11/01/2021

THE ELEVATOR

Control System Technologies Project

Edoardo Panichi – Lapo Carrieri

INTRODUCTION

An elevator is, nowadays, a machine that can be found either in the newest skyscraper or in a little city building. Nonetheless the functioning in these two situations can be quite similar. One fundamental component in order to achieve the right behavior of the lift is a good Logic Controller. Our project tries to achieve this objective using the SFC language jointly with ST language, all implemented in the Codesys environment.
The project has four GAs (DoorGA, MotorGA, VelocityGA and CalculationGA) commanded by a Policy.

ACTIONS	SENSORS	ACTUATORS	TYPE	GA
Closing door	Closed, Presence	Closing	DO-DONE	DoorGA
Opening door	Open, Presence	Opening		
Slow	RampSensor	Vel	START-STOP	VelocityGA
Fast	RampSensor	Vel		
Moving up	LimitUp, DeckSensor	Up	START-STOP	CalculationGA
Moving down	LimitDown, DeckSensor	Up		
Moving	NONE	Motor_on	DO-DONE	MotorGA
Staying	NONE	Motor_on		

POLICY (initialization)

The policy immediately calls VelocityGA asking for the Initialization procedure. In this way we are sure to operate with LOW SPEED for all the whole initialization procedure.
Then it can call DoorGA to close the doors.
Now it calls the initialization path of the CalculationGA that is divided in two parts:

1. The first part sets UP as FALSE because we need to bring the elevator to the bottom first.
2. The second part (it starts when the lift meets LimitDown) sets Current to -1 as the lift meets two deck sensors before reaching the 1st floor. Moreover, the value of UP has changed as now we need to go up.

Meanwhile the initialization path of the MotorGA is turned on: the variable Motor_on becomes true until the lift reaches the first floor.
Meanwhile CalculationGA is activated; it starts and never stops. It continuously calculates the current floor and the destination floor, and it sends the result to the policy and to all the other GAs.
When the lift has reached the first floor (calculated by the CalculationGA) the motor is turned OFF and then it opens the doors.
Now the initialization is finished, so the policy calls VelocityGA (this GA will continue working as long as the elevator works) and starts it in its "Nominal Functioning".

POLICY (Nominal functioning)

After the Waiting block the policy is divided in four branches thanks to a parallelism.
The first branch is an isolated and cyclic loop that controls the MotorGA and the DoorGA in the right order based on the logic sequence of the actions. The sequence is:
- Door_close
- Motor: perform two main actions: turn the motor ON to reach the destination, then turn it OFF giving the signal of DONE to the policy
- Door_open
All these blocks run cyclically as long as the lift is operative.

The other three are launched and, as for VelocityGA and CalculationGA, they work cyclically at every moment. These blocks are:

ARRAI: controls the calls from the simulation lift recording them in three arrays of seven cells in order to maintain the signal during the time. The arrays are: L (for calls internal to the lift), A_up (for the up arrow external calls), A_down (for the down arrow external calls).
These arrays are sent to the CalculationGA that checks in a specific order all the registered calls and calculates the destination floor.
Also, the same block dissimulates a call from the 6th and the 1st floor when we discover respectively that the 7th floor and the Ground DeckSensor have some trouble (and so FaultSensorAlarm is turned ON and we have reached the limit sensors).

LED: Turning on the right LED is achieved just copying the value of the arrays in the Vis variables. The Emergency LED is turned on according to the value of E_counting (active if we are in emergency).

Delete_array: Thanks to this block the values stored in the arrays are deleted when the lift has reached the right floor (and so the users' call is satisfied). Every time the lift arrives to a floor we eliminate:

1. The array value of the button inside the lift corresponding to the current floor.
2. The array Updeck/Downdeck value corresponding to the current floor if the lift is going Up/Down.

When a Deck Sensor is broken, we cannot serve that floor, so all the calls coming from the unavailable floor (identified by Broken1) are immediately deleted.

VelocityGA

VelocityGA is a START-STOP GA, it has the goal of changing the velocity of the lift based on the Ramp_Sensor and the Destination of the lift.
Here we have two important blocks “StopORGo”, “Ramp_Range” and “Ramp_Fault”:

- The first block calculates whenever we are inside or outside the two ramp sensors around the deck sensor of a floor. It is very useful because it creates a variable (RampRange) that every time specifies if we are in the range near the DeckSensor. (RampRange changes its value only when a RampSensor becomes true, whereas it is not affected when RampSensor becomes false.)
- The second one is useful to control directly the velocity; it slows down the lift only in case it is arriving at the ramp sensor just before the destination floor. Then the velocity will increase again when we will get out of the RampRange.
- The third block is used to detect a fault of a ramp sensor. It is very similar to the Deck_Fault block. It turns on the RampSensorAlarm LED and it allows the elevator to keep working at both levels of speed (with some limitations, see comments along the code for a deeper analysis). The block is able to detect a ramp fault if the lift reaches a Deck_Sensor with count_ramp=1. Indeed, this situation can occur only if there is a broken RampSensor. So, when the lift overcomes the DeckSensor, RampRange is set to TRUE because this should have been done automatically when the RampSensor was detected. If a call is coming from a floor that is not affected by the malfunctioning of the sensor, then the behavior of the lift is not changed. If the call comes from a floor, such that the elevator, due to the absence of a ramp sensor, cannot slowdown on time, then the elevator goes to the closest available

floor. Actually, if a RampSensor is broken the affected floors are still reachable from the opposite direction (not true for the 7th floor and the Ground floor).

DoorGA

DoorGA is a DO-DONE GA, its aim is to handle the procedure of opening and closing of the door. To achieve this goal, it uses three sensors (Presence, Closed and Open) and two actuators (Closing and Opening). The two basic actions associated with this GA are identified by an alternative choice where, according to the request of the policy, expressed through Do_What, the two different actions can take place. Instead, a fundamental task of this GA is handled thanks to a parallelism used at the beginning that stays active forever. This branch continuously checks for people in the middle of the doors. If a person is detected, then the closing procedure is immediately stopped, and the doors are completely open. If nobody is now in between of the two doors the closing procedure can start again.

MotorGA

MotorGA is a DO-DONE GA, it is activated by the policy every time the doors have been closed and it sends a done to the policy when the motor is turned off.
The aim of this GA is to handle the switching OFF and ON of the motor that moves the elevator. To achieve this goal, it controls the actuator Motor_on. The GA is also aware of the value of the actuator Vel and of the sensor DeckSensor, both of these information are not taken directly from the actuator and the sensor, but from a copy of their values into two new variables in order to respect the rules the GA approach.

The GA is structured with an alternative choice that allows the Policy to select between the nominal functioning or the initialization procedure.
The motor has to be turned ON every time that the lift performs a movement, and when this displacement is done, so it has reached the destination, the motor is turned OFF. So, to achieve the behavior just described when the Policy calls the MotorGA both actions are performed before sending a DONE message back.

It is important to understand the transitions that allow turning ON and OFF the motor: the motor can be turned ON only if we are not in emergency AND only if the calculated current position is different from the calculated destination; then the motor is turned OFF in two situations, both valid only if the velocity is SLOW and DeckSensor is TRUE:

1. The current position equates the destination. That means the destination has been reached.
2. The 'E_counting' is equal to 1 and so we are in emergency. In this case the point 1 could fail because, as soon the new DeckSensor is reached, the Destination Block, that is still working considering the emergency mood (E_counting=1), calculates a new destination (temp_destination = GVL.Current + 1), even if actually we need to stop now. This happens even if it would have stopped at that floor anyway.

CalculationGA

The aims of this GA are: calculate the current position, the next destination and detect a possible DeckSensor Fault. To achieve these goals, it uses three sensors (LimitUp, LimitDown and DeckSensor) and an actuator (UP). The GA is also aware of the value of two other actuators (Vel and Motor_on) and also of another sensor (Ramp_Sensor), but these information are not taken directly from these other actuators and the sensors, but from a copy of their values into new variables in order to respect the rules the GA approach.

As all the tasks that this GA handles must be continuously checked and executed, the code is structured with a parallelism of four blocks that jump inside themselves. In this way every PLC cycle all these blocks are calculated and we avoid any delay in updating the variables.
Of these four blocks, only three are valuable to be mentioned: Current, DeckFault and Destination.

In “Current” the variable I_Current is updated according to the moving direction (up or down) when a deck sensor is detected.

In “DeckFault” a possible broken DeckSensor is handled; if this happens the lift
keeps working but it cannot serve any call coming from the floor where the DeckSensor is broken. The detection of the problem is done counting the number of RampSensors met without meeting a DeckSensor, if this count (variable count_ramp) reaches three then we have a broken DeckSensor. Particular cases are those in which the broken DeckSensor belongs either to the Ground floor or to the 7th floor, in these cases the detection is done thanks to the LimitUp and LimitDown sensor. When the GA detects a fault immediately updates, according to the moving direction, the current position as usually this is done thanks to the detection of the DeckSensor. Moreover, the information of which DeckSensor is broken is stored in the variable Broken_1.

In “Destination” the next stop is calculated. The priority for the calls is determined as follow:

1. If the lift is going up the priority is given to all the calls coming from floors higher than the current position of the lift:
   1.1. The updeck and inside button calls in an increasing array index order (from the lower to the higher floor)
   1.2. The downdeck calls in a decreasing array index order (from the higher to the lower floor)
2. If the lift is going down the priority is given to all the calls coming from floors lower than the current position of the lift:
   2.1. The downdeck and inside button calls in a decreasing array index order (from the higher to the lower floor)
   2.2. The updeck calls in an increasing array index order (from the lower to the higher floor)

Some exceptions to the above priorities are when the emergency button has been pressed, in this case the stop happens at the first available floor, or when a DeckFault is detected and so the floor with the broken sensor must be skipped.
Actually, in the code there are two variables called Destination: Temp_Destination and I_Destination (this is the real next destination); the difference between the two is that I_Destination can be updated with the value of Temp_Destination only if the lift is out of the RampRange (see VelocityGA) or it is still at a floor (motor OFF).
As in this block also the actuator UP, associated to Final_go_up, is managed, a similar code is implemented between go_up and Final_go_up, indeed the latter is update only if the motor is OFF, this because the change of direction can be executed only when we are still.

(Much more information about the code and the variables can be found as comment along the code)