor0CabPresence	I	Cabin presence ground floor
29	Floor1CabPresence	I	Cabin presence 1st floor
30	Floor2CabPresence	I	Cabin presence 2nd floor
31	Floor3CabPresence	I	Cabin presence 3rd floor
32	Floor0DoorOpened	I	Ground floor open door
33	Floor1DoorOpened	I	1st floor open door
34	Floor2DoorOpened	I	2nd floor open door
35	Floor3DoorOpened	I	3rd floor open door
36	Floor0DoorClosed	I	Ground floor closed door
37	Floor1DoorClosed	I	1st floor closed door
38	Floor2DoorClosed	I	2nd floor closed door
39	Floor3DoorClosed	I	3rd floor closed door
30	Floor0UpBt	I	Call up from ground floor
31	Floor1UpBt	I	Call up from 1st floor
32	Floor2UpBt	I	Call up from 2nd floor
32	Floor1DwBt	I	Call down from 1st floor
34	Floor2DwBt	I	Call down from 2nd floor
35	Floor3DwBt	I	Call down from 3rd floor

Table 2.1: I/O of the elevator

2.3 Programming environment

In our case the PLC was programmed through a software provided by its manufacturer ABB: Automation Builder 2.2.

ABB Automation Builder is an integrated software suite for machine builders and system integrators, aiming to automatize machines and systems. It is mainly used to program, debug and maintain automation projects.

The development environment provided by the software is CoDeSys (version 2.3.9.62). It lets the user create an application to be uploaded on the PLC module in order to control the system.

Although different languages have been presented to us and can be used in order to write the control program, we choose Structured Text for the development of our project. Like all others CoDeSys development languages, it is supported by the international industrial IEC 61131-3 standard and it is a high-level language, this allowing us to implement complex instructions for the control application.

2.3.1 Overview on CoDeSys

The main sections of CoDeSys are:

1. POUs : Program Organization Unit
In this section the main program and all the function can be programmed and visualized.
2. Data Types
In this section it is possible to create new types of variables. In our case, since the elevator structure is easily "modulable", we used this section to define some *struct* types which would represent all I/O of a single floor (then an array of four floors was defined), all I/O of the cabin, and another type in order to represent the Door Opening/Closing operation.
3. Visualization
In this section the user can create a graphic interface in order to simulate the behaviour of the program regardless of the physical system. Given the Covid-19 pandemic and therefore the impossibility to work in direct contact with the physical system from the beginning, the development of a visual simulation (see chapter 3) has been critical for the success of the project development.
4. Resources
This is the section where the user can define global variables, add libraries and configure some of the PLC parameters.

3. Visual Simulation

In order to be able to work remotely on the project, we implemented a dynamic simulator, thanks to the CODESYS interface. Through this graphical interface we could test directly, even when hardware was unavailable, the elevator's response to our code.

For the implementation of the visual simulator, we created a static and elongated rectangle representing the elevator shaft, two smalls rectangles for each floor that describe the operation of opening and closing doors and in dark grey inside the shaft we represent the cabin of the elevator.

For what concerns the different lights indicators of the system, for each floor we have the buttons related to upcall and downcall, merged with the corresponding leds so that each button lights up if pressed. The lights denoting the elevator presence at a precise floor are present next to the call buttons, at each floor, indicating when the elevator is in that particular position.

The slider introduced is aimed to handle the switching of functionalities between the simple single call logic and the second optimised one.

At the end, we have also the cabin panel, through which we can manage the input buttons indicating the *cabin calls*; in our case, we decided to indicate whenever a *cabin call* button is pressed, lighting it in red to have a clearer overview of the pending requests. Furthermore, in the cabin panel there is also the *emergency stop* button that activates the emergency functionality, and the *obstacle* simulator. They both change colour when pressed and return to the original state if re-pressed.

For the cabin and door motion a *Movement* function was developed (see appendix 7.2), that manage movements according to the activation/deactivation of the output variables corresponding to the cabin and doors motors and increment or decrement the variables which describe absolute position on the graphical environment.

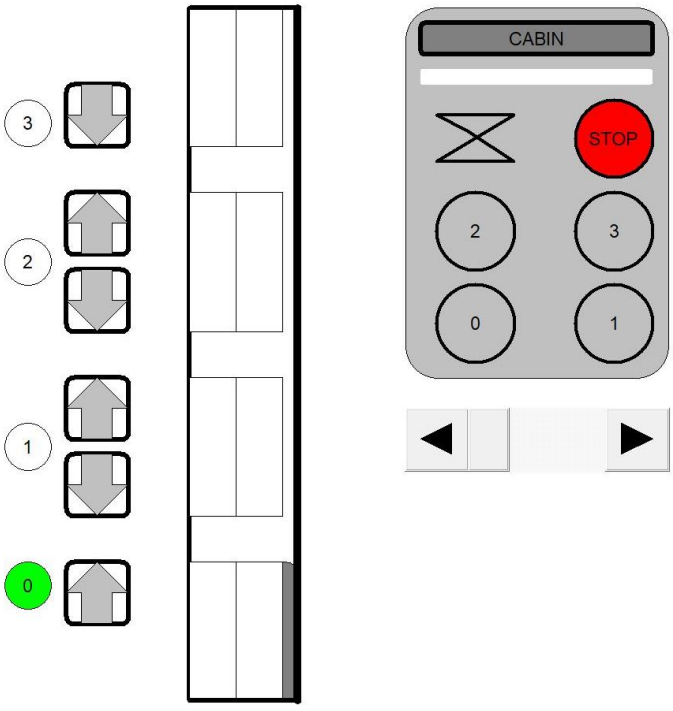


Figure 3.1: Visual simulation on CodeSys

In the function *CabinSensor* we developed the virtual sensors variables in order to provide to the code the needed feedback from the system, which in this case were unavailable.

Both these functions must be excluded from the build when the code is employed to control the real system, since in that case the cabin and the doors will be actually moved by the motors and the PLC will have the real sensors feedbacks at its disposal.

4. Development

To carry out the project, first we started from the basic logic of the elevator, the single call management: the elevator waits until one of the call buttons is pressed and performs the request, it moves to the floor if the call arrives from a different level or open the doors if it arrives from the same.

After this basic logic we coded the emergency case program, the program that must operate when the emergency button is pressed; we immediately focused on this element of the code because it is a very important part of the program, that guarantees the safety of the system. We implemented this solution in both our logics, the basic and the multiple one.

In the end we completed the requirements implementing the optimized logic for the managements of multiple simultaneous calls.

For all our programs we started from the flow chart design, to have an overview of its behaviour and to start thinking about how to compile the code; only in a second moment we focused on the proper code writing.

4.1 First Logic - Single call

4.1.1 Description

As said before, this is our first approach for the management of the elevator, the simplest logic in which an elevator could work.

In this case the program does not have a memory storage for the calls, therefore the first call request is taken into account and fully completed before having the possibility of considering another call; all the requests performed during the processing of another call are simply neglected by the program.

4.1.2 Requirements

Each call is performed by pressing one of the buttons linked to the desired floor. Once the call is saved as the desired position to be reached, the elevator must move upwards or downwards, depending on its current position, until the sensor detecting the cabin presence at the desired floor becomes active.

The plant can only move when all the doors are closed, therefore the doors cannot change their state until the elevator stops at their corresponding floor.

When the cabin reaches its destination, the system undergoes the doors management operations. They need to be fully opened, stay opened for a certain time interval to allow the entrance or exit of the users, and then be closed again.

During the door management phase, we also need to check for possible obstacles between the doors, that must impede the closure of the doors and their reopening.

In our system, the obstacle is defined by the position of the obstacle switch of the didactic elevator. As long as the obstacle is present, the doors must be kept open and they can start the closure phase only after the removal of the obstacle.

After the doors are completely closed and the desired floor has been reset, the system remains still, waiting for the next call.

Moreover, during the motion of the cabin, there is also the possibility of the activation of the emergency stop command. This causes the elevator to stop immediately in the position at which the button was pressed and all the lights except for the cabin light to blink intermittently, until the emergency button is pressed again. If this button is pressed while one of the four cabin presence sensors detects the presence of the cabin in correspondence of a floor, as well as stopping its motion the elevator also opens the door to let the passengers escape from the cabin.

Lastly, there is the lights management that concerns different kind of leds. One of them is the light inside the cabin, that has to be turned on for all the time in which the elevator is operating, and is turned off only when the elevator is still for a while or the door is opened.

Then we have the leds representing the presence of the cabin at the corresponding floor, which are used to indicate, both to who is inside and outside the elevator, at which floor the cabin is located in that moment. They are triggered by the presence sensors positioned at the various floors and the light is on only for the time that the related sensor is active.

The last kind of lights is the one related to the requested call. When a call button is pressed, its light is switched on until the corresponding floor is reached, so in this simple case only one of these leds can be lighted up at the same time.

Here below it is represented the flow chart for this first logic, with all the steps performed by the algorithm.

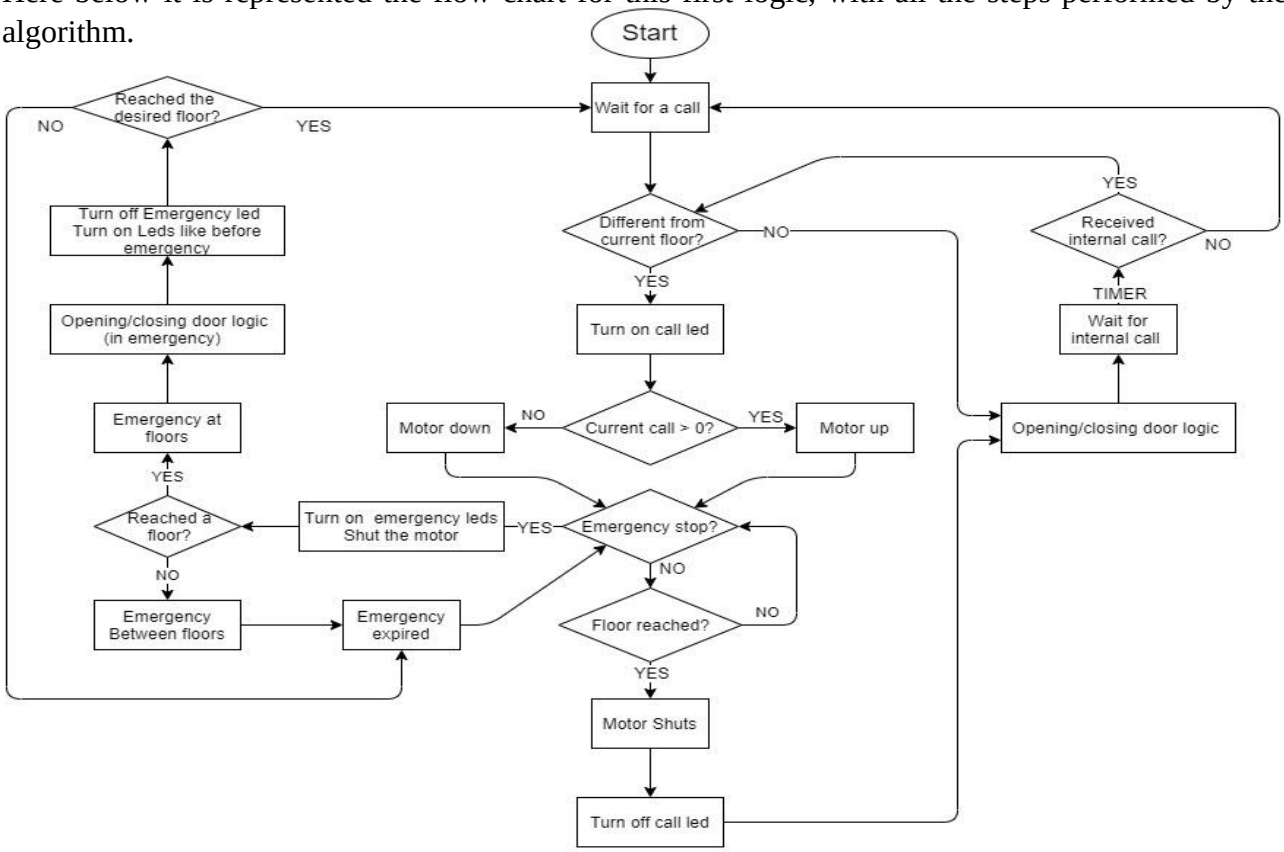


Figure 4.1: Flow chart of the program for a single call management

Here below there are the separated float schemes of, respectively from left to right, Emergency management, opening/closing door logic in emergency and opening/closing door logic. The only difference between emergency and standard door logic is that, in the case of emergency the door remain open until the emergency expires, while in normal condition the door closes when the timer expires.

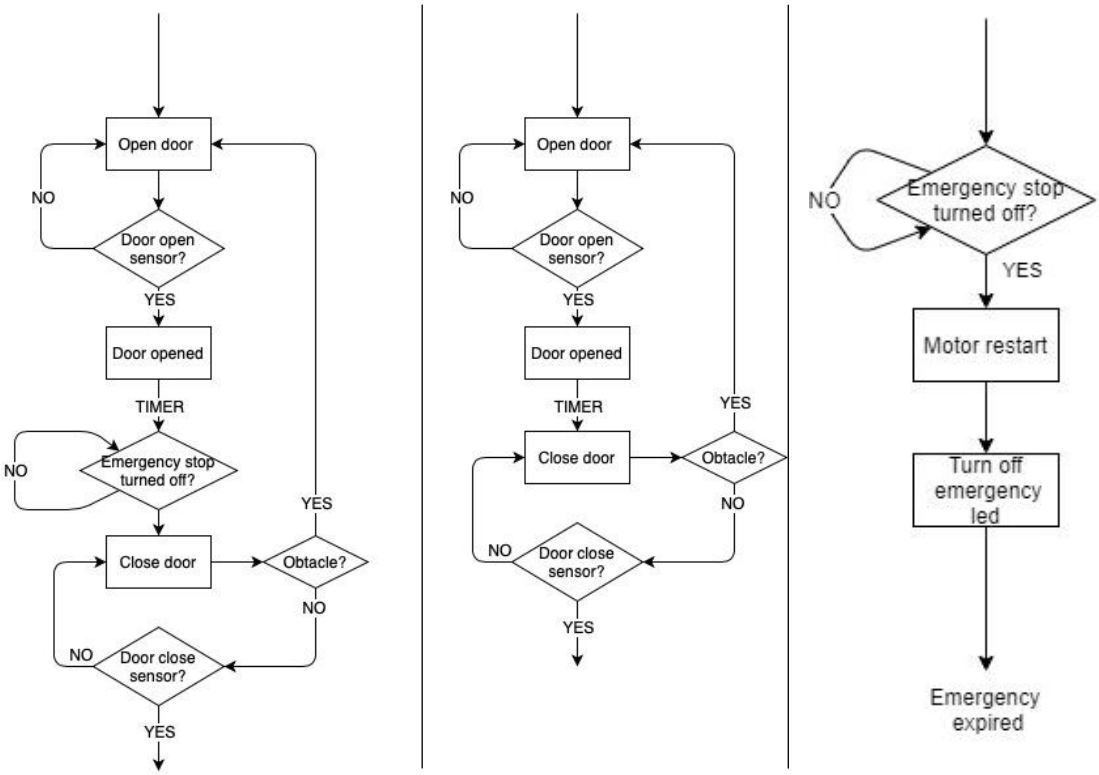


Figure 4.2: Opening/closing door logic in emergency; Opening/closing door logic in normal conditions; Emergency between floors

4.1.3 Example

The user arrives in front of the doors of the elevator and presses the floor up or downcall button, if the elevator is yet in the same floor the door opens immediately, if not, the elevator starts moving to reach the floor from where the call has arrived and opens the door to let the user getting in. After a timer, and if there are no obstacles detected, the door starts closing; when the door is fully closed the user can press one of the four cabin button to reach the desired floor, the elevator starts moving to the desired floor if it is not already there and then when the floor is reached it opens the door.

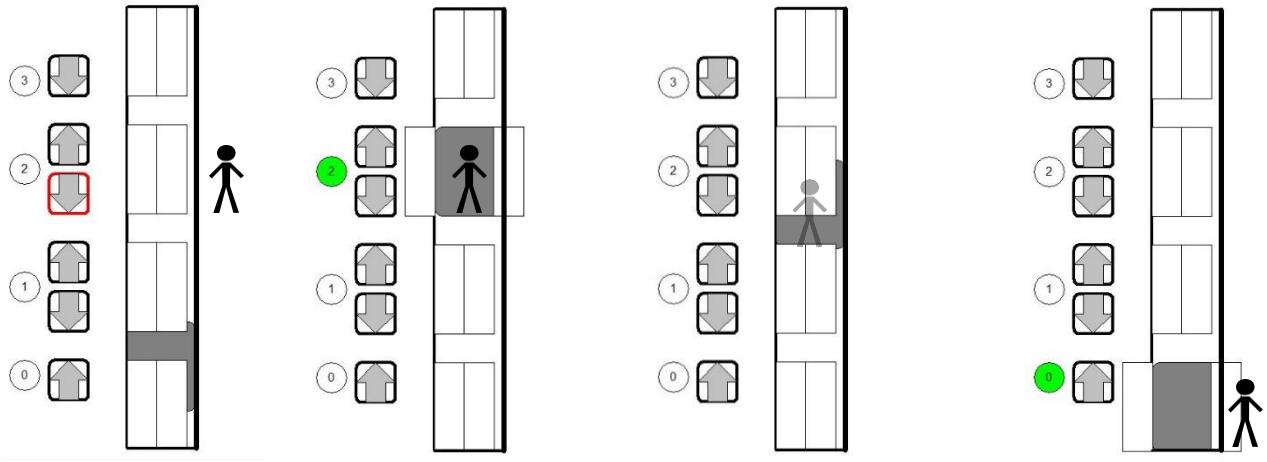


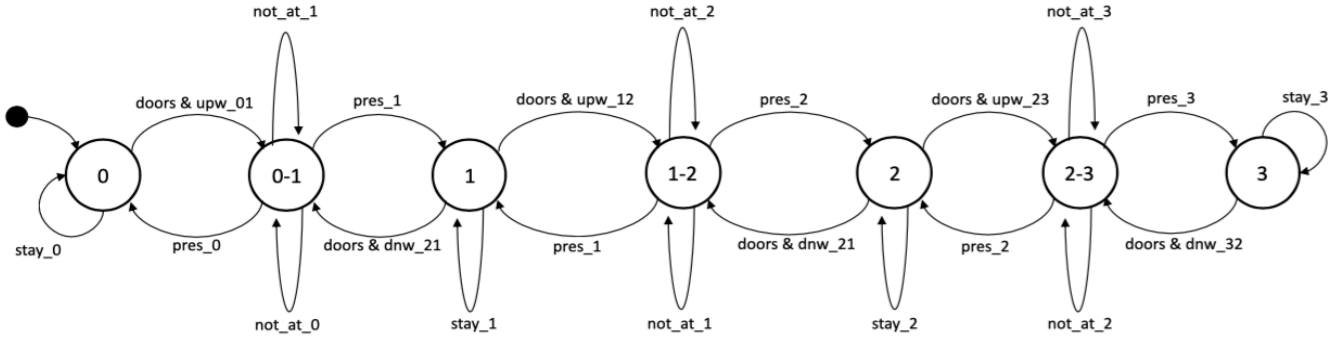
Figure 4.3: Simulation example, first logic

4.1.4 Modelling

Since the project is dealing with a discrete time system, finite state machines are used to characterize the evolution of the states of the different elements of the plant.

Here we report the modelling phase of the single call program.

The elevator can be defined by seven states: four defining the presence of the cabin at a precise floor and three indicating that the elevator is between two floors. We represented the three states defining the “in between” floor state only for simplicity, in our code we don’t have a variable that takes the state “in between” but we derived it through the state of the motor and by the presence sensors.



Repeated:

event	meaning
stay_0	• Sensors detect cabin at 0 floor • Desired floor is 0
stay_1	• Sensors detect cabin at 1 floor • Desired floor is 1
stay_2	• Sensors detect cabin at 2 floor • Desired floor is 2
stay_3	• Sensors detect cabin at 3 floor • Desired floor is 3
doors	Doors are correctly closed

Sensors:

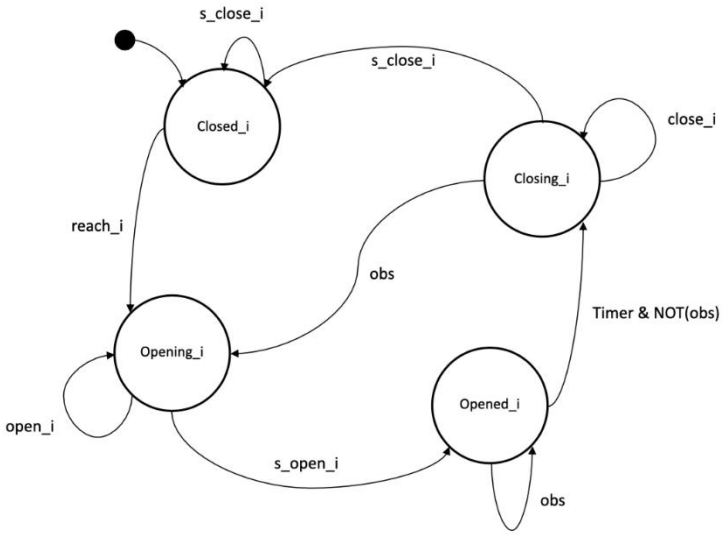
event	meaning
pres_0	Sensors detect cabin at 0 floor
pres_1	Sensors detect cabin at 1 floor
pres_2	Sensors detect cabin at 2 floor
pres_3	Sensors detect cabin at 3 floor
not_at_0	Sensors have not yet detected presence at 0 floor
not_at_1	Sensors have not yet detected cabin presence at 1 floor
not_at_2	Sensors have not yet detected cabin presence at 2 floor
not_at_3	Sensors have not yet detected cabin presence at 3 floor

Motion:

event	meaning
upw_01	Requested floor greater than 0
upw_12	Requested floor greater than 1
upw_23	Requested floor greater than 2
dnw_32	Requested floor smaller than 3
dnw_21	Requested floor smaller than 2
dnw_10	Requested floor smaller than 1

Figure 4.4: Elevator model

Obviously, the model for the doors of a floor is the same for each floor. It can be modelled by four states: two steady states, when it is completely closed or opened, and two dynamic states, when it is closing or opening. Here below it is reported the model for the door of the i-th floor.

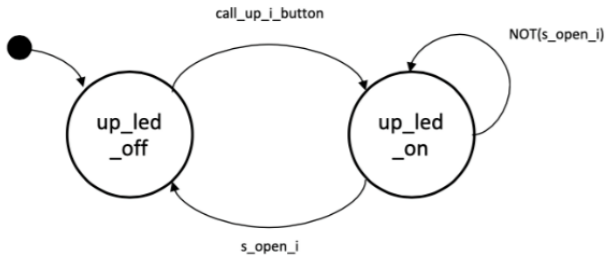


event	meaning
s_close_i	Sensors detect closed doors at floor i
s_open_i	Sensors detect open doors at floor i
timer	Timer expires
obs	An obstacle is detected
close_i	Door is closing
open_i	Door is opening

Figure 4.5: Door model

All the lights have two states (ON and OFF), but there is a difference in the activation condition, here we will report all the models for the lights.

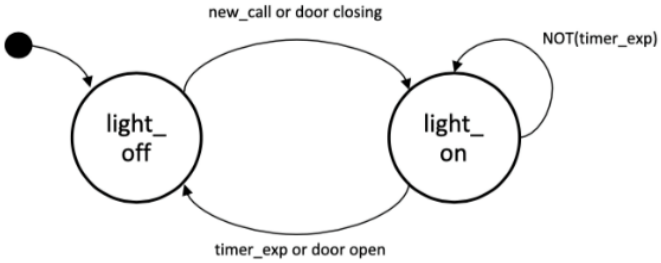
The lights for up or down calls on the floors work in the same way, here there is the up-call lights model, for the down-call lights we just have to change all the “up” written with “down”.



event	meaning
call_up_i_button	Call up button at i-th floor is pressed
s_open_i	Sensors detect open doors

Figure 4.6: i-th Floor call up or down lights model

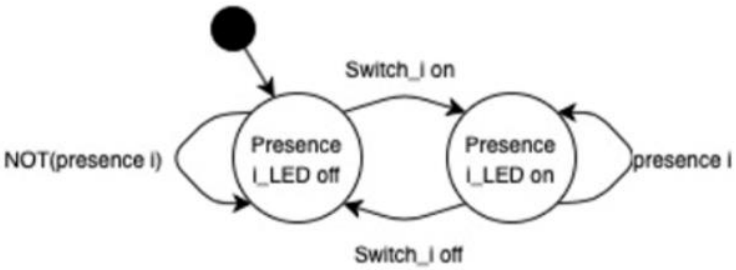
Model of the cabin light.



event	meaning
new_call	Elevator is requested at any floor
door closing	The door is closing
door open	The door is open
timer_exp	Inactivity timer expires

Figure 4.7: Cabin light model

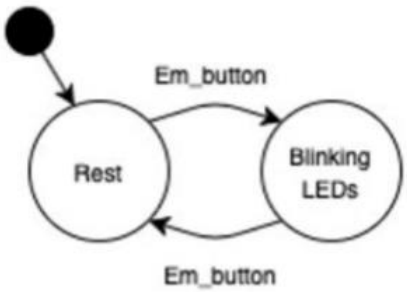
The presence of the elevator turns on the corresponding floor and cabin light, here the model.



event	meaning
Switch_i_on	Elevator arrives at floor i
Switch_i_off	Elevator leaves floor i
presence_i	Sensors detect cabin presence at floor i

Figure 4.8: Presence light model

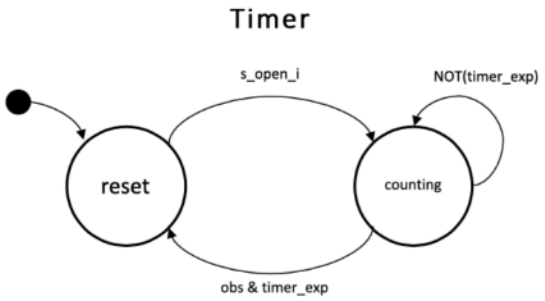
When the elevator is in an emergency state all the lights (except the cabin light) follow this model.



event	meaning
Em_button	Emergency button pressed

Figure 4.9: Model of lights management in case of emergency

The timer that keeps the door open for 2 seconds follows this model.



event	meaning
s_open_i	Sensors detect open doors at floor i
timer_exp	Timer expires
obs	An obstacle is detected

Figure 4.10: Door timer model

4.1.5 Implementation

From the modelling phase we have the basis for the software implementation, which was entirely developed using structured text.

The code is composed of a main program in which initially there is the assignment of the system inputs through a specific function, then there are all the functions defining the actual software and a specific program for the emergency case, which triggers another task with high priority so that emergency operations are privileged over the ordinary ones. At the end, there is another function that allocates all the system outputs. In this way we have the reading phase of the inputs at the beginning of each iteration of the code and the setting of the outputs at the end of the elaboration of all the system variables inside the entire program.

INPUT ASSIGNMENT → DATA PROCESSING → OUTPUT ASSIGNMENT

When the program starts the first time, the first step focuses on the initialization of the plant. The doors are all opened and closed once so that eventual mistakes in doors positions are corrected, then the cabin is brought to floor 0. When this phase has been concluded, the software enters the part of the script that actually defines the system behaviour during its execution and that is described below.

Initially, we have the definition of the current floor at which the cabin is located. After we have all the functions that can be called during the program execution, door opening/closing management, emergency case and obstacle detection.

At this point, there is the selection of the desired floor, the program enters this part when a call, from the floors or from the cabin, is detected. It so changes the variable *DesFloor* value to the correspondent floor of the call and perform the managing of the lights.

Now there is the moving management of the elevator, if the desired floor is different from the current floor the elevator starts moving in the direction of the call. The program recursively checks this case, and when the desired floor is reached it stops the motors and performs all the operations needed: open and subsequently close the door and management of the lights.

In case there was an emergency stop, at any time during the execution, the program enters in the emergency management task and launches the emergency process: shuts the motor, blinks all the lights and if we are in correspondence of a floor opens the door.

At the end of the movement, when the cabin is still at the destination, the elevator wait until there is a new call.

We can have the enabling of the obstacle case at any time, this causes the stoppage of the door, if it is closing and its immediately reopened. The door can start the closing operation only after the removal of the obstacle.

4.2 Second Logic – Optimized multiple calls management

4.2.1 Description

Once the basic single-motion task of the elevator is realized, we now deal with the management of a series of calls to be satisfied following a certain “smart” order. To optimize the logic, as in many working elevator plants, the application is developed following the well-known *elevator algorithm*, also known as *SCAN algorithm* (see section 4.2.5).

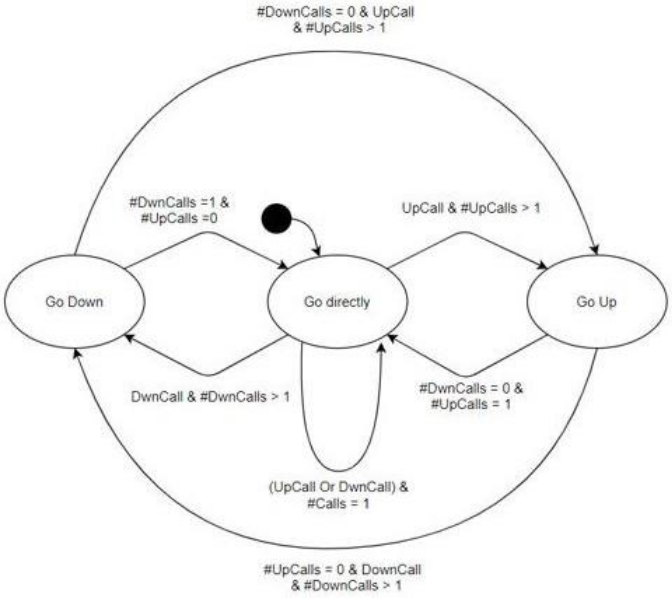
4.2.2 Requirements

To fulfil the requirement of satisfying multiple calls providing the best user experience, given that calls can come at any time, from any floor with any direction, and from the cabin, a simple FIFO approach is not the best solution in terms of power consumption and is also not pleasing in terms of user experience (waiting time, average usage time, comfort). The possibility of knowing in advance the intention of the user (through the *Upcall* and *Downcall* buttons) allows us to program the elevator in such a way that the average passenger travel time and total usage time is reduced as much as possible, thus providing the most comfortable experience to the user.

Moreover, the emergency situations can occur and must be treated as in the first case: so when an obstacle is detected impeding the door, it opens until the obstacle is removed, and when the stop button is pressed, the cabin stops its motion with the same behaviour previously described in section 4.1.2. Additionally to that, we add that no requests can be made while the Stop button has not been pressed a second time, suggesting that the emergency has ended.

4.2.3 Modelling

Since all the fundamental modelling of the system has been built in the developing of the first logic, we expanded the system created during the previous phase, basically implementing two additional functions: one for the calls storage in two *queues* dedicated to cab calls and floor calls, one for the selection of the next *desired floor* among the requests contained in the queues, following as said a precise logic.



event	meaning
#DwnCalls	Number of down calls from lower floors (wrt elevator)
#UpCalls	Number of up calls from higher floors (wrt elevator)
#Calls	Total number of calls in the queue
DownCall	Call requesting the cab to go down
UpCall	Call requesting the cab to go up

event	meaning
Added	Call added to the queue
Button_Call	Button calling the cab pressed
Repeated	The Floor called is already in the queue
Elevator State	Position of the elevator
Doors Closed	Sensors detect that the door has been closed
Deleted	Call deleted from the queue

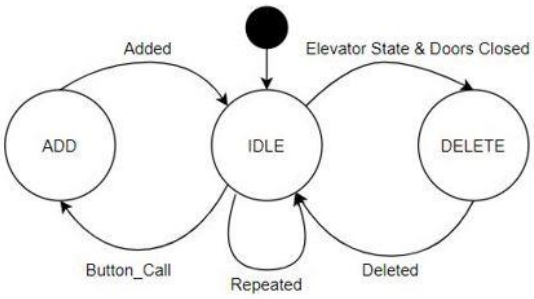


Figure 4.11: Second logic model

4.2.4 Implementation

As said, the two models presented in the previous section have been coded with two different functions which have then been plugged into the code for the first logic.

The two queues have been realized as two arrays, one for the cab calls with four elements (maximum simultaneous requests) and one for the floor calls, with six elements divided in turn into a section of tree elements for the upcalls and a section of three elements for the downcalls, in this case the maximum number of simultaneous requests is covered, too.

Each request is added in a FIFO order inside the corresponding queue, if not already present, and when it is satisfied it is brought at the end of the array and removed. Regarding the removal of a request, we had to have in mind that, if both calls have been made from the same floor to go up and to go down, when we reach that floor we only satisfy the request corresponding to the direction of the motion we are undergoing at that moment.

For what concerns the decision of the next *desired floor* to be reached, another function has been coded which develops the so-called elevator algorithm or F-SCAN algorithm, having the two queues as lists of inputs.

4.2.5 F-SCAN/Elevator algorithm

The F-SCAN algorithm is a well-known scheduling algorithm used in computer science to serve the different requests of a disk starting from a random initial position.

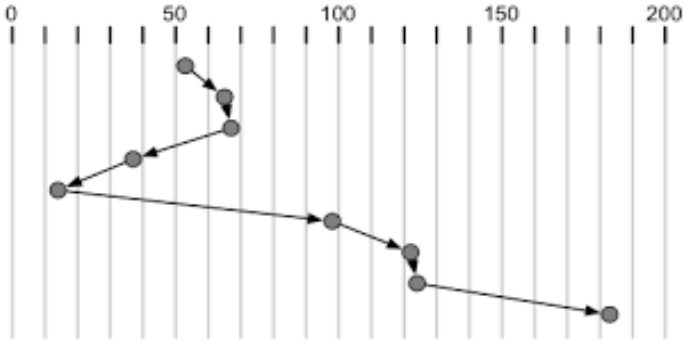


Figure 4.12: Elevator algorithm

In F-SCAN disk scheduling algorithm, head starts from any position of the disk and moves towards one of the two ends, servicing requests in between one by one[2]. It uses two sub-queues. During the scan, all of the requests are in the first queue and all new requests are put into the second queue. Thus, service of new requests is deferred until all of the old requests have been processed. When the scan ends, the arm is taken to the first queue entries and is started all over again[3].

Then the direction of the head is reversed, and the process continues as head continuously scan back and forth to access the disk.

This is exactly the behaviour that we are looking for for our elevator, where our head is represented by the cabin and the requests are the calls stored in the two queues, while the two queues of the algorithm described before, are “virtual” queues created from scratch, one for the ongoing motion and one for the opposite direction, changing in runtime. That's why the SCAN algorithm is also known as the elevator algorithm.
Anyway, some little adaptations must be included in our versions: above all, we must ignore downcalls if we are going upwards and vice versa, unless that call is the last one before switching and starting the motion along the opposite direction of movement.

[2]: SCAN (Elevator) Disk Scheduling Algorithms, https://www.geeksforgeeks.org/scan-elevator-disk-scheduling-algorithms/
[3]: FSCAN. Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/FSCAN

4.2.6 Example

In the example we provide, we simulate a series of calls both from the floors and from the cabin in order to observe the behaviour of the elevator. The elevator, called with an upcall and downcall from floor 1, starts its motion from floor 0. In the meanwhile, another upcall is received from floor 2. The people waiting at floor 1 and wanting to go upstairs enter the cabin (guided by the upcall led that turns off when the floor is reached) and press the cabin call 3 button, while the people wanting to go downstairs have to wait for the return of the cabin in the opposite direction. Then the elevator reaches floor 2, where the waiting people are picked and request another call to reach floor 3, and in the end floor 3 is reached. Once floor 3 has been served, the motion restarts towards downwards direction, reaching the pending request at floor 1, where passengers, once on board, call for floor 0.

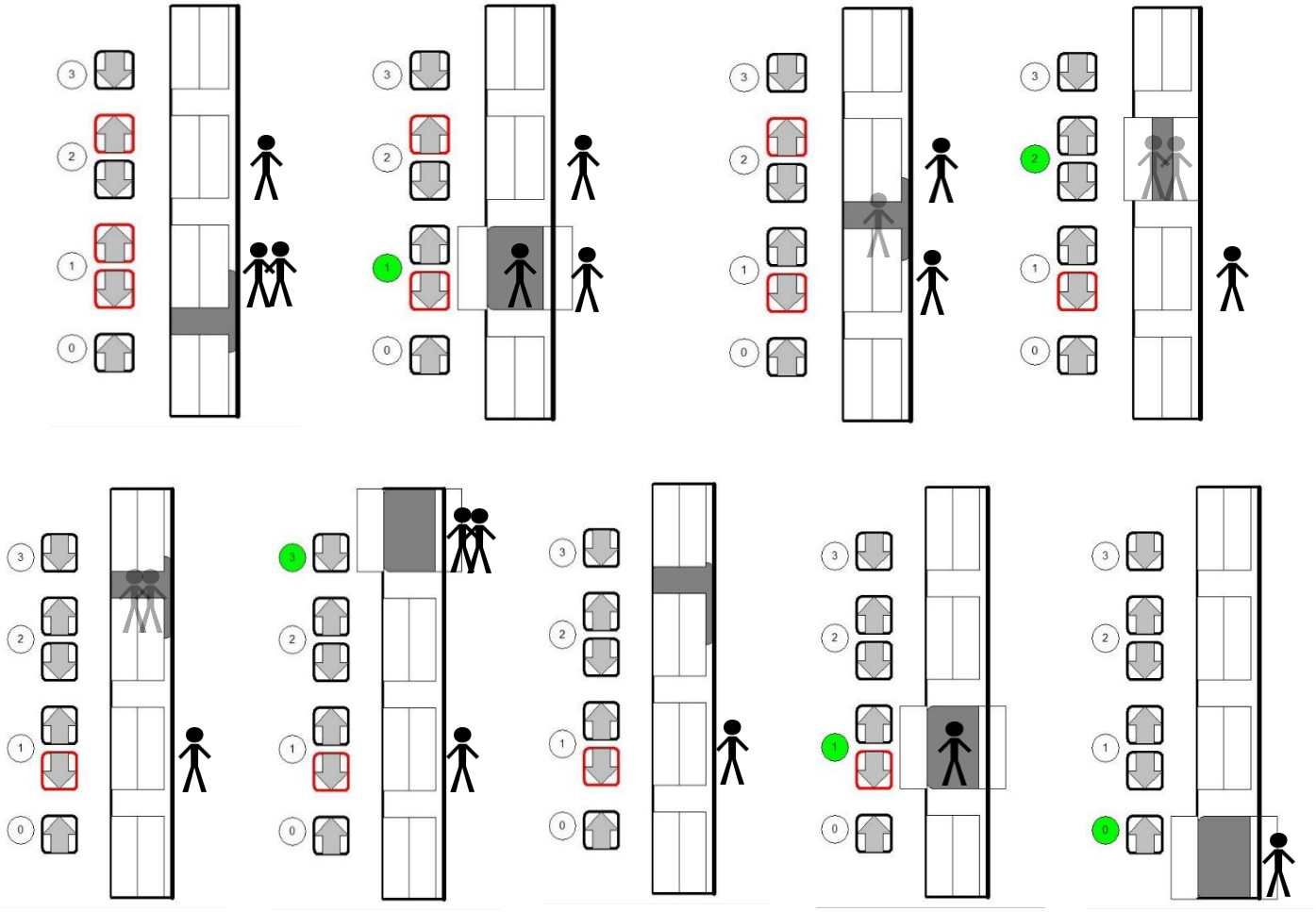


Figure 4.13: Simulation example, second logic

5. Safety Precautions

While coding the functions for the logics of the elevator, our group has always kept in mind the safety of potential users of a similar plant in the real world. Taking a proper working of the actuators and of the sensors for granted, we developed two features in order to provide a decent level of safety measurements.

5.1 Stop when an escape path is detected

An elevator, especially its cabin, is a cumbersome component from the point of view of safety, mainly because of the tiny size of this one. Given that, the rising of claustrophobic fears and panic sensations in the passengers are common in the daily use of a similar plant.
For this reason, we tried to provide an escape path whenever possible in case the stop button in pressed and so an emergency stands out.
First of all, in the cabin is present at a floor level in idle position and with the doors closed, the pressure of the stop button causes the opening of the doors, so that a possible person trapped inside the cabin can exit without any kind of problem.

Moreover, the same feature has been developed in case the cabin is moving and finds itself in correspondence with a floor which is not the desired one. If the stop button is pressed and one of the cabin sensors present at each floor detect the presence of the cabin, the doors open (and remain opened until the stop button is pressed again), so that sensitive passengers are able to find an immediate escape path from the narrow environment of the cabin. Note that the cabin sensors, from what we have seen during the laboratory hours working with the real model of the plant, detect the presence of the cabin in a slightly enlarged "window" with respect to the effective one (see pictures at the end of this section), but in a way that the user must never climb to reach a safe position (if anything, a small jump of few centimetres is required).

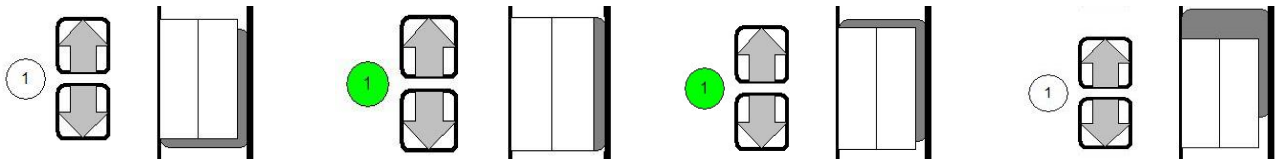


Figure 5.1: Example of cabin sensors tolerance

5.2 Safe single/multiple logic switch

The switch slider that allows us to shift from one logic to the other in the visual simulation (see chapter 3), in principle, authorise the switching at any moment, causing potential overlaps which can lead to malfunctions of the software, like conflicts between queue requests and single request, leds

turned on when no requests have been made, floors served when not needed. To avoid these undesired, and sometimes unsafe, situations, we had to provide a “smart path” for the logic switch.

Since CoDeSys visualisation section does not provide a different behaviour for the slider, we had to keep it as it is and so the slide can be moved anytime. The proper logic switch, however, is permitted only if the cabin motors are off, one of the four cabin sensors detect the presence of the cabin at its floor and we are not in an emergency. Whenever a logic shift is operated, every request, both single and/or contained in the queues, is reset, so the elevator has no requests to fulfil right after the switch until a new one is made from scratch.

6. Conclusions

Throughout the development of the project, although most of the time was spent on the code validation both on the simulation and on the real plant by webcam, we had the possibility to design and tune an automated system from scratch for the first time, starting from the plant modelling up to the selection, implementation and in the end validation of a suitable control logic. Thanks to this process, the main theachings we learnt from this laboratory experience are for sure the unforeseen complications you have to face when dealing with a real system, the importance of acquiring data from well-placed sensors in order to use them to apply the appropriate inputs to the actuators, a procedure we were able to do thanks to the use of the PLC which worked as interface between the sensors and the programming environment. Moreover, we faced the difficulties and the pleasures of working in a team with people of different backgrounds but with a common goal.

6.1 Results:

- **Modularity**

  The structure of the code, which is divided in different functions (see appendix) each one with a goal to accomplish, allows each module to be improved or modified without the need to intervene on the other functions. Moreover, we coded the emergency so that the *stop* push button triggers a new task, with higher priority with respect to the normal one commonly used, so that all emergency operations have priority on the usual.

- **Virtual Simulation**

  The simulation presented the same behaviours of the real system, so it allowed a complete independence from the hardware. This has proven to be useful to develop and have a first verification phase on new code parts.

- **Optimization**

  The multiple calls logic optimizes user experience and the overall cost of the system over the long term in the real case.

- **Robustness with respect to user's misuse**

  As it is conceived, the correct functioning of the multiple optimised logic relies on a proper use by the people calling the elevator from their floor: in fact, the two buttons indicating the *upcall* or *downcall* present in the middle floors are supposed to be a way for the user to communicate with the system, providing information on the desired direction. An improper use of these two buttons could lead to big losses of time and energy consumption, undesired behaviours for the optimised logic.

  Let us suppose, for example, there are 2 users: the first "liar" user calling the elevator from floor 1 using the *upcall* button but heading to floor 0 (so the corrected call would have been

through the *downcall* button), and a "honest" user calling the elevator from floor 2 correctly with the *upcall* button, heading to floor 3.

In order to deal with these kind of errors, we coded the logic so that eventual misuses of the *upcall* or *downcall* buttons do not penalise the "honest" users, which are privileged over the "liar" users, which are, in a way, punished instead.

In the case described above, the user at floor 1 pushes the *cabin call* for floor 0, but the cabin continues his way upwards, to satisfy the request made by the other user. When the calls for this direction are satisfied, then the request from the bad user is then served.

6.2 Proposals:

- **N-Floors implementation**

As already written, the code allows perfect scalability of the variables that define the size of the elevator.

In fact, each floor, no matter where it is located, is seen as a repeated structure with the same components (presence sensors, doors motors, call buttons, etc.), and just the first and the last floors can be considered as "special cases", which the elevator is supposed to move within.

In order to enlarge the system and develop an elevator for a N-Floors building, it would be enough to expand the floor array, the queues (in order to be able to contain every call at the same time), and to adjust a few *for cycles* in the second logic function *desired_floor1* according to the total number of floors to be represented.

- **Reinforcement Learning for elevator dispatching**

So far, the modelling proposed in this report considers an algorithm based on the so-called pain index of the user, where the main goal is the satisfaction of the customer rather the optimality in terms of costs and time. Also, it is worth noting that for this project there was no data/constraints related to the speed of the elevator, the opening and closing times, the stop time, the cabin capacity and the volume of the users for each floor so any implementation further than the one described in the report was not considered. However, if these kinds of variables are considered, a new spectrum of control methodologies are available, one of them is the implementation of Q-Learning algorithms.
The working principle behind the elevators can be seen as a stochastic process where the controller has multiple choices to proceed and the arrival of the passengers can be modelled as a Poisson process[4]. Unlike the classical heuristic controllers used in elevators, the reinforcement learning algorithms make use of the evolution through the control policies of the system state at a certain time *t* and *t+1* where a reward will be specified to the controller according to the goals stablished. The aim is to maximize the accumulated value of the reward function[5]. The action space for an elevator consists on the number of available call requests and the position of the cabin while the control action has three different states: Upward movement, Downward movement and still, for the reward policy, the time parameters are the ones that are used the most. With this type of system description, the Q-Learning is the most

[4]: The University of Alabama in Huntsville: UAH, The Poisson Process [Online], [cited 18th May 2020]. Available: http://www.math.uah.edu/stat/poisson/)
[5]: C. J. Watkins, & P. Dayan. "Q-learning". Machine learning, 8(3-4), 279-292,1992.
[6]: H. Li, "The implementation of reinforcement learning algorithms on the elevator control system," 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, 2015, pp. 1-4, doi: 10.1109/ETFA.2015.7301554.

suitable algorithm for this scenario since the learner and the decision-maker (control policy) has the ability of acquiring knowledge to perform optimally by implementing Markovian domains according to the experience of the consequence for each action taken, without having the necessity of a full picture of the environment where the system is placed[6].

- **Other Safety Measures**

In this project we have been very focused on the safety of the elevator, but the measures can be further expanded. An elevator can be a very dangerous place in certain conditions, for example in case of fire; in our model we only had the *stop* button at disposal to simulate an imminent risk and we only focused on it to manage the emergency functions. To increase safety, other external variables of danger could be added to the model, given of course the appropriate sensors and considered into the logic of the elevator. For example, in case of fire when the elevator is not working, then do not open the door and stay still, or if the alarm turns on when the elevator is moving with people on it, then reach the ground floor.
Another variable of security could be weight control, i.e. if there is too much weight on the elevator, threshold defined by the manufacturer, do not close the door and turn on an indicator light.
These two examples of safety measures are only for "external" variables that can influence the behaviour of the elevator, but the system could be integrated also with other internal safety precautions, for example using sensors inside each motor, both for the doors and the cabin, that detect if the engine is moving or not and compare this value with the variables of the code, if they differ means that there is a problem.

In few words, given the correct sensors in to detect whatever kind of risk for the user, lots of safety expedients and automatisms can be considered to ensure the lowest level of hazard on the plant.

6.3 Distance learning experience:

Due to Covid-19 epidemic, during this academic year the course that supervises the development of this project has been covered in distance learning mode through the Microsoft Teams platform, like all the rest of Politecnico di Milano courses of the same semester, a first time experience both for professors and students.
From our side we have to say we had to deal with some issues arising from this situation, mainly related to the inability to spend the course hours inside the laboratory alongside the plant model and the PLC.
Since the access to these areas was completely forbidden at the beginning of the semester, we were able to come in contact with a plant components description through some presentation slides provided by teachers, so we spent the first classes just modelling our system as a discrete system with Finite State Automata and flow diagrams, in order to have a better overview of what we had to expect from our code.
After the PLC ABB software and the CoDeSys development environment were presented to us, we shifted our focus to proper programming of the code: thanks to the modelling through FSA and some preliminary agreements on the use of the variables, the development of the two tasks and of each

function has been mainly done in parallel between us, since there was no way to work together simultaneously on the same machine, and then the whole code was put together periodically.
Most of the validation of our code has been made on the visual simulation we created, at the beginning, thanks to the visualisations section of CoDeSys, as described in chapter 4.
The interaction with the real plant only came in a second moment, starting after the first half of the semester, always through Microsoft Teams platform. We took control of the laboratory PC which the PLC was connected to, uploaded our code into the PLC and were able to look at the elevator model behaviour through a webcam pointing at it.
Although in the first moments we went through some difficulties in understanding what was going on, mainly visually because a 360° view was not provided, but also because the model was only available once a week and because the safety key, if activated, could only be turned by the professor present at the laboratory, with some experience with these circumstances we managed to remotely control the real plant in a proper way, using the virtual push buttons we created for our simulation.

7. Appendix
7.1 Elevator Datasheet

DIDACTICAL MODELS

Didactic lift

The ASC89 lift is a model which may be connected to a PLC or some microprocessors.
It comprises 24 outputs and 21 inputs.
You can only use a part of input/outputs if you want to do easy programmes

MAIN FEATURES :

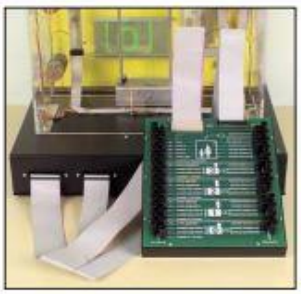
- Opening and closing of the doors on each floor is done by electric servo motors.
- The rear of the lift is visible through the sides and the bottom which are transparent
- The route of the lift is sensed at each floor by a photo-detectors.
- Two limit switches, high & low, (without program control) stop the lift if there is an error in the program.
- All of the buttons and switches are fitted with de-bounce circuits.
- The outputs are protected against the possibility of a short-circuit.
- The rear sliding door is of a transparent Plexiglass design and there is no manual access possible, as there is risk of damaging the servomotor.
- The mechanical controls are sturdy and can withstand any likely faults.

ref. ASC89-24 LOGIC ON 24V

ref. ASC89-05 LOGIC ON 5V

4 LEVELS EACH LEVEL HAS	1 electrically opening door - 1 photo-detector for 'door closed' - 1 photo-detector for 'door-open' 2 safety limit switches for door open/close (No control from the program possible) 1 button to call the lift 'up' (except the 3rd floor) with indicator lamp. 1 button to call the lift 'down' (except the ground floor) with indicator lamp 1 lamp to indicate the presence of the lift - 1 photodetector to indicate the presence of the lift
CONTROLS INSIDE THE LIFT	4 buttons for each floor - 1 stop button 1 switch to simulate a blocked door 4 lights for each floor - 1 light inside the lift (simulating the lighting)
UNIT SUPPLIES POWER TO	the motors - the LED - internal logic to the unit.
OTHERS SPECIFICATIONS	Dims 780 x 480 x 440mm Weight 15kg Supply 220V The unit is available in two driving logic values, 24V or 5V.

OPTION INTERFACE FOR DIDACTIC LIFT



ASMAT is an interface allowing a quick connection to the ASC89 lift from a PLC. To help in quickly identifying the functions of the connectors, small symbol and piece of text next to each connector, allowing immediate understanding of its function. The operating sense: Lift - PLC or PLC - Lift. It is clearly indicated by vertical arrows.
Metal box: 22x272x32mm. Weight: 250g.

CONNECTION ASMAT - LIFT	CONNECTION ASMAT - PLC
Two flat cables are fitted. One connector for the inputs and the other for the outputs.	The front face of the board has two columns of 4mm plugs, which are used to connect to the PLC with normal leads. The plugs on the left are for the inputs to the PLC, on the right for the outputs.

ref. ASMAT

OPTION PLC FOR DIDACTIC LIFT

AUTOMASC is delivered in a plastic box including:

- a 30 inputs / 26 outputs PLC (dry contacts)
- an interface for the lift
- supplies for the PLC outputs
- All cables to the lift, mains cord.

AUTOMASC is connected to the lift rear connectors with 2 flat cables, one for inputs, the other one for outputs.

PROGRAMMING
User can program AUTOMASC with 2 languages: Instructions list or contact language. You can program it from PC, using the Télémécanique® TWIDO suite software (included)

OTHER FEATURES
The front panel is transparent to see many LED, showing the PLC state. AUTOMASC is supplied with a demo program, which can be modified or completed. The technical leaflet indicates the corresponding between the lift inputs and outputs and the ones of the PLC, allowing the development of a complete program.
Mains: 220/240V - 50Hz - 50VA
Dimensions: 350 x 190 x 170 mm
Weight: 2.7kg

ref. AUTOMASC

CE

7.2 Description of the Code

The development of the code is strictly related with the flow charts and the modelling through the finite state automatons reported in section 4.1.4 and 4.2.3. Given the structure of the modelled system, the code is divided using several blocks and functions that allows an understanding of the code and therefore it is easier to find faults related to specific components of the system. The method with which the code was written makes it possible to have a margin of adaptability if the system requires to be enlarged.

7.2.1 Functions definition

To have a better understanding of the code, all the functions that were implemented are described below.

CabinSensors: Virtual sensors are created to develop all the virtual simulation outside the plant, because these sensors are a fundamental part of the logic and they are also synchronized with the visual design of the system within the program.

CURRENT_FLOOR: Gives the last floor where the cabin was placed according to the sensors given on each floor, also manage the cabin led on when the cabin is placed at the doors of the floor and off in any other case.

DATA_IN: The communication bridge between the real word inputs (Sensors, Buttons) connected to the PLC and the input variables described through the data types is managed by this program, This data assignment allows a better organization of the code and also provides the main structure of adaptability and scalability of the code since the core of the data structure is always the same.

DATA_OUT: This program is the counterpart of *DATA_IN* while the previous one manages the input variable this is the one who assigns the controlled variables to the plant outputs (Actuators). However, this maintain all the properties and advantages as the previous. The combination of these programs allows the implementation of basic cyclic functions such as FOR function.

Desired_Floor: The management of the calls is done through the SCAN Algorithm where the desired floor it is decided at any time instant following different paths of behaviour depending on the conditions of the elevator. For instance, the function declares a flow direction according to the first call that is made, and the program focuses on a specific part of the queue where the calls of that directions are stored. Then the nearest call is chosen, while the elevator is moving, the function is constantly searching for a new nearest call is declared and it is done until the elevator reaches the maximum or minimum value of the specific part of the queue, after if there is another call in the other direction, the flow is changed and the process is repeated until there are no calls in the queue, at this moment the assignment of the flow returns to a standby state.

Door_Closing: It manages the closure of the door after the door is opened and a timer flag is triggered. If the obstacle sensor is activated while the door is closing, the movement of the

doors is stopped. When the door reaches the closure position and the sensor is activated a timer that manages the occupied state and the cabin light is triggered in case that the stop button is pressed the closure condition changes according to that so a different management of the flags is executed.

Door_Opening: Similarly, to the *Door_Closing* function it manages the door opening operations for normal conditions and when a cabin obstacle is detected. When the door is fully opened the motors are shut down and a timer for the closure of the door is activated.

InCaseOfEmergency: Determines the behaviour of the elevator when the stop button is pressed depending on the conditions in which the elevator is at the moment. It is declared as a PRG block since this functionality has priority over the others because is the main safety measure available in the system. The safety measure is divided in three different scenarios that are described as:

- *Floor presence same as desired floor:* The cabin is at the desired floor and the door maintains opened until the emergency is over
- *Floor presence different from desired floor:* The cabin is not at the desired floor, but it is present at the doors of another floor, in this case, a way of scape is provided opening the doors where the cabin is standing.
- *Between floors:* The cabin is standing between floors, so the action taken is to stop de motion motors and wait until the emergency is over.

A memory of all the output variables is taken at the moment where the scenario is chosen so after the emergency is over the elevator returns to the normal operation with all of the data saved.

> ***BlinkingLights:*** Right after the scenario is chosen a set of blinking functions are activated so the available leds of the elevator behave according to this function, so the user notice that the elevator is not working in normal conditions.
>
> ***EmManagementAtFloor:*** If one of the two first scenarios is chosen this function is called and manages the opening of the doors and stop the motion of the cabin also calls the *BlinkingLights* function.
>
> ***EmManagementBetweenFloors:*** If the third is chosen this function is called and stops the motion of the cabin and calls the *BlinkingLights* function.
>
> ***RestartOP:*** Restarts the previous operations after emergency elapsed.
>
> ***ReturnLightsState:*** Resets the lights according to the memory data that was taken at the beginning of the emergency.
>
> ***StopBlinkingLights:*** The blinking functions set at the *blinkingLights* functions are disabled

The emergency is elapsed when the stop button condition is set to FALSE.

INIT: Initializes the elevator with a set of preestablished conditions that guarantee the correct functioning of the program irrespective of the previous settings declared in the plant. First, each available door is opened until the sensors detect all the doors are opened, then for each floor all the leds and outputs are set to FALSE while the door are being closed, at the same time, the cabin light are turned off and the door variables are reset, if the cabin is not present

at the ground floor, the movement its activated. Once all the doors are completely closed and the cabin is at the ground floor, the variables related to the operation are set to nominal values and after all conditions are fulfilled, the elevator is ready to be operated.

Logic_Switch: Manages the change of logic between the simple and the optimized / SCAN this is a security measure to manage the change of control, it is established that the change is made only if the cabin is still on a floor or if the elevator is stopped in correspondence with a floor.

Movement: Determines the position of different objects on the visualization panel related to the virtual simulation environment. The objects are changed according to the activation/deactivation of the output variables of the elevator (Door_Open/Close, Cabin_Up/Down).

PLC_PRG: This is the main PRG of the code, it manages the call of all the functions described in this section, it does the initialization and input assignment, then according on the chosen logic checks for floor or cabin calls, depending on the desired floor the cabin is moved or not, all the leds are set up and the elevator is declared occupied. When the cabin reaches the desired floor, the motors and the respective leds are turned off and the flag to manage the doors are set. The output and input assignment are done irrespective of the conditions of the elevator.

Queue: Every time a call button is pressed, the call is added to the corresponding queue with FIFO logic**.** There are two queues to manage all the available calls on the plant, the first one is related to the cabin and the second one is related to the calls on the floor, this second queue is subdivided in two parts, one for the up calls on the other for the down calls, the queue is sorted such that all the empty values are at the end of the array. When a floor has been served and the door management is done, that floor is eliminated from the queue depending on the direction of the movement. This function is only related to the optimized multiple call management logic.

Queue_Led_Reset: When the logic is changed from the optimized multiple management to the single call management and there are pending calls in the queues, all the values are reset and therefore all pending leds are turned off.