INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA MECÂNICA




Electronic Redesign of an Industrial Lift

URTZI AGIRRE ALBIZU
Abril de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

ERASMUS PROJECT

Electronic Redesign Of An Industrial Lift

Author

URTZI AGIRRE ALBIZU

26.FEB.2016

Acknowledgements

In first place, I would like to thank to Professor Francisco J. G. Silva and Professor Luis Pinto Coelho for guiding me through this project until the end. Also, I would like to thank to all the company ABER Ltd, for the opportunity of developing this project in the company, and particularly to Ricardo J Bernardes, Vitor Alves, Rui Silva, Rui Fernandes and Rúben Marques for helping me and for the company throughout all these months.

Abstract

This project, realized at the company ABER Ltd, describes the process followed for the developing of an electronic control system for a hydraulic elevator. The previous control system was based on relay logic, and the company wanted to change it to a microcontroller based technology. To do so, different approaches were studied and finally the selected technology for the development was the Raspberry Pi. After, the software needed for all the elevator types was developed, and the interface hardware was selected. In the end, several test were made to adjust the software and the hardware and to prove the good operation of the system.

Keywords

Hydraulic, elevator, electronics, control, microprocessor, Raspberry Pi

List of Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

AC	*Alternating current*
ALU	*Arithmetic logic unit*
ARM	*Acorn RISC Machine*
BJT	*Bipolar junction transistor*
CPU	*Central processing unit*
D	*Diode*
FC	*Limit switch*
FCA	*Upper limit switch*
FCB	*Lower limit switch*
GPIO	*General purpose input output*
HDMI	*High-definition multimedia interface*
I2C	*Inter-integrated circuit*
IDE	*Integrated development environment*
LED	*Light-emitting diode*
MOSFET	*Metal-oxide-semiconductor field-effect transistor*
OS	*Operative system*
PC	*Personal computer*
PCB	*Printed circuit board*
PLC	*Programmable logic computer*
PTO	*Power take off*
Q	*Transistor*
R	*Resistor*
RAM	*Random-access memory*
ROM	*Read-only memory*
RPi	*Raspberry Pi*
SD	*Secure digital*
SW	*Switch*
SWOT	*Strengths weakness opportunities threats analysis*
UART	*Universal asynchronous receiver/transmitter*
UC	*Optocoupler*
USB	*Universal serial bus*

V	*Volts*
VAC	*Volts alternate current*
VDC	*Volts direct current*

1. INTRODUCTION

Company characterization

The problem

Methods

1. Introduction

1.1. Company characterization

ABER, situated in Maia, Portugal, is a company that manufactures hydraulic products. ABER is an expert in the domain of the oil-hydraulic and in the transmission of power applied mainly on trucks (cranes and tippers). ABER manufactures a wide range of products:

- Hydraulic Pumps
- Hydraulic Motors
- Power Take Off's PTO
- Hydraulic Valves
- Pneumatic/Mechanic Controls
- Tippers Scissors
- Hydraulic Power Packs
- Oil Tanks
- Wet Kits
- Hydraulic Accessories

Apart from this, the section ABERMOVE manufactures lifting systems

- Parking systems
- Housing systems
- Industrial systems
- Urban Waste Collector Systems

Figure 1: ABER logo

1.2. The Problem

As told before, ABER Company does, among other things, hydraulic elevators. These elevators are manufactured customized to the client requirements, so every one of them will have different type and quantity of sensors and actuators. The operating principle of these elevators is a hydraulic system, comprising one or more hydraulic cylinders, a pump that applies the needed pressure so as the cylinder can rise, and an electrovalve that lets the hydraulic fluid to get out of the cylinder and makes the elevator descend by gravity.

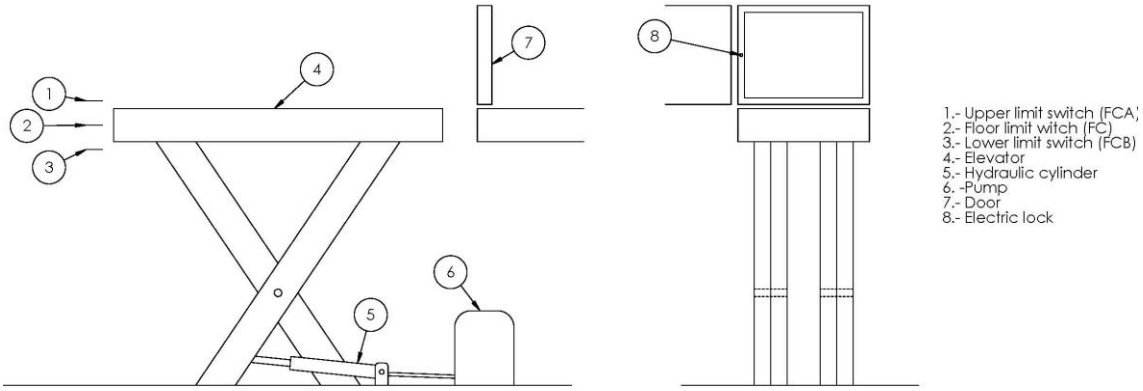


Figure 2: Elevator scheme

At the same time, the hydraulic circuit is controlled by an electric circuit. This one is based on ladder logic, carried out by some relays and sensors. These elements control the position and the movement of the elevator, as well as the state of the doors.

This technology has been used along the last decades and it is widely supported, although it has some disadvantages comparing to more modern technologies:

- Price: Component's price is quite high, specially comparing to integrated circuits.
- Size: In addition to the components being large, the required number of components for minimally complex applications is really high.
- Flexibility: As the logic is made at hardware level, any change in the logic implies changing the hardware. This means that every different machine needs different hardware.
- Limitations: This technology does not support any kind of informatics implementation, so the innovative capabilities are very limited.

All of the above makes it difficult to standardize the control system and to add new features, so the ABER Company has decided to change the control system of its elevators.

1.2.1. Company needs

To make this possible, company needs must be taken into account. ABER manufactures elevators with different characteristics, which can be the following:

- One or two descending speeds
- One or two ascending speeds
- Two or three levels
- Auto-levelling system: This system automatically realigns the elevator when a charge forces it to descend
- Hydraulic lock system

Moreover, the company manufactures other systems similar to a hydraulic elevator, so they wish to adapt this project to those systems.

The company has predicted a production of 70 elevators per year, mostly two level elevators, with one ascending and descending speed and without auto-levelling system.

Also, the ABER wants to integrate informatics applications to the elevator, such as telematic control though the mobile phone or creating usage reports.

1.2.2. Objectives

The main objective of the project is designing a control system that is capable of managing the elevator and has the following characteristics:

- Standardize the hardware used for the controlling system.
- The system must be able adapt to new type of elevators, with the lower amount of work possible.
- Allow implementing new features and innovations to the system.
- Reduce the price as much as possible.

1.3. Methods

The first step is to analyse the current situation of the control electronics, so as to find out what technologies are used nowadays, and evaluate their pros and cons. Afterwards, the first solutions are proposed, based on previously studied technologies. Once the options are presented, one of them must be selected and developed in depth.

Development is divided in two sections. The first one is hardware, which consist in designing the required hardware to manage the voltage level of the main circuit. Secondly, there is the software design, which consist in designing the logic of the system and the programming needed so as this logic can be executed by the microcontroller.

Once development is finished, it should be tested in an actual elevator and needed adjustments should be made. Last step is to write a memory including all the information gathered during the project, and explaining the process followed and the conclusions reached. To organize the process, it has been divided in the tasks written bellow, and a Gantt diagram has been made to control the timeline:

1. Problem analysis
2. Studying possible options (Brainstorm)
3. Selection process
4. Design
   4.1. Hardware
   4.2. Software
5. Evaluation and analysis
6. Solution evaluation
7. Documentation

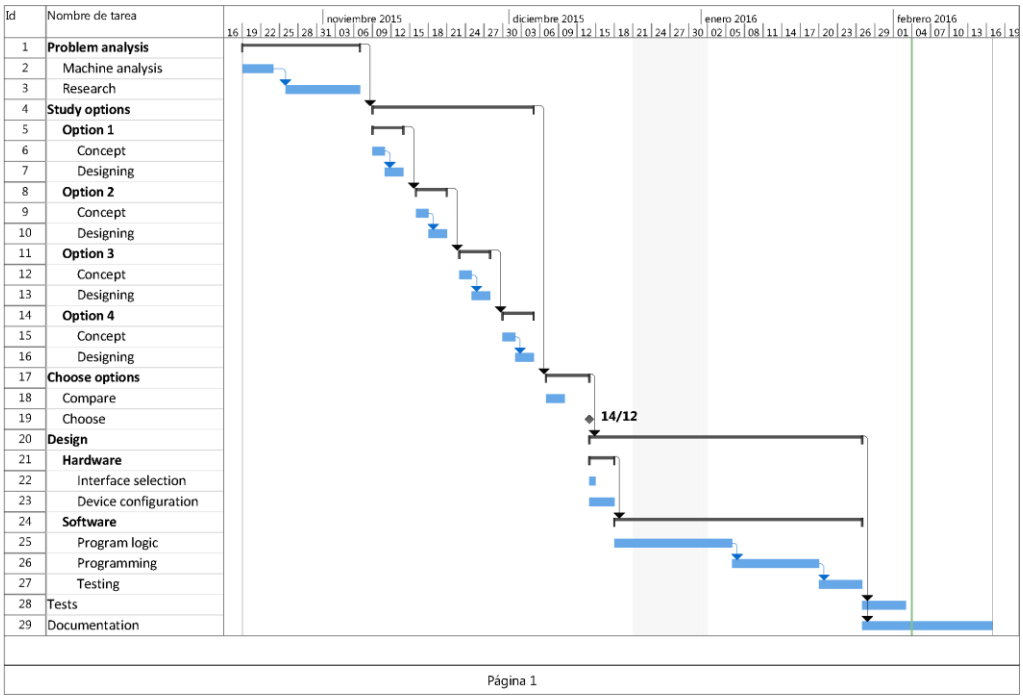


Figure 3: Gantt diagram

2. BACKGROUND

System description

Automation technologies

State of the art

2. Background

2.1. System Description

2.1.1. Components

In order to describe properly the problem, first a description of the elevator components will be made. It will be divided in the hydraulic and the electric circuit.

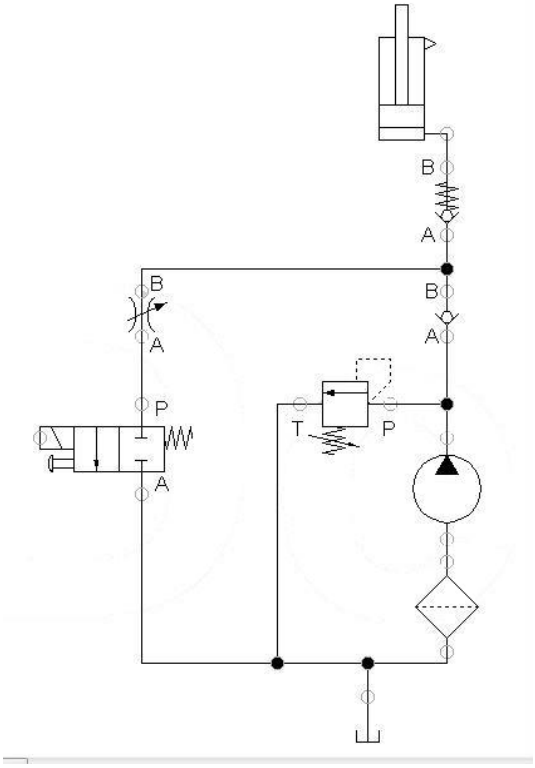


Figure 4: Hydraulic circuit

- Discharge valve: The discharge valve is responsible of letting the hydraulic fluid flow from the cylinder to the tank, making the elevator descend. It is a monostable 2/2 normally closed valve, electrically controlled, with manual control for emergency situations.

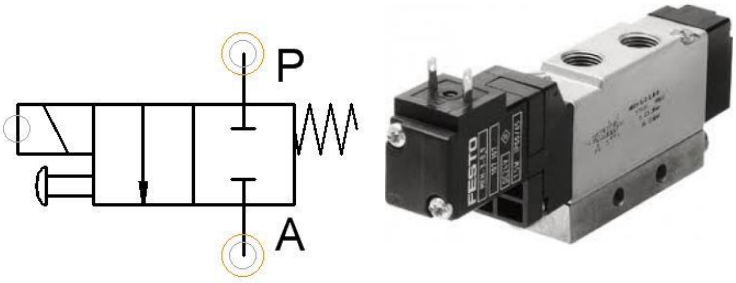


Figure 5: Discharge valve (www.contranspower.com)

- Pressure limit valve: The pressure limit valve is responsible of maintaining the system pressure below safety limits, and it is situated after the pump.

Figure 6: Pressure limit valve (www.austworld.com)

- Hydraulic cylinder: The simple effect cylinder is the final element of the circuit that transmit movement to the elevator. When the elevator has to go up, the hydraulic fluid enters the cylinder, whereas when it has to descend, the weight of the elevator forces the fluid out of the cylinder.

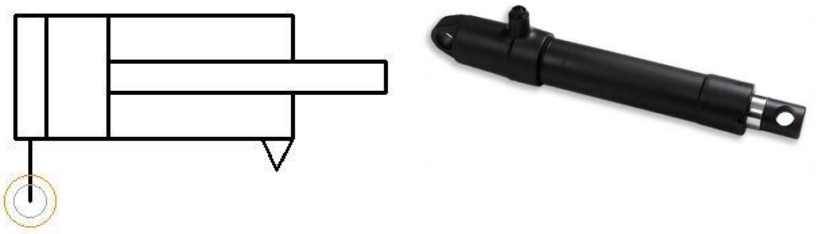


Figure 7: Hydraulic cylinder (www.pedro-roquet.com)

- Pump: The pump is the element that pushes the hydraulic fluid through the circuit, transferring the work to the cylinder.

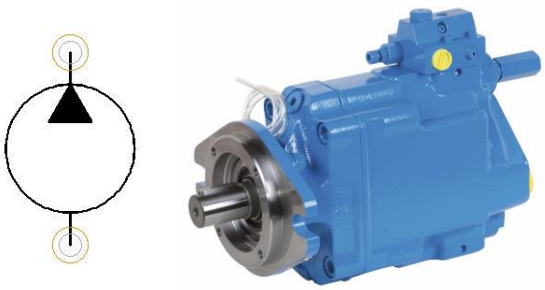


Figure 8: Pump (img.directindustry.es)

- Check valve: The check valve only allows fluid to flow in one way, preventing it to enter the pump from its outlet

Figure 9: Check valve (www.allprosperity.com)

- Tank: The tank is the place where the hydraulic fluid is stored, from where the pump takes it and where it goes when it leaves the cylinder.

Figure 10: Tank (aber.es)

The electric part is divided in power circuit, which supplies power to the transformer and the motor, and the control circuit, which is supplied by the transformer and its task is to operate the elevator. The power circuit is directly connected to the main electricity source, which is 230 VAC, and the control circuit is connected to the transformer, that supplies 24 VDC.

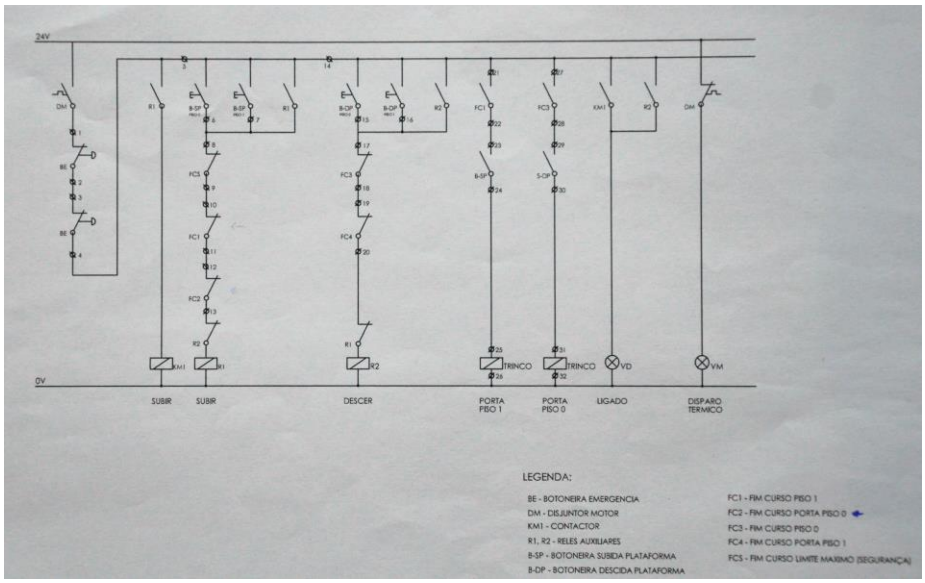


Figure 11: Electric circuit

The components of the circuits are the following:

- Limit switch: This sensor is usually activated by moving parts of a machine, with the purpose of detecting its specific position. This case is used for detecting when the platform arrives to the floor, detecting changes in the levelling and the door closing movement.

Figure 12: Limit switch (www.tracepartsonline.net)

- Electromagnetic lock: An electromagnetic lock is a locking device that works with an electric signal. In this case, the lock is used for locking the doors when the elevator is not in the floor.

Figure 13: Electromagnetic lock (products.dorma.com)

- Motor: It is a monophase AC motor, connected to the 230 V line. Its function is to activate the pump. The motor is controlled through a contactor.

Figure 14: Motor (roydisa.es)

- Contactor: The contactor is a mechanical switch operated electrically used for high voltages and power, being controlled by signal with low voltage and current intensity.. In this case it is used to control the motor.

Figure 15: Contactor (kiirux.com)

- Buttons: Buttons are user-activated switches, used for choosing where to move the elevator.

Figure 16: Buttons (static.grainger.com)

- Transformer: The transformer converts the input voltage of 230 VAC to the control voltage of 24 VDC

Figure 17: Trasnformer (www.cetronic.es)

- Security systems: The security systems are placed at the beginning of the control line, so that the circuit can be completely closed in an emergency.
    - Emergency button
    - Circuit breaker

2.1.2.Auto-levelling

Auto levelling system consist in adjusting the level of the elevator when it is moved by external forces, for example, adding or removing load. There are three signals that are used to detect the unbalance. FCA is a limit switch that detects when the elevator is higher than it should, FCB detects when it is lower, and FC detects the correct level.

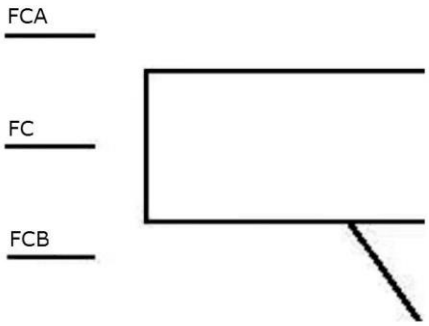


Figure 18: Auto-level sketch

2.1.3.Hydraulic locks

The hydraulic locks system is another kind of auto-levelling, but in this case the platform rest on the lock, as seen in the image. This locks are extracted by hydraulic power. To control this system two limit switches are needed: FCA is the place where the platform waits to the lock to be completely out, and FC detects when the platform is resting on the lock.

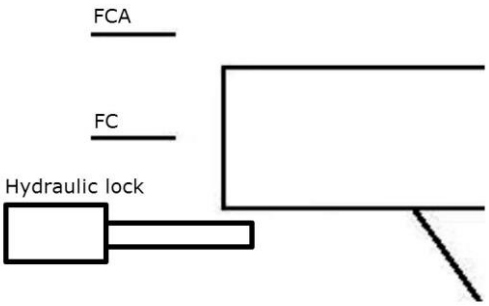


Figure 19: Hydraulic locks sketch

2.1.4. Two speeds

Elevators with auto-levelling or with hydraulic locks can manage two different. The two pumps are connected to the same motor, and the second is bypassed by a valve, as seen in the image. For the ascension the motor is connected. To change between speeds, the bypass valve is activated or deactivated, allowing the power of the pump to be transmitted to the cylinder.

For descend, two valves in parallel are connected. In this case, to change between speeds one or two valves are opened. This increases the discharge flow, which means faster speed.

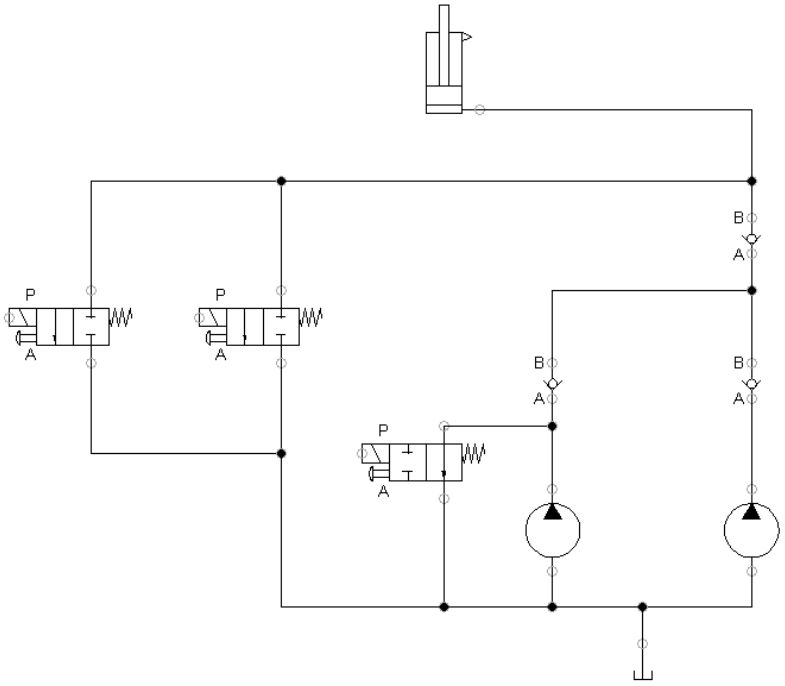


Figure 20: Two speeds scheme

2.2. Automation technologies

2.2.1. PLC

Before modern automation technologies, controlling systems were mainly composed of dedicated relay circuits. This means huge relay panels with hundreds or thousands of relays, which makes it very difficult to update, because every relay had to be rewired. Also, maintenance is a tough task, because to check where an error is the operator has to check all the components and electric connexions. This situation changed in the 70s, when the PLC started to work

2.2.1.1. Hardware

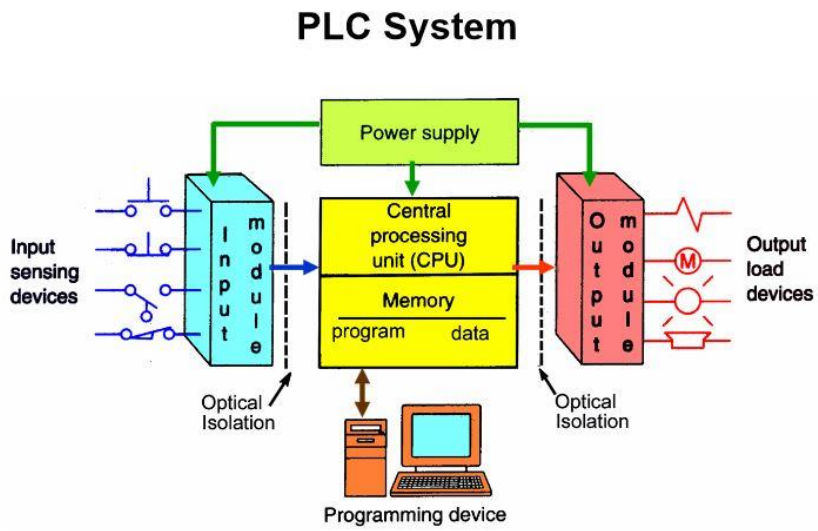


Figure 21: PLC hardware

- Input module: The input module connects the input terminals to the rest of the system. Each terminal is usually electrically isolated from the internal electronics by optoisolators. These modules. This is a way of passing on the status of the input (on or off) by use of a light emitting diode and phototransistor. A typical optoisolator is shown. They have the advantage of reducing the effects of spurious pulses generated from electromagnetic sources. It is also a safety feature to prevent live voltages appearing on the input lines in the event of a fault.
- Output module: The output module contains switches activated by the CPU in order to connect two terminals and so allow current to flow in the external circuit. This will activate devices such as pneumatic solenoid valves, hydraulic solenoid valves, motors, pipe line valves, heating elements and so on. The switch may be a transistor or a relay.

- Memory: The PLC has RAM and ROM. The programme, when written and entered, is stored in the RAM. The ROM contains permanent programmes such as that required to monitor the status of the inputs and outputs and to run diagnostic tests.
- Power supply: It gives the needed power so the CPU and input and output modules can work.

2.2.1.2. Software

The PLC is programmed with logical commands. This may be done through a programming panel or by connection to a computer. Computers are able to run programming software with graphics, simulators, diagnostics and monitoring. This could be a laptop carried to the site or a main computer some distance away. Often the programme is developed and tested on the computer and the programme is transferred to the PLC.

The most common programming language is ladder language. This language represents a program by a graphical diagram based on the circuit diagrams of relay logic hardware. The reason for representing sequential control logic in a ladder diagram was to allow factory engineers and technicians to develop software without additional training to learn a language, as these workers where used to working with relay circuits.

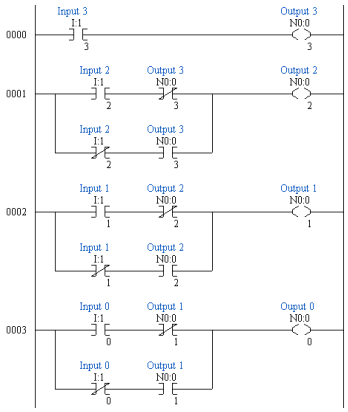


Figure 22: Ladder language example

2.2.2. Microcontroller

A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input and output peripherals. Program memory is also often included on chip, as well as RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications consisting of various discrete chips.

2.2.2.1. Hardware

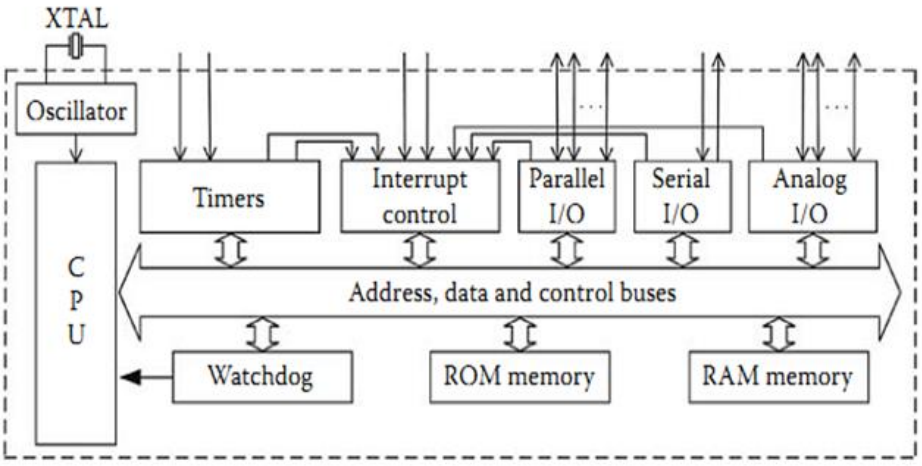


Figure 23: Microcontroller hardware

- Analogue to digital converter: Converts analogue signals to digital so the microcontroller can manage it.
- Bus: The bus is the group of wires that serves a function. Typically, there are three: address bus, data bus and control bus.
- Oscillator: The oscillator or clock provides a square signal that set the timing of the system.
- CPU: The central processing unit administers all the operations of the system. The CPU is formed by an ALU, several registers and control and synchronisation logic. The arithmetic and logic unit (ALU), is where mathematical and logical operations take part.
- ROM memory: Read only memory is a non-volatile memory where programs are stored
- RAM memory: Random access memory is a general purpose read-write volatile memory used to store temporary data in a program.
- Interrupts: Interrupts allow the microcontroller to react to an internal or external event. For example, a change in an input or output, time related or when an A/D conversion has ended.
- Serial input-output port: This port is prepared to communicate with other devices via serial communication.

- Timers: Timers are used to count time during the execution of a program, for example to execute some subroutine when an amount of time has passed, or to generate a delay.
- Watchdog: The watchdog is a safety feature that controls that time critical modules work properly.

2.2.2.2. Software

Microcontrollers are designed to be programed in assembly language. Assembly is a low level programming language, which is unique to each microcontroller family. It consists on an instruction set that covers the basic operations that a microcontroller can make. These characteristics makes assembly a tough language to program with, because complex operations require a lot of instructions, but this makes it more efficient than higher level languages. This high level languages, like C, also can be used and actually nowadays are more used than assembly, leaving this only for operations that require really good resources management.

2.3. State of the art

To set properly the background, a brief state of the art study has been made.

In “Congestion-Free Elevator Control Using Microcontroller”, developed by Poorvi Behre in 2013, a microcontroller based system is used to control an elevator, with the objective of reducing the congestion of the system.

“Design and Development of a Microcontroller-Based Cargo Lift Control System”, developed by Md. Anwarul Kabir and Md. Abul Hasan Bakar in 2013, states the lower costs of an elevator based in microcontroller control system.

In “Implementation of PLC Based Elevator Control System”, developed by Sandar Htay and Su SU Ti Mon in 2014, the developed the implementation of a PLC based control system and stated the economical and performance advantages compared to a relay based logic controller.

3. DEVELOPMENT

3. Development

3.1. Problem analysis

Considering the company needs, the objectives and the system itself, the following points has been considered:

- Voltage level: Both sensors and actuator work at 24 V. Microcontrollers do not work at this voltage level, so it will be necessary to convert 24 V to a voltage supported by a microcontroller.
- Controller sizing: So as the controller can be used in any of the elevators, it should have enough inputs and outputs to handle all the sensors and actuators. Bellow some tables are shown with the number of inputs and outputs of each elevator, considering the different possible speed configurations referred as U for ascending speed and D for descending speed:

Table 1: Standard elevators IO number

2 floors	1U 1D
Inputs	6
Outputs	4
Total	10

3 floors	1U 1D
Inputs	9
Outputs	5
Total	14

Table 2: Auto-level elevator IO number

2 floors	1U 1D	1U 2D	2U 2D
Inputs	10	10	10
Outputs	4	5	6
Total	14	15	16

3 floors	1U 1D	1U 2D	2U 2D
Inputs	15	15	15
Outputs	5	6	7
Total	20	21	22

Table 3: Hydraulic locks elevator IO number

2 floors	1U 1D	1U 2D	2U 2D
Inputs	10	10	10
Outputs	6	7	8
Total	16	17	18

3 floors	1U 1D	1U 2D	2U 2D
Inputs	18	18	18
Outputs	8	9	10
Total	26	27	28

- Flexibility: As the controller can be installed in different elevators and more innovations are planed around this controller, it is important to maintain as general as possible for future improvements.

3.2. Brainstorming & Preliminary drafts

After analysing the problem, different solutions have been developed. There are two major things to keep in mind, the electronic designing and components that will be used for the interfaces, and the technology used for the controlling system. So, the distribution of this section will be as follows:

- Input interface
- Output interface
- Technologies

3.2.1. Input interface

The main objective of the input interface is to convert the voltage level of the sensor line to the logic level of the microcontroller. This means reducing the voltage from 24V to something near to 5V (this depends on the microcontroller selected). To achieve this, the following options are proposed:

1. Voltage divider:

The voltage divider distributes the supplied tension between two resistors. So in the output I1 will be the required voltage for the microcontroller.

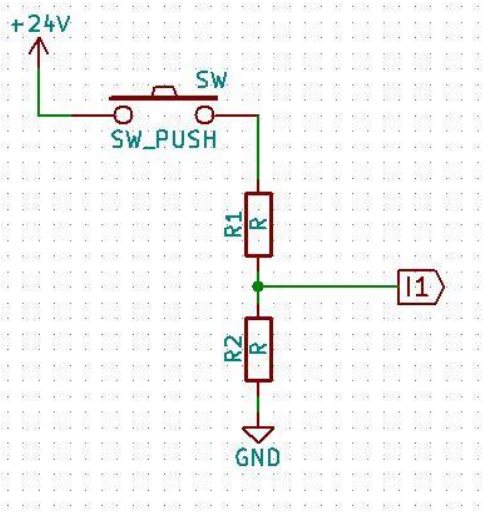


Figure 24: Voltage divider

- SW: The sensor
- R1 & R2: Resistors that divide de tension. The relation of their values is directly related to the voltage division.

2. Optocoupler:

The optocoupler isolates the 24V line from the microcontroller's line, providing better security to the microcontroller in case of a change in the input voltage.

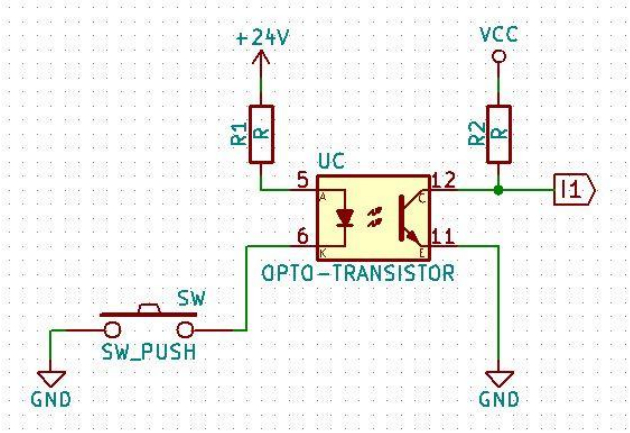


Figure 25: Optocopupler input

- SW: The sensor
- UC: The optocoupler
- R1: Resistor that limits the current through the LED of the optocoupler
- R2: Resistor that limits the current through the transistor of the optocoupler when it is conducting and the current that reaches the microcontroller when the optocoupler is not conducting.

3.2.2. Output interface

The aim of the output interface is to convert the output signal of the microcontroller to the working voltage of the actuators, which is 24 V. In this case, as the actuators have solenoids, it is necessary to consider their discharge current and protect the microcontroller from it. With this in mind, the proposed solutions are these:

1. Transistor:

In this case the output of the microcontroller is connected to the gate of a BJT transistor or to the base of a MOSFET transistor. The transistor works as a switch, allowing the current flow through the charge.

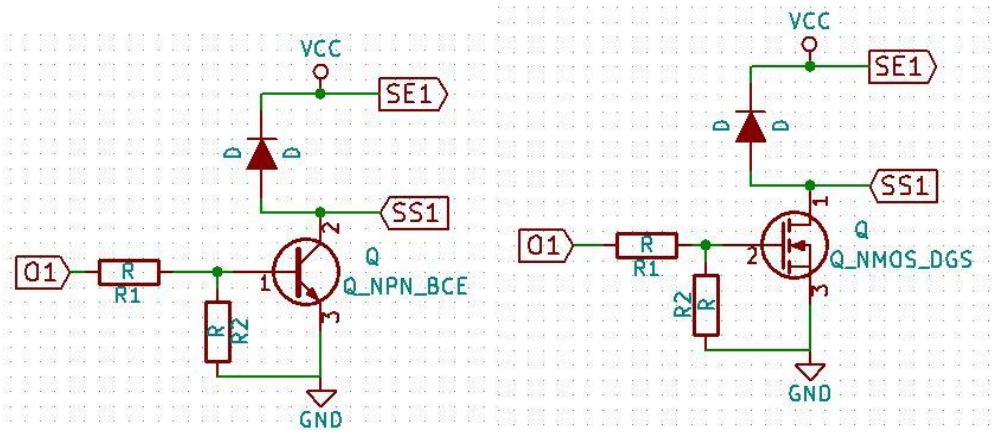


Figure 26: Transistor output

- Q: Transistor that allows current through the charge.
- R1: Resistor that limits the current output of the microcontroller.
- R2: High value resistor that connects the output of the microcontroller to the ground. This avoids problems during the system boot.
- D: Diode that dissipates the energy stored in the solenoid.

2. Optocoupler plus transistor:

In this case there is an optocoupler that isolates the microcontroller and commutates a transistor that at the same time lets the current flow through the charge.

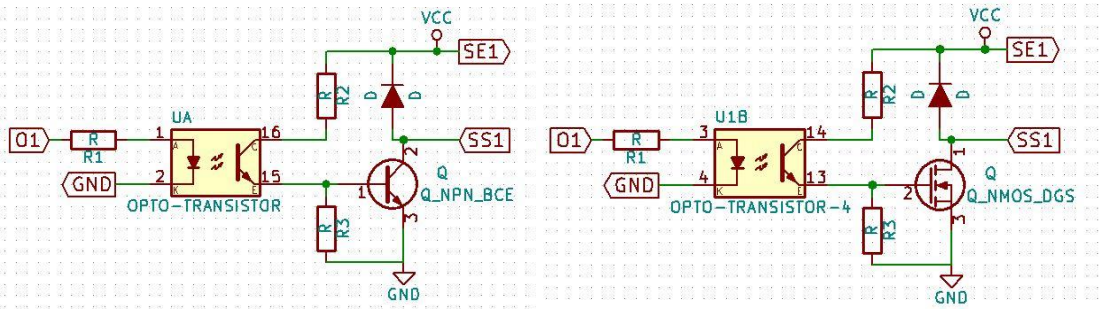


Figure 27: Optocoupler plus transistor output

- Q: Transistor that allows current through the charge.
- UA & U1B: The optocoupler
- R1: Resistor that limits the current through the LED of the optocoupler

- R2: Resistor that limits the current through the optocoupler transistor and the transistor base.
- R3: High value resistor that connects the microcontroller output to the ground. This avoids problems during the system boot.
- D: Diode that dissipates the energy stored in the solenoid.

3. Relay:

In this option the relay isolates the two levels of voltage. Relays are very robust and offer really good electrical insulation, although are larger and more expensive than transistors.

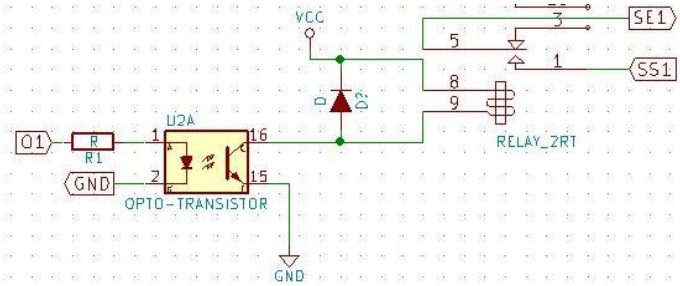


Figure 28: Relay output

- Relay: Relay that allows current through the charge.
- U2A: The optocoupler
- R1: Resistor that limits the current through the optocoupler LED.
- D: Diode that dissipates the energy stored in the solenoid.

3.2.3. Technologies

For the technology analysis, the two systems used nowadays have been chosen: PLCs and microcontrollers. After the proper investigation, these are the solutions achieved:

1. PCB design:

This solution is based on the design of an electronic circuit that is able to handle the input and output voltage so as sensors and actuators can communicate with the microcontroller, which will be responsible of the control, choose the necessary and design a PCB where it will be printed. There are two options for the PCB design:

O1

Design a PCB valid for all the possible elevators, what means designing it with the maximum inputs and outputs that an elevator could have. This leads to a lot of wasted components,

which is an extra cost. On the other hand, it will imply a reduction in cost due to the larger order comparing to the second option, which will be explained bellow.

Also, this option adds an advantage from the viewpoint of inventory, as it reduces the required circuits to one.

O2

As described in the problem analysis, the majority of the elevators are simple. This means that if produced like described in the first option, the most of the boards will have unused components. Considering the predicted demand of elevators, the company will produce 70 elevators per year, 56 of them simple and 14 of other types.

So the approach of this option is to design two kinds of boards, one for the simplest elevator, which will be the majority, and another for the most complex one, like the one proposed in the previous section. With this, the number of wasted components will be much lower, as well as the board size.

The key of this option is to reduce the cost of components, although the production cost will be higher due to the smaller order size.

2. Prebuilt multifunction board:

Other technology that can be used to control the elevator is predesigned microcontroller boards. These boards come with all the necessary hardware to make input and output connections to the microcontroller. One of the most interesting characteristics of this kind of boards is that is relatively easy to add new features, just connecting what is known as “shields”, which are also prebuilt boards that enhances the main board.

Raspberry Pi:

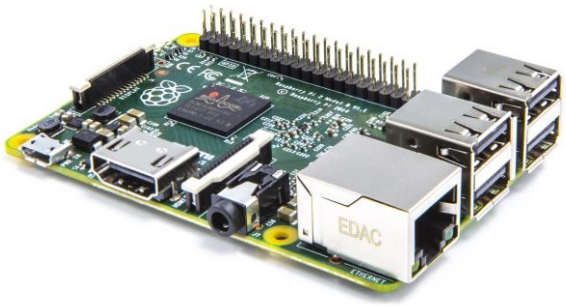


Figure 29: Raspberry Pi

Raspberry Pi is a low-cost Linux computer with the size of a credit card. It supports a large list of programming languages, but the most common is Python. It is driven by a Broadcom

BCM2836 microcontroller, which microprocessor is based in ARM quad-core Cortex-A7 processor cluster.

Raspberry Pi has the following IO ports:

- 40 GPIO header
- 4 USB ports
- 1 Ethernet port
- 1 HDMI port
- 1 3.5 mm audio jack
- SPI Bus
- I2C connection
- I2S connection
- UART connection

Table 4: Raspberry Pi characteristics

Price	40 €
Size	8.6cm x 5.4cm x 1.7cm
Memory	1024 MB
Microcontroller	Broadcom BCM2835
Frequency	900 MHz
On Board Network	10/100 wired Ethernet RJ45
Multitasking	Yes
Input voltage	5 V
Flash Memory	SD card (from 2 to 16 GB)
USB ports	Four
Operative system	Linux
Integrated developing system (IDE)	Any Linux IDE

3. PLC

The last technology considered is installing a PLC. PLCs are specially designed to work in industrial automation processes. The hardware comes completely designed and adapted to work with 24 V signals, and it is highly reprogrammable. The drawback of this technology is its unitary cost, which is considerably high.

3.3. Selecting the best idea

Considering the options written above, advantages and disadvantages will be analysed and the most appropriate will be selected.

3.3.1. Technology

The comparison between technologies have been made in terms of flexibility, innovation capabilities, and price, as this are the stabilised objectives of the project.

3.3.1.1. Flexibility

- PCB: The software is modifiable, although it requires programing knowledge. Hardware modifications needs new design of the PCB
- PLC: The software can be changed and ladder programing is very intuitive. The hardware is prepared to expand it with input and output modules.
- Raspberry Pi: The software is modifiable, but also needs programming knowledge. The hardware can be with more input/output interfaces or with new devices.

3.3.1.2. Innovation capabilities

- PCB: The microcontroller mounted on the PCB is capable of managing a large variety of devices, but it would require new hardware design to do it.
- PLC: PLCs are designed to manage industrial machinery's inputs and from this generate output signals, but if more complex operations are needed ladder logic is not the best language to do it, so innovations capabilities are limited.
- Raspberry Pi: The Raspberry Pi has the sufficient power and the capacity to connect and manage different types of devices that allows the development of new utilities.

3.3.1.3. Price

- PCB: To calculate the budget, an estimation of the components needed have been made. Component price is estimated as an average of price of different manufacturers and suppliers, and PCB manufacture and assembly estimation has been consulted to a PCB manufacturer. The prices have been overestimated, due to the uncertainty of final component prices and layout, in order to in any case the final costs are same or less than here estimated. The resulting price for the PCB design is of 63 € per board.
- PLC: PLC prices varies substantially between brands, but the final price considering the CPU and the necessary input/output modules would not be lower than 200 €.

- Raspberry Pi: The Raspberry Pi and the required interface and power supply cost around 71 €.

3.3.1.4. Conclusions

After analysing the different technologies proposed and its advantages and disadvantages, the Raspberry Pi has been chosen as the most suitable option. Its flexibility and adaptability combined with its low price fulfils all the requirements stated before.

The PLC's price is a huge drawback, because the capabilities that it offers can be achieved with the other two options, but much cheaper.

PCB design is a good option but it has the risk of becoming obsolete if the company wants to implement new features.

Table 5: Technologies comparison summary

	1º	2º	3º
Flexibility	PLC	Raspberry Pi	PCB
Innovation capabilities	Raspberry Pi	PCB	PLC
Price	PCB	Raspberry Pi	PLC

3.3.2. Input interface

The voltage divider is simpler and cheaper because it uses fewer components, although the voltage at the microcontroller input is not so constant and depends on factors like variability of the 24 V line or the resistivity of the sensor connected. This could be a concern if the changes are high enough, being possible that some sensor with high resistivity could generate a voltage drop too large. On the other hand, the optocoupler offers isolation between the 24 V line and the microcontroller input line, and always supplies the needed voltage to the microcontroller, because there is an independent power supply.

3.3.3. Output interface

In the case of the output interface, it is important to consider isolation due to the kind of actuators that will be connected. The possibility of voltage or current peaks is higher, so the optocoupler is a good option. Between the transistor and the relay, the transistor is smaller and cheaper, but the relay is more robust and offers a better protection to the main board.

3.4. Developing the main idea

3.4.1. Software

The goal in developing this software is that, besides working in elevators proposed above, serves as a base structure for the development of future programs. Analyzing all elevators' characteristics, the following concepts were extracted:

- There are some sensors that work as input signals of the system, which number is not previously defined.
- There are some actuators that work as output signals of the system, which number is not previously defined.
- The number of levels is not defined.
- All floors have the same amount of sensors and actuators.

Keeping this in mind, the program structure was designed.

3.4.1.1. Program structure

- Modules:

There are two modules used in the programs:

  - `time`: It provides function to manage dates and time. The only function used from this module is `time.sleep()`, which serves to delay the current execution
  - `GPIO`: This module provides functions to manage GPIOs.

- Class sensor:

This class represents a generic sensor. It stores the name in `self.name` and the port of the sensor in `self.port`. `GPIO.input()` is a function from the GPIO module that reads the value of the sensor.

```
035 class sensor:
036 # Class that represents a sensor
037     def __init__(self, name, port):
038         self.name = name
039         self.port = port
040 
041     def value(self):
042         return GPIO.input(self.port)
```

Figure 30: Class sensor example

- Class actuator:

This class represents a generic actuator, and stores the name in `self.name` and the associated port of the actuator in `self.port`. `GPIO.input()` is a function from the GPIO module that changes the value of the sensor.

```
045 class actuator:
046 # Class that represents an actuator
047     def __init__(self, name, port):
048         self.name = name
049         self.port = port
050 
051     def set_value(self, valor):
052         if(valor == 1):
053             GPIO.output(self.port, 0)
054         elif(valor == 0):
055             GPIO.output(self.port, 1)
```

Figure 31: Class actuator example

- Class floor:

This class includes the floor number in `self.number` and all the sensors and actuators related to a floor. This class changes in each program, depending on the systems sensors and actuators.

```
025 class floor:
026 # Class that represents all the sensors and actuators of a floor
027     def __init__(self, number):
028         self.number = number
029         self.FC = sensor("FC" + str(self.number), 0)
030         self.door = sensor("door" + str(self.number), 0)
031         self.button = sensor("button" + str(self.number), 0)
032         self.lock = actuator("lock" + str(self.number), 0)
```

Figure 32: Class floor example

- define_IOs():

In this function the number of floors and the ports associated to each sensor and actuator are defined.

  - Outputs:
    - P: Array that groups all floor objects
    - `motor`: Actuator type object that represents the motor
    - `valve`: Actuator type object that represents the valve

```
059 def define_IOs():
060 # Function that configures all the objects of the program and asigns
061 # the GPIO ports to each sensor and actuator
062     p0 = floor(0)
063     p1 = floor(1)
064     p2 = floor(2)
065     P = [p0, p1, p2]
066 
067     motor = actuator("motor", 37)
068     valve = actuator("valve", 38)
069 
070     P[0].FC.port = 7
071     P[0].door.port = 11
072     P[0].button.port = 13
073     P[0].lock.port = 15
074 
075     P[1].FC.port = 19
076     P[1].door.port = 21
077     P[1].button.port = 23
078     P[1].lock.port = 29
079 
080     P[2].FC.port = 31
081     P[2].door.port = 33
082     P[2].button.port = 32
083     P[2].lock.port = 35
084     return P, motor, valve
```

Figure 33: `define_IOs()` example

- GPIO_config():

This function sets the required ports as inputs or outputs. It uses `GPIO.setup()` from the GPIO module. This function requires the port that will be set up, `GPIO.OUT` to set it as an output and `GPIO.IN` to set it as an input. For inputs it is possible to set pull down or pull up resistors, with `GPIO.PUD_DOWN` and `GPIO.PUD_UP` respectively.

  - Inputs:
    - P: Array that groups all floor objects
    - motor: Object that represents the motor
    - valve: Object that represents the valve

```
087 def GPIO_config(P, motor, valve):
088 # Function that sets the GPIOs as inputs or outputs
089 # Its inputs are all the sensors and actuators of the program
090     for p in P:
091         GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
092         GPIO.setup(p.door.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
093         GPIO.setup(p.button.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
094         GPIO.setup(p.lock.port, GPIO.OUT)
095 
096 
097     GPIO.setup(motor.port, GPIO.OUT)
098     GPIO.setup(valve.port, GPIO.OUT)
099     return
```

Figure 34: `GPIO_config()` example

- initialization():

This function places the elevator in a correct and known position. Its operation varies from program to program.

  - Inputs:
    - P: Array that groups all floor objects
    - motor: Object that represents the motor
    - valve: Object that represents the valve
  - Outputs:
    - actual: variable that stores the floor where the elevator is.
    - destination: variable that stores the floor where the elevator is going.

- Interrupts

Interrupts are events that allow asynchronous function calls. This means that during the normal execution of the program, when there is an external signal such as a button press, the execution is stopped and the interrupt handling function is called. When this function has finished, it continues the main program where it was stopped.

Interrupts are used to detect button presses and auto-levelling and to react accordingly. This section changes from program to program and will be explained in further sections.

- Main function

This is the main part of the code. Here is where the operation of the elevator is managed. It changes from program to program and will be explained in further sections.

3.4.1.2. Standard elevator

The elevator must move between floors, when requested by pressing the corresponding button. There are limit switches in each floor. To go up a motor must be activated, to move down a valve must be opened.

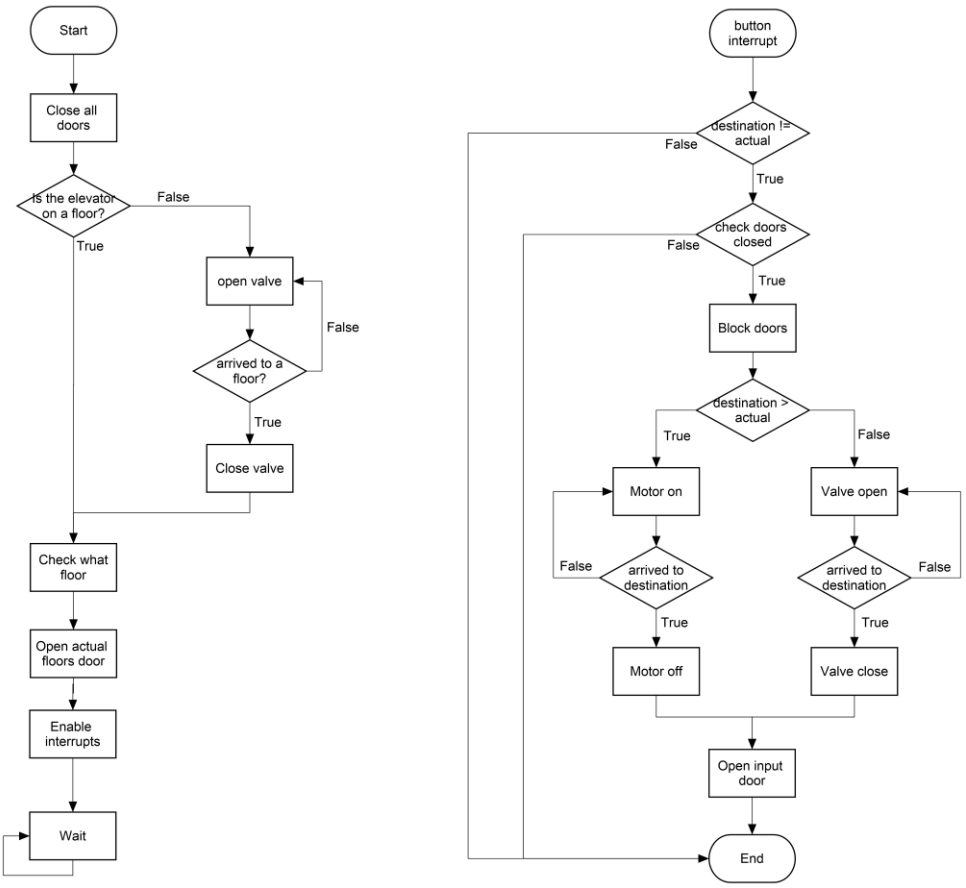


Figure 35: Standard elevators fluxogram

Variables

- **P**: array that stores all floor objects
- **p0**: floor type object that represents the first floor of the elevator
  - **p0.number**: Integer type variable that stores the floor number
  - **p0.FC**: Sensor type object that represent the limit switch of the floor
  - **p0.door**: Sensor type object that represents the limit switch of the door
  - **p0.button**: Sensor type object that represents the button of the floor
  - **p0.lock**: Actuator type object that represents the electric lock of the door
- **p1**: floor type object that represents the first floor of the elevator.
  - **p1.number**: Integer type variable that stores the floor number
  - **p1.FC**: Sensor type object that represent the limit switch of the floor

  - **p1.door**: Sensor type object that represents the limit switch of the door
  - **p1.button**: Sensor type object that represents the button of the floor
  - **p1.lock**: Actuator type object that represents the electric lock of the door
- **p2**: floor type object that represents the first floor of the elevator.
  - **p2.number**: Integer type variable that stores the floor number
  - **p2.FC**: Sensor type object that represent the limit switch of the floor
  - **p2.door**: Sensor type object that represents the limit switch of the door
  - **p2.button**: Sensor type object that represents the button of the floor
  - **p2.lock**: Actuator type object that represents the electric lock of the door
- **motor**: Actuator type object that represents the motor of the system
- **valve**: Actuator type object that represents the valve of the system
- **actual**: Integer type variable that stores the floor where the elevator is.
- **destination**: Integer type variable that stores the floor where the elevator is going.

Table 6: Standard elevators inputs/outputs

Input	Port
P0 button	13
FC floor 0	7
Door closed 0	11
P1 button	23
FC floor 1	19
Door closed 1	21
P2 button	32
FC floor 2	31
Door closed 2	33

Output	Port
Motor	37
Valve	38
Lock door 0	15
Lock door 1	29
Lock door 2	35

Code

- initialization():

This function sets the initial conditions of the system. It sets "actual" to negative value and checks if the platform is in any of the floors. If the elevator is in a floor, "actual" and “destiny” are set to the actual floor value. If the platform is not in any floor, it descends until it reaches one and sets "actual" and "destiny" to the floor number.

```
102 def initialization(P, motor, valve):
103 # Function that set the initial conditions of the system
104 # Explanation: Sets "actual" to negative value and checks if the
105 # platform is in any of the floors. In that case sets "actual" to
106 # the value of the actual floor. If the platform is not in any floor,
107 # it desdends until it reaches one and sets "actual" and "destination"
108 # to the floor number.
109     actual = -1
110     for p in P:
111         if  (p.FC.value() == 1):
112             actual = p.number
113             destination = actual
114         else:
115             while(p.door.value() != 1):
116                 pass
117             p.lock.set_value(1)
118     if (actual < 0):
119         while((P[0].door.value() and
120                P[1].door.value()) != 1):
121             pass
122         for p in P:
123             p.lock.set_value(1)
124         while((P[0].FC.value() or
125                P[1].FC.value()) == 0):
126             valve.set_value(1)
127         valve.set_value(0)
128
129         for p in P:
130             if(p.FC.value() == 1):
131                 actual = p.number
132                 destination = actual
133                 p.lock.set_value(0)
134                 break
135     return actual, destination
```

Figure 36: Initialization of standard elevator

- interrupts

In this program, interrupts are configured to be called when a button is pressed. When this happens, the value of “destination” is changed to the number of button pressed. The following code is executed when an interrupt happens:

```
165 def P1_handler(port):
166     global destination
167     destination = 0
168     return
```

Figure 37: Button interrupt handler

The function `enable_int()` is used to enable interrupts, while `disable_int()` is used to disable them. This is needed because while the elevator is moving there should not be any change in the “destiny” variable.

```
138 def enable_int(P):
139 # Enables interrupts
140     GPIO.add_event_detect(P[0].button.port,
141                           GPIO.FALLING,
142                           callback=P1_handler,
143                           bouncetime=500)
144     GPIO.add_event_detect(P[1].button.port,
145                           GPIO.FALLING,
146                           callback=P2_handler,
147                           bouncetime=500)
148     GPIO.add_event_detect(P[2].button.port,
149                           GPIO.FALLING,
150                           callback=P3_handler,
151                           bouncetime=500)
152     return
153
154
155 def disable_int(P):
156 # Disables interrupts
157     GPIO.remove_event_detect(P[0].button.port)
158     GPIO.remove_event_detect(P[1].button.port)
159     GPIO.remove_event_detect(P[2].button.port)
160     return
```

Figure 38: `enable_int()` function

- main():

First, configuration functions are executed and then it enters the loop. When a button is pressed an interrupt occurs and it sets "destination" to the number of the floor pressed. If the door is not closed "destination" is set again to the actual value. If it is, the door is locked and interrupts are disabled until the movement is over. The movement happens as follows:

1. It checks if the destination floor is up or down
2. If it is up activates the motor
3. If it is down opens the valve
4. When the destination limit switch is pressed the valve is closed and the motor is shut down.

When it arrives, destination door is unlocked, "actual" value is updated and interrupts are enabled again.

```
183 ##MAIN
184 # Explanation: First configuration functions are executed and then
185 # enters the loop. When a button is pressed a interrupt occurs and
186 # it sets "destination" to the number of the floor pressed. If the door
187 # is not closed "destination" is set again to the actual value. If it is,
188 # the door is locked and interrupts are disabled until the movement
189 # is over. The movements is as follows:
190 #   1.-It checks if the destination floor is up or down
191 #   2.-If it is up activates the motor
192 #   3.-If it is down opens the valve
193 #   4.-When the destination limit switch is pressed the valve is
194 #      closed and the motor is shut down.
195 # Destination door is unlocked, "actual" value is updated and
196 # interrupts are enabled again.
197 global destination
198 [P, motor, valve] = define_IOs()
199 GPIO_config(P, motor, valve)
200 [actual, destination] = initialization(P, motor, valve)
201 enable_int(P)
202
203 while(1):
204     try:
205         time.sleep(0.3)
206         if (actual != destination and P[actual].door.value() == 1):
207             disable_int(P)
208             P[actual].lock.set_value(1)
209             while(P[destination].FC.value() != 1):
210                 if (actual > destination):
211                     valve.set_value(1)
212                 elif(actual < destination):
213                     motor.set_value(1)
214
215             valve.set_value(0)
216             motor.set_value(0)
217             P[destination].lock.set_value(0)
218             actual = destination
219             enable_int(P)
220         else:
221             destination = actual
222     except:
223         GPIO.cleanup()
224         raise
```

Figure 39: Standard elevator's main program

3.4.1.3. Auto-levelling elevator

This elevator works as the standard elevator, but it has auto-levelling system. When it is at rest and there is a level change, due to charging or discharging the elevator for example, it must return to the floor level. Two limit switches are added to detect the level change. Also, this two limit switches allows to add two ascending and descending speeds.

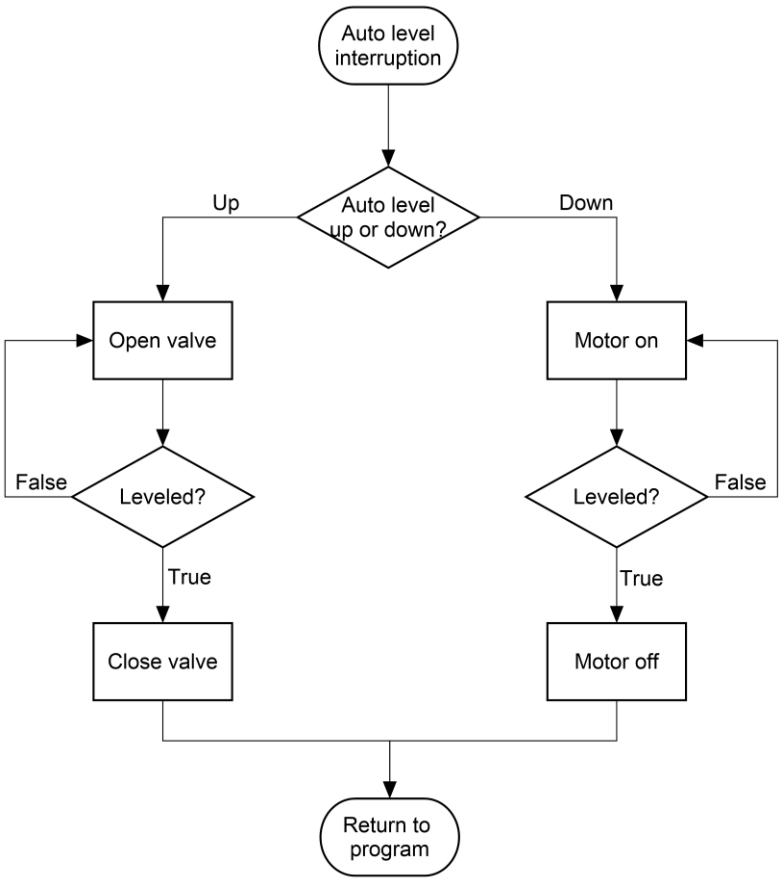


Figure 40: Auto-level interrupt fluxogram

Variables

- **P**: array that stores all floor objects
- **p0**: floor type object that represents the first floor of the elevator
- **p0.number**: Integer type variable that stores the floor number
  - **p0.FC**: Sensor type object that represent the limit switch of the floor
  - **p0.FCA**: Sensor type object that represent the upper limit switch for auto-levelling.
  - **p0.FCB**: Sensor type object that represent the lower limit switch for auto-levelling.
  - **p0.door**: Sensor type object that represents the limit switch of the door
  - **p0.button**: Sensor type object that represents the button of the floor

  - **p0.lock**: Actuator type object that represents the electric lock of the door
- **p1**: floor type object that represents the first floor of the elevator
  - **p1.number**: Integer type variable that stores the floor number
  - **p1.FC**: Sensor type object that represent the limit switch of the floor
  - **p1.FCA**: Sensor type object that represent the upper limit switch for auto-levelling.
  - **p1.FCB**: Sensor type object that represent the lower limit switch for auto-levelling.
  - **p1.door**: Sensor type object that represents the limit switch of the door
  - **p1.button**: Sensor type object that represents the button of the floor
  - **p1.lock**: Actuator type object that represents the electric lock of the door
- **p2**: floor type object that represents the first floor of the elevator
  - **p2.number**: Integer type variable that stores the floor number
  - **p2.FC**: Sensor type object that represent the limit switch of the floor
  - **p2.FCA**: Sensor type object that represent the upper limit switch for auto-leveling.
  - **p2.FCB**: Sensor type object that represent the lower limit switch for auto-levelling.
  - **p2.door**: Sensor type object that represents the limit switch of the door
  - **p2.button**: Sensor type object that represents the button of the floor
  - **p2.lock**: Actuator type object that represents the electric lock of the door
- **motor**: Actuator type object that represents the motor of the system
- **valve**: Actuator type object that represents the valve of the system
- **actual**: Integer type variable that stores the floor where the elevator is.
- **destination**: Integer type variable that stores the floor where the elevator is going.

Table 7: Auto-levelling elevator's inputs/outputs

Input	Port
P0 button	13
FC floor 0	7
FCA floor 0	8
FCB floor 0	10
Door closed 0	11
P1 button	23
FC floor 1	19
FCA floor 1	22
FCB floor 1	24
Door closed 1	21
P2 button	33
FC floor 2	29
FCA floor 2	36
FCB floor 2	38
Door closed 2	31

Output	Port
Motor	15
Valve	18
Lock door 0	12
Lock door 1	26
Lock door 2	35

Code

The main part of the code is the same as in the standard elevator. Auto-levelling management is made by an interrupt.

- Auto-levelling interrupt

For auto-levelling, one interrupt is assigned to FCA and FCB on each floor. This interrupts calls a function that makes the elevator go up or down, depending on what limit switch is activated. When FC is activated, the elevator is levelled.

```
265 def AL0_up(port):
266     level = 0
267     AL_up(level)
268     return
269
270
271 def AL0_down(port):
272     level = 0
273     AL_down(level)
274     return
```

```
150 def AL_up(piso):
151     while(P[piso].FC.value() != 1):
152         motor.salida(1)
153     motor.salida(0)
154     return
155
156
157 def AL_down(piso):
158     while(P[piso].FC.value() != 1):
159         valve.salida(1)
160     valve.salida(0)
161     return
```

Figure 41: Auto-levelling interrupt handler

To avoid possible problems during operation, this interrupt only must be enabled on the floor the elevator is. The interrupt enabling has been divided in two functions, `enable_int_general()` for button interrupts and `enable_AL()` for auto-level interrupts.

```
167 def enable_int_general(P):
168 # Enables button interrupts
169     GPIO.add_event_detect(P[0].button.port,
170                           GPIO.FALLING,
171                           callback=P1_handler,
172                           bouncetime=500)
173     GPIO.add_event_detect(P[1].button.port,
174                           GPIO.FALLING,
175                           callback=P2_handler,
176                           bouncetime=500)
177     GPIO.add_event_detect(P[2].button.port,
178                           GPIO.FALLING,
179                           callback=P3_handler,
180                           bouncetime=500)
181     return
```

Figure 42: Button interrupt enabling function

```
235 def enable_AL(actual, P):
236 # Enables autolevel interrupts depending on the floor number
237     if(actual == 0):
238         enable_int_0(P)
239     if(actual == 1):
240         enable_int_1(P)
241     if(actual == 2):
242         enable_int_2(P)
243     return
```

```
183 def enable_int_0(P):
184 # Enables Autolevel interrupts for floor 0
185     GPIO.add_event_detect(P[0].FCA.port,
186                           GPIO.RISING,
187                           callback=AL0_down,
188                           bouncetime=500)
189     GPIO.add_event_detect(P[0].FCB.port,
190                           GPIO.RISING,
191                           callback=AL0_up,
192                           bouncetime=500)
193     return
```

Figure 43: Auto-levelling interrupts enabling functions

When disabling is needed, both of them must be disabled, so all of them are included in `disable_it()`:

```
219 def disable_int(P):
220 # Disalbes interrupts
221     GPIO.remove_event_detect(P[0].button.port)
222     GPIO.remove_event_detect(P[1].button.port)
223     GPIO.remove_event_detect(P[2].button.port)
224
225     GPIO.remove_event_detect(P[0].FCA.port)
226     GPIO.remove_event_detect(P[0].FCB.port)
227
228     GPIO.remove_event_detect(P[1].FCA.port)
229     GPIO.remove_event_detect(P[1].FCB.port)
230
231     GPIO.remove_event_detect(P[2].FCA.port)
232     GPIO.remove_event_detect(P[2].FCB.port)
233     return
```

Figure 44: Auto-levelling interrupts disabling functions

Two speeds

For the configuration with two speed two outputs more are used, one for the second ascent speed and one for the second descended speed.

Table 8: Added outputs for two speeds

Output	Port
Motor 1	15
Valve 1	18
Motor 2	16
Valve 2	40

Also, two variables more are added for those new outputs, which at the same time are stored in two arrays.

- **`motors`**: array that stores all motor objects.
- **`valves`**: array that stores all valve objects.
- **`motor1`**: Actuator type object that represents the slow ascending speed.
- **`valve1`**: Actuator type object that represents the fast descending speed.
- **`motor2`**: Actuator type object that represents the slow ascending speed.
- **`valve2`**: Actuator type object that represents the fast descending speed.
- **`i`**: integer type variable used to switch between speeds.

- Changes in code

To be able to switch between speeds auto-levelling limit switches are used. When the elevator arrives to one of them, it deactivates fast speed.

```
342             while(P[destination].FC.value() != 1):
343                 if (actual > destination):
344                     while(P[destination].FCA.value() != 1 and i == 0):
345                         valves[0].set_value(1)
346                         valves[1].set_value(1)
347                     valves[1].set_value(0)
348                     i = 1
349                 elif(actual < destination):
350                     while(P[destination].FCB.value() != 1 and i == 0):
351                         motors[0].set_value(1)
352                         motors[1].set_value(1)
353                     motors[1].set_value(0)
354                     i = 1
```

Figure 45: Code changes for two speeds

3.4.1.4. Hydraulic locks

An elevator with hydraulic locks works similar to the standard elevator. The difference is that the platform rests on hydraulic locks. This changes the initialization and operation logic.

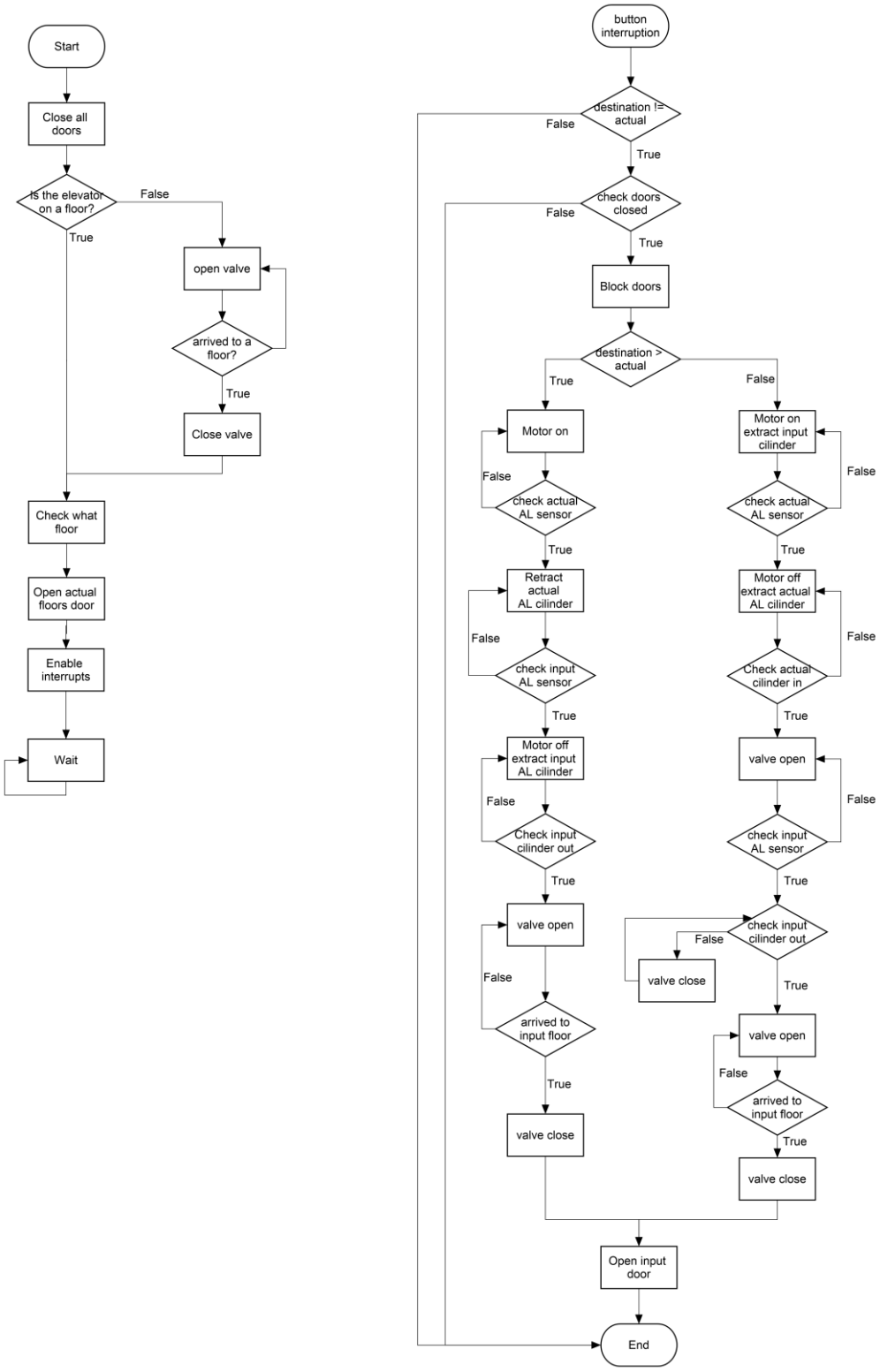


Figure 46: Hydraulic lock elevator s fluxogram

Variables

- **P**: array that stores all floor objects
- **p0**: floor type object that represents the first floor of the elevator
  - **p0.number**: Integer type variable that stores the floor number
  - **p0.FC**: Sensor type object that represent the limit switch of the floor
  - **p0.FCA**: Sensor type object that represent the upper limit switch of the floor.
  - **p0.door**: Sensor type object that represents the limit switch of the door
  - **p0.lock_FCO**: Sensor type object that represents the limit switch that detects the hydraulic lock is out.
  - **p0.lock_FCI**: Sensor type object that represents the limit switch that detects the hydraulic lock is in.
  - **p0.button**: Sensor type object that represents the button of the floor
  - **p0.lock**: Actuator type object that represents the electric lock of the door
- **p1**: floor type object that represents the first floor of the elevator
  - **p1.number**: Integer type variable that stores the floor number
  - **p1.FC**: Sensor type object that represent the limit switch of the floor
  - **p1.FCA**: Sensor type object that represent the upper limit switch for autoleveling.
  - **p1.door**: Sensor type object that represents the limit switch of the door
  - **p1.lock_FCO**: Sensor type object that represents the limit switch that detects the hydraulic lock is out.
  - **p1.lock_FCI**: Sensor type object that represents the limit switch that detects the hydraulic lock is in.
  - **p1.button**: Sensor type object that represents the button of the floor
  - **p1.lock**: Actuator type object that represents the electric lock of the door
- **p2**: floor type object that represents the first floor of the elevator
  - **p2.number**: Integer type variable that stores the floor number
  - **p2.FC**: Sensor type object that represent the limit switch of the floor
  - **p2.FCA**: Sensor type object that represent the upper limit switch.
  - **p2.door**: Sensor type object that represents the limit switch of the door
  - **p2.lock_FCO**: Sensor type object that represents the limit switch that detects the hydraulic lock is out.
  - **p2.lock_FCI**: Sensor type object that represents the limit switch that detects the hydraulic lock is in.
  - **p2.button**: Sensor type object that represents the button of the floor

- **p2.lock**: Actuator type object that represents the electric lock of the door

- **motor**: Actuator type object that represents the motor of the system
- **valve**: Actuator type object that represents the valve of the system
- **actual**: Integer type variable that stores the floor where the elevator is.
- **destination**: Integer type variable that stores the floor where the elevator is going.

Table 9: Hydraulic lock elevator's inputs/outputs

Input	Port
P0 button	13
FC floor 0	7
FCA floor 0	8
Lock FCO 0	12
Lock FCI 0	16
Door closed 0	11
P1 button	23
FC floor 1	19
FCA floor 1	18
Lock FCO 1	24
Lock FCI 1	26
Door closed 1	21
P2 button	33
FC floor 2	29
FCA floor 2	28
Lock FCO 2	36
Lock FCI 2	38
Door closed 2	31
Output	Port
motor	3
Valve	37
Lock door 0	15
Lock out 0	10
Lock door 1	21
Lock out 1	22
Lock door 2	35
Lock out 2	32

Code

- initialization():

The initialization function follows the same logic as in the other elevators, but the hydraulic locks must be extracted. If it is not on a floor it descends until FCA instead of FC, then extracts the hydraulic lock and finally descends until FC.

```
131 def initialization(P, motor, valve):
132 # Function that set the initial conditions of the system
133 # Explanation: Sets "actual" to negative value and checks if the
134 # platform is in any of the floors. In that case sets "actual" to
135 # the value of the actual floor. If the platform is not in any floor,
136 # it desdends until it reaches one and sets "actual" and "destination"
137 # to the floor number.
138     actual = -1
139     for p in P:
140         if  (p.FC.value() == 1):
141             actual = p.number
142             destination = actual
143         else:
144             while(p.puerta.value() != 1):
145                 pass
146             p.lock.set_value(1)
147     if (actual < 0):
148         while((P[0].FCA.value() or
149                P[1].FCA.value()) == 0):
150             valve.set_value(1)
151         valve.set_value(0)
152 
153         for p in P:
154             if(p.FCA.value() == 1):
155                 while(p.lock_FCO.value() != 1):
156                     p.lock_out.set_value(1)
157                 p.lock_out.set_value(0)
158                 while(p.FC.value() != 1):
159                     valve.set_value(1)
160                 valve.set_value(0)
161 
162                 actual = p.number
163                 destination = actual
164                 p.lock.set_value(0)
165     return actual, destination
```

Figure 47: Hydraulic lock elevator's initialization

- Interrupts

Interrupts work as in the standard elevator.

- main()

First, configuration functions are executed and then it enters the loop. When a button is pressed an interrupt occurs and it sets "destiny" to the number of the floor pressed. If the door is not closed "destiny" is set again to the actual value. If it is, the door is locked and interrupts are disabled until the movement is over. If the destination is down, the movement is as follows:

1. Platform elevates until FCA
2. Hydraulic lock is retracted
3. Destination hydraulic lock is extracted and discharge valve open
4. When the platform arrives to the destination the valve is closed

If the destination is up, the movement is as follows:

1. Platform elevates and Hydraulic lock is retracted
2. Platform stops when it arrives to destinations FCA
3. Destination hydraulic lock is extracted and discharge valve open
4. When the platform arrives to the destination the valve is closed

Destination door is unlocked, "actual" value is updated and interrupts are enabled again.

```
##MAIN
# Explanation: First configuration functions are executed and then
# enters the loop. When a button is pressed a interrupt occurs and
# it sets "destination" to the number of the floor pressed. If the door
# is not closed "destination" is set again to the actual value. If it is,
# the door is locked and interrupts are disabled until the movement
# is over. If the destination is down, the movement is as follows:
#   1.-Platform elevates untill FCA
#   2.-Hydraulic lock is retracted
#   3.-Destination hidraulic lock is extracted and discharge valve open
#   4.-When the platform arrives to the destination the valve is closed
# If the destination is up, the movement is as follows:
#   1.-Platform elevates and Hydraulic lock is retracted
#   2.-Platform stops when it arrives to destinations FCA
#   3.-Destination hidraulic lock is extracted and discharge valve open
#   4.-When the platform arrives to the destination the valve is closed
# Destination door is unlocked, "actual" value is
# updated and interrupts are enabled again.
global destination
[P, motor, valve] = define_IOs()
GPIO_config(P, motor, valve)
[actual, destination] = initialization(P, motor, valve)
enable_int(P)

while(1):
    try:
        if (actual != destination and P[actual].puerta.value() == 1):
            disable_int(P)
            P[actual].lock.set_value(1)
            if (actual > destination):
                while(P[actual].FCA.value() != 1):
                    motor.set_value(1)
                motor.set_value(0)
                while(P[actual].lock_FCI.value() != 1):
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                valve.set_value(1)
                while(P[destination].lock_FCO.value() != 1):
                    if(P[destination].FCA.value() == 1):
                        valve.set_value(0)
                while(P[destination].FC.value() != 1):
                    valve.set_value(1)
                valve.set_value(0)
            elif(actual < destination):
                while(P[destination].lock_FCI.value() != 1):
                    P[destination].lock_out.set_value(0)
                while(P[destination].FCA.value() != 1):
                    motor.set_value(1)
                while(P[destination].lock_FCO.value() != 1):
                    motor.set_value(0)
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                while(P[destination].FC.value() != 1):
                    valve.set_value(1)

            valve.set_value(0)
            motor.set_value(0)
            P[destination].lock.set_value(0)
            actual = destination
            enable_int(P)
    except:
        GPIO.cleanup()
        raise
```

Figure 48: Hydraulic lock elevator's main program

Two speeds

For the configuration provided with two speed, two more outputs have been used, one for the second ascent speed and one for the second descended speed.

Table 10: Added outputs for two speeds

Output	Port
motor 1	3
Valve 1	5
Motor 2	37
Valve 2	40

Also, two variables more are added for those new outputs, which at the same time are stored in two arrays.

- **`motors`**: array that stores all motor objects.
- **`valves`**: array that stores all valve objects.
- **`motor1`**: Actuator type object that represents the slow ascending speed.
- **`valve1`**: Actuator type object that represents the fast descending speed.
- **`motor2`**: Actuator type object that represents the slow ascending speed.
- **`valve2`**: Actuator type object that represents the fast descending speed.
- **`i`**: integer type variable used to switch between speeds.

Changes in code: To be able to switch between speeds limit switches are used. If the elevator is going up, when it arrives to FC slows down. When it is going down, when it arrives to FCA slows down.

```
252  if (actual > destination):
253      while(P[actual].FCA.value() != 1):
254          motors[0].set_value(1)
255      motors[0].set_value(0)
256      while(P[actual].lock_FCI.value() != 1):
257          P[actual].lock_out.set_value(0)
258          P[destination].lock_out.set_value(1)
259      valves[0].set_value(1)
260      valves[1].set_value(1)
261      while(P[destination].lock_FCO.value() != 1):
262          if(P[destination].FCA.value() == 1):
263              valves[0].set_value(0)
264              valves[1].set_value(0)
265      while(P[destination].FCA.value() != 1):
266          pass
267      valves[1].set_value(0)
268      while(P[destination].FC.value() != 1):
269          pass
270  elif(actual < destination):
271      while(P[destination].lock_FCI.value() != 1):
272          P[destination].lock_out.set_value(0)
273      while(P[destination].FC.value() != 1):
274          motors[0].set_value(1)
275          motors[1].set_value(1)
276      motors[1].set_value(0)
277      while(P[destination].FCA.value() != 1):
278          pass
279      motors[0].set_value(0)
280      while(P[destination].lock_FCO.value() != 1):
281          motors[0].set_value(0)
282          P[actual].lock_out.set_value(0)
283          P[destination].lock_out.set_value(1)
284      while(P[destination].FC.value() != 1):
285          valves[0].set_value(1)
```

Figure 49: Changes in code for two speeds

3.4.2. Hardware

After deciding that the project was going to be developed in a Raspberry Pi, the rest of the hardware must be selected. The decision was to search for already designed boards that suites the needs, rather than designing them.

3.4.2.1. Input interface

Although at first implementing an input interface was thought, the possibility of connecting directly the sensors to the Raspberry Pi was considered. The sensors work well when connected to 3.3 V from the Raspberry Pi, so this solution was adopted to save input interface cost.

In case the input interface is needed due to the kind of sensor that will be connected, the following input interface has been considered:

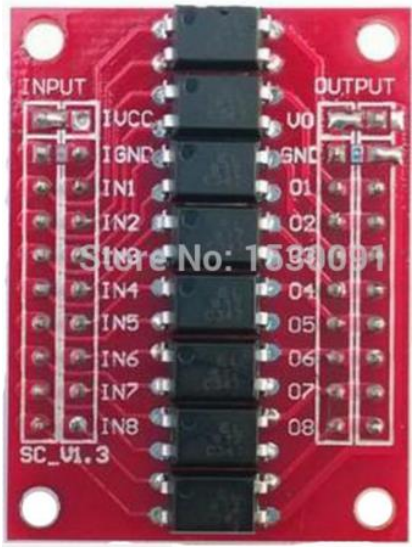


Figure 50: Proposed input interface

This board follows the proposed design before in this document. It has one optocoupler that isolates one voltage level from the other.

3.4.2.2. Output interface

As stated in the designing options selection, a relay based output interface has been selected. It is powered directly from the Raspberry Pi 3.3 V pin, and they can switch voltages until 28 VDC. This board is produced in size of 2, 4, 8 and 16 relays, giving the opportunity of adapting to each project.

Figure 51: Selected output interface

3.4.2.3. Connections

For the connections between the main board and the interfaces 40 pin ribbon wires have been selected.

Figure 52: Jumper wires

3.4.3. Testing

3.4.3.1. GPIO testing

After writing the first program this testing was made. It consists in simulating the elevator with LEDs representing actuators and buttons representing sensors. The connections were made as seen in the image.

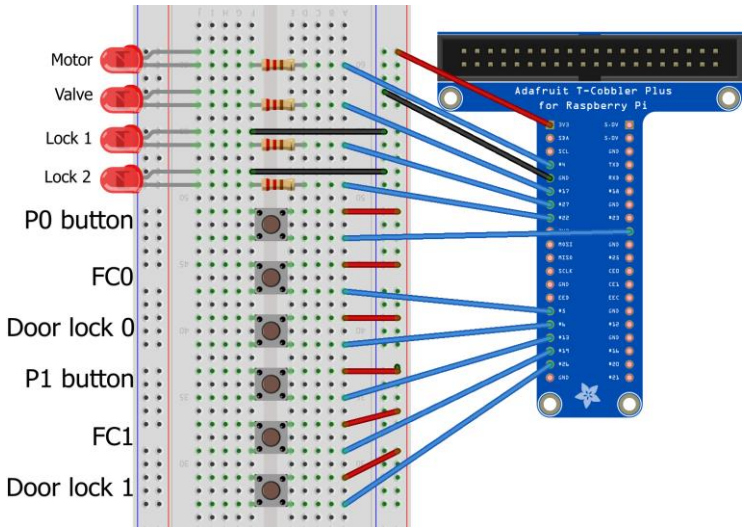


Figure 53: Test layout

The objectives of this test are:

- Check that the written logic is correct
- Check that the control of the IOs is correct

The result of this test was satisfactory.

3.4.3.2. Program testing

This test consists in checking the programmed logic. To do so, the system outputs have been configured to appear on screen, and the inputs where simulated with switches. The objective of the test was to check the operation of the programs and adjust it to work properly. This test was made for all the programs designed.

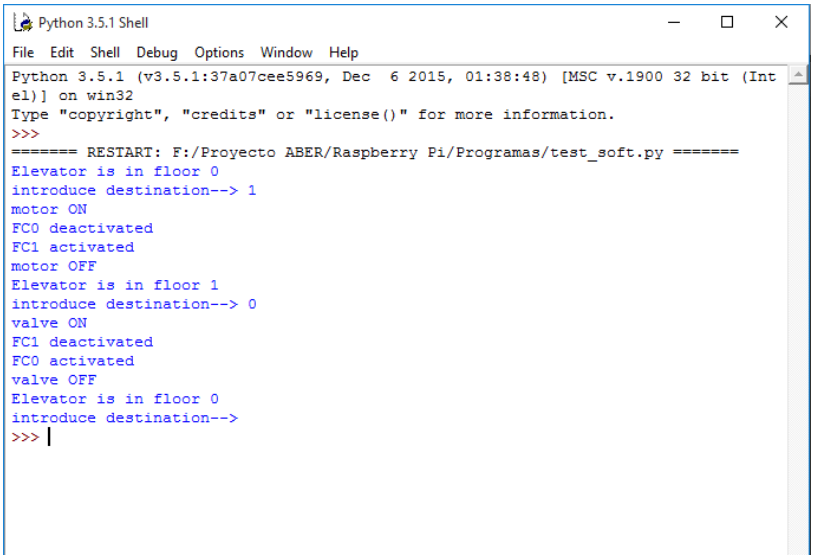


Figure 54: Testing screen

3.4.3.3. Interface testing

To check the interface a simple hydraulic system was prepared. It consisted in a biestable electrovalve that controlled a cylinder. The movement was initialized by a PC command, and it was interrupted using limit switches, one when the cylinder was ascending and another when it was descending.

The objective of this testing was to check the response of the interface to a real input and output situation, and check that the program works well with real equipment.

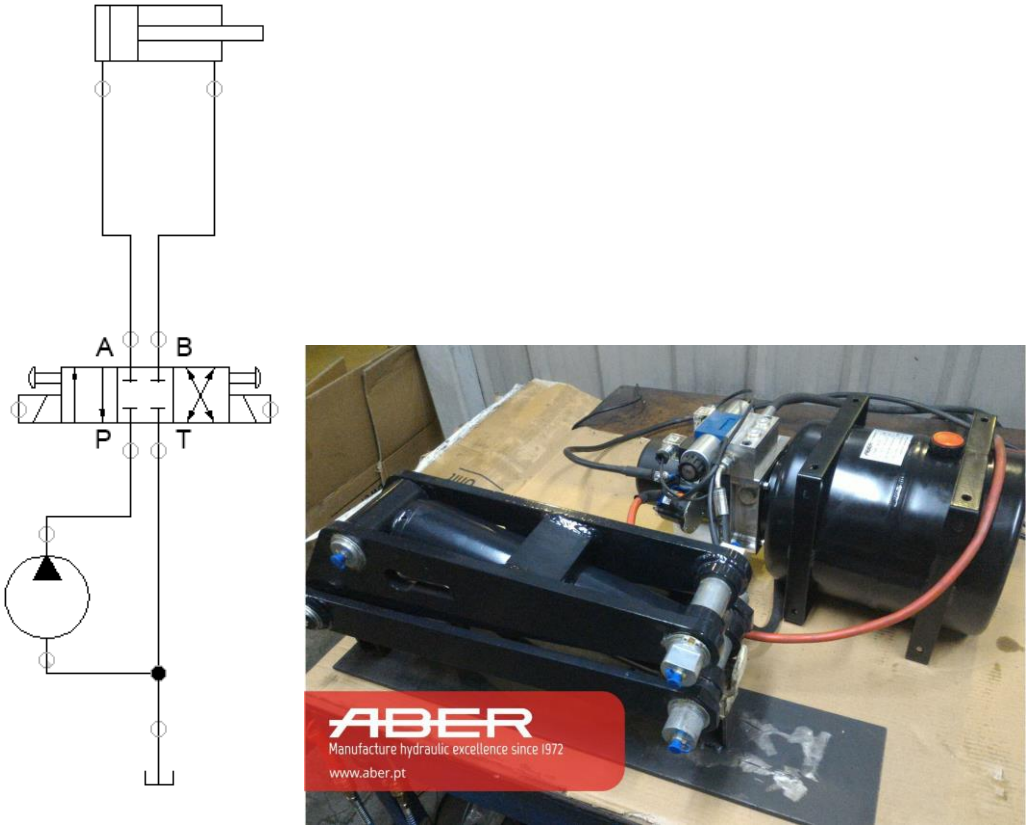


Figure 55: Hydraulic scheme and actual assembly

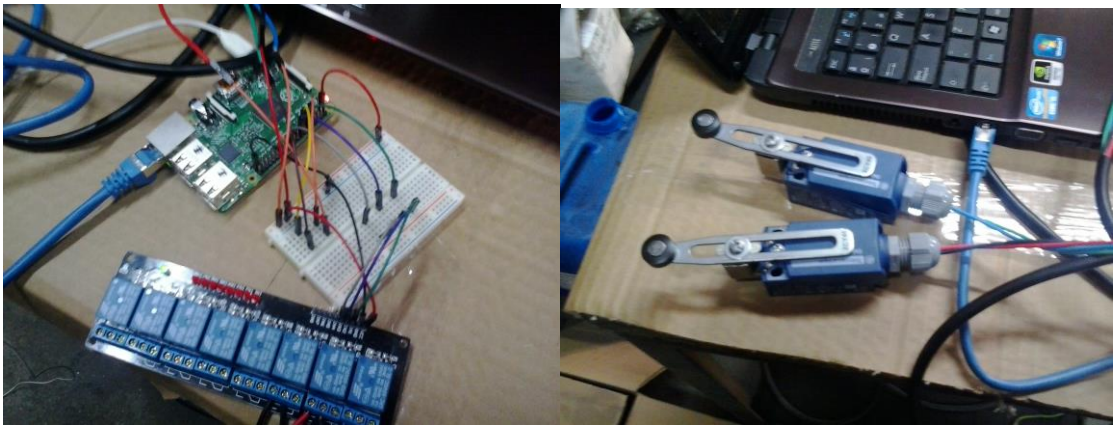


Figure 56: System connection

The program designed for this situation is based on the standard elevator program. It is simplified as it does not need so many inputs and outputs, and the virtual input is added to start the movement.

```
001 #Hardware test
002 #
003 #Inputs:     FC1
004 #            FC2
005 #            virtual input
006 #
007 #outputs:
008 #            up
009 #            down
010
011 import RPi.GPIO as GPIO     # import GPIO module
012 import time                 # import time module
013 GPIO.setmode(GPIO.BCM)      # set pin references to BCM
014
015
016 ##CLASES
017 class floor:
018 # Class that represents all the sensors and actuators of a floor
019     def __init__(self, number):
020         self.number = number
021         self.FC = sensor("FC" + str(self.number), 0)
022
023
024 class sensor:
025 # Class that represents a sensor
026     def __init__(self, name, port):
027         self.name = name
028         self.port = port
029
030     def value(self):
031         return GPIO.input(self.port)
032
033
034 class actuator:
035 # Class that represents an actuator
036     def __init__(self, name, port):
037         self.name = name
038         self.port = port
039
040     def output(self, valor):
041         GPIO.output(self.port, valor)
042
043
044 ##FUNCIONES
045 def define_IOs():
046 # Function that configures all the objects of the program and asigns
047 # the GPIO ports to each sensor and actuator
048     p0 = floor(0)
049     p1 = floor(1)
050     P = [p0, p1]
051
052     motor = actuator("motor", 18)
053     up = actuator("up", 23)
054     down = actuator("down", 25)
055     valves = [up, down]
056
057     P[0].FC.port = 17
058     P[1].FC.port = 4
059
060     return P, motor, valves
```

```
061
062
063 def GPIO_config(P, motor, valves):
064 # Function that sets the GPIOs as inputs or outputs
065 # Its inputs are all the sensors and actuators of the program
066     for p in P:
067         GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
068
069     GPIO.setup(motor.port, GPIO.OUT)
070     GPIO.setup(valves[0].port, GPIO.OUT)
071     GPIO.setup(valves[1].port, GPIO.OUT)
072
073     return
074
075
076 ##MAIN
077 #
078
079 global destino
080 [P, motor, valves] = define_IOs()
081 GPIO_config(P, motor, valves)
082 motor.output(1)
083 valves[0].output(1)
084 valves[1].output(1)
085
086 actual = 1
087 destino = 1
088
089 while(1):
090     try:
091         print("lift is on floor " + str(actual))
092         destino = int(input("introduce floor number--> "))
093         if (actual != destino):
094             while(P[destino].FC.value() != 0):
095                 if (actual < destino):
096                     motor.output(0)
097                     valves[0].output(0)
098                 elif(actual > destino):
099                     motor.output(0)
100                     valves[1].output(0)
101
102             motor.output(1)
103             valves[0].output(1)
104             valves[1].output(1)
105             actual = destino
106         else:
107             destino = actual
108     except:
109         GPIO.cleanup()
110         raise
```

3.4.4. Safety

The European standard norm for hydraulic elevators is the EN 81-2. All safety systems are operated apart from the control electronic circuit and act directly on the power supply of the whole system. With this system, if the board does not work properly the system will stop regardless what the control outputs are.

3.4.4.1. Safety devices

- Emergency button: The emergency button is a human operated switch that cuts completely the power of the system
- Safety limit switches: There are limit switched situated to detect if the elevator is out of the designated range of movement. Also, there are limit switches situated below the platform to prevent it from crushing someone.
- Limit switches are normally closed to avoid operating of the system when the sensor is broken or a wire is cut.
- Actuators are supplied independently from the control board, so it can be cut without the intervention of the board

3.5. Budgeting

The budget covers all the components related to the system designed for test. The number or type of some components may vary from one project to another, such as the output interface. Workforce hours have not been considered.

Table 11:Budget

Element	Quantity	Price	Total
Raspberry Pi 2	1	40,00 €	40,00 €
Power supply	1	8,00 €	8,00 €
Output interface	1	20,00 €	20,00 €
microSD card	1	5,00 €	5,00 €
Jumper wires	1	1,00 €	1,00 €
Enclosure	1	15,00 €	15,00 €
Conformal coating	1	10,00 €	10,00 €
TOTAL			**99,00 €**

3.6. Critical analysis and prospects for improvement

Table 12: SWOT analysis

Strengths	Weakness
• Versatility: The product is capable of adapting to a lot of products made by the company	• Robustness: The designed product is divided in different parts, which have to be assembled together.
Opportunities	**Threats**
• Innovation: Its innovation capabilities offers the opportunity to differentiate from the competition.	• Product life: As the hardware is manufactured by third parties, there is the possibility that they leave the production

As seen in the SWOT anylisis, after the development of the proyect there are still some points to take care of. As future developments, I would recomend upgrading the Raspberry Pi to the Compute module produced by the same company. This module has the same microprocessor and peripherals, but offers more inputs/outputd and more control about the microprocessor capabilities. The counterpart is that it is not as ready to use as the Raspberry Pi, so it needs extra hardware to operate. So, as second future development, I would recommend designing the hardware needed for this module, adapting its hardware to the especific needs of the company.

This developments will improve the robustness, making the device more compact, and it also would reduce the need of third party products.

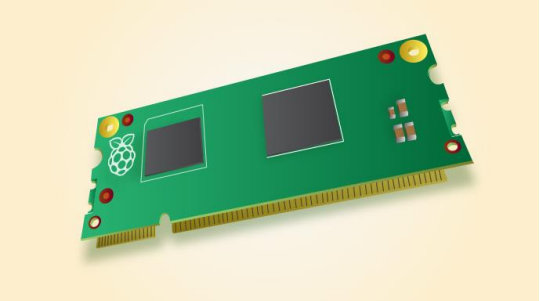


Figure 57: Compute module

3.7. Equipment instructions manual

3.7.1. Raspberry Pi raw configuration

This is a manual to configure the Raspberry Pi. All the software mentioned is free software and is just a suggestion, any other similar software can be used. This manual is done under a Windows 7 PC. To follow this manual, you will need:

- PC
- Ethernet cable
- Raspberry pi
- SD card (4 GB minimum, but 8 GB is recommended)
- SD card reader
- Internet connexion

1. Write the OS in the SD card

First, download the latest version of the OS from the official site of Raspberry.

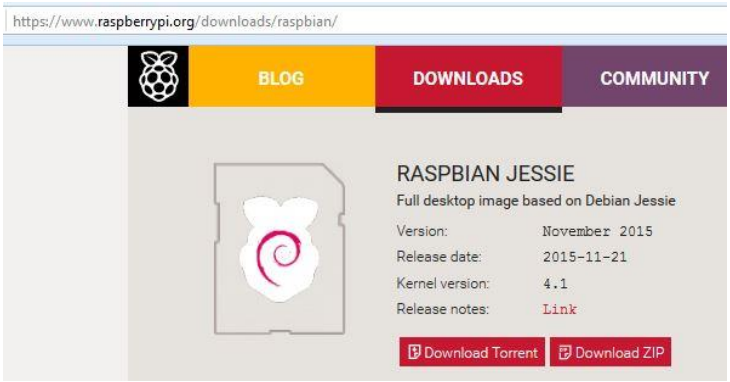


Figure 58: OS download

To be able to write the OS image in the card, a program like win32 disk imager is needed. To download it, just go to the download page shown below.

Figure 59: Win32 Disk Imager download

Now, open Win32 Disk Imager and write the image in the SD card. Choose the path of the image and be careful with the drive letter. Be sure that is referred to the SD card reader. When the configuration is done press write.

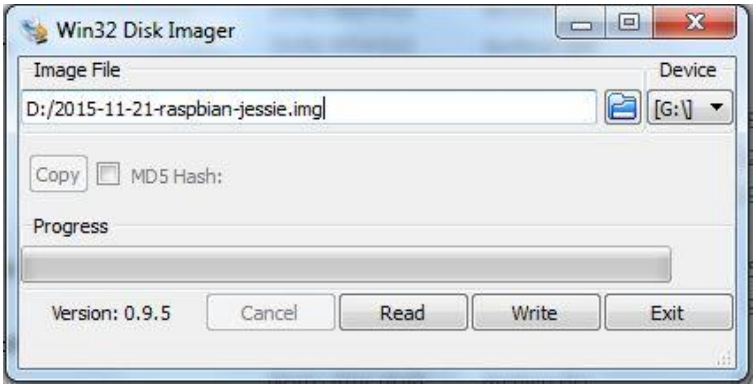


Figure 60: Win32 Disk Imager

2. Set a fixed IP for the Raspberry

A fixed IP is needed so as the remote connexion can be done. First you need to know what range of IPs is assigned to the Raspberry Pi. To do that, put the micro SD card in the RPi, connect the Ethernet cable to the PC and to the RPi and then plug the power supply. Now open the command window (cmd.exe), type 'ipconfig' and press enter.

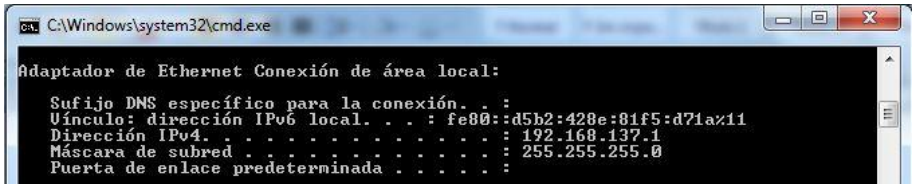


Figure 61: ipconfig

A list of wireless and local connexions should appear. Search for the Ethernet connexion and write down IP shown. In this case is 192.168.137.1.

Now unplug the power supply, extract de micro SD card and put in the PC. There should be a file named cmdline.txt. This file has only one line. Add at the end the line the IP as shown on the image. Do not add a new line, this file must have only one line.

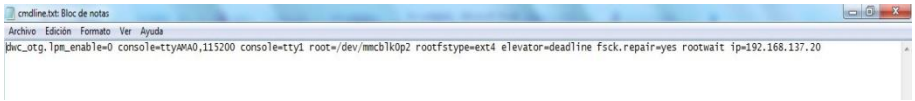


Figure 62: cmdline.txt

3. Configure RPi

Before configuring the RPi, you should check that it is allowed to connect to the Internet. Open the Network and Sharing Centre, and in the list of connexions click on the network that has Internet connexion.

Figure 63: Network and Sharing Centre

In this new window, click on properties and be sure the first checkbox is marked.

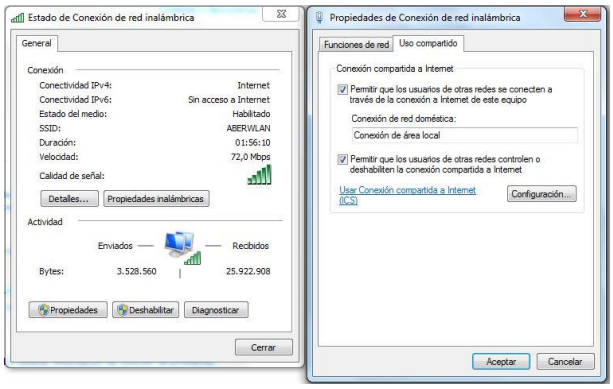


Figure 64: Network properties

After this is checked, you need a program to make the SHH connexion. Download Putty from its site, or any other similar program.

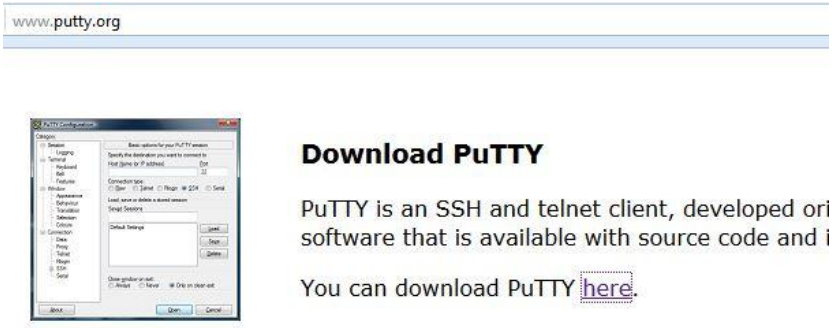


Figure 65: Putty download

Put again the SD card in the RPi and connect again to the PC and to the power supply and open Putty. Paste the fixed IP as shown in the image and press Open. A warning may appear. If so, press yes.

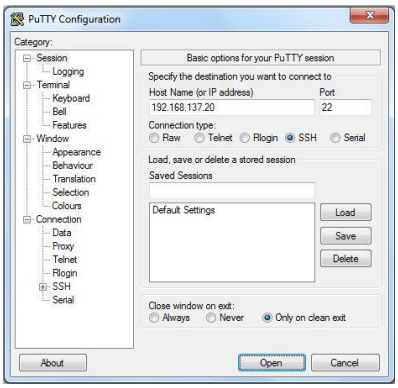


Figure 66: Putty

A black window should open. Here you must put the username and password. By default, the username is ‘pi’ and the password ‘raspberry’.

Figure 67: RPi login

Now you have access to the RPi terminal. Here enter the raspi-config command.

```
pi@raspberrypi:~ $ sudo raspi-config
```

In the configuration menu, choose the Expand Filesystem option. This will allow using all the memory of the micro SD card

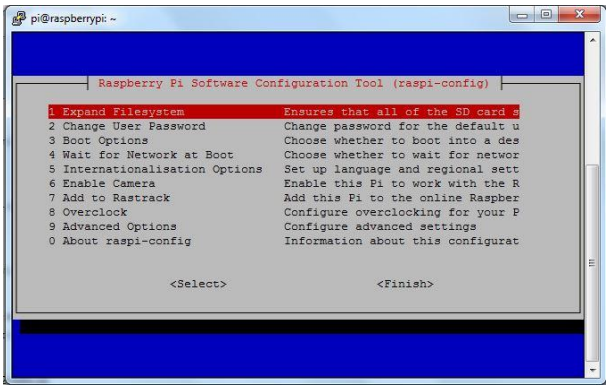


Figure 68 raspi-config

After this, execute the following commands to check for updates

```
pi@raspberrypi:~ $ sudo apt-get update
pi@raspberrypi:~ $ sudo apt-get upgrade
```

4. Remote desktop

To install the remote desktop, enter the following command. After this point, Putty is no longer needed, the connexion will be done with the remote desktop program from Windows.

```
pi@raspberrypi:~ $ sudo apt-get install xrdp
```

Open remote desktop, paste the fixed IP for the RPi and connect. Now enter the username and password and the RPi's desktop will appear.

Figure 69: Remote desktop

5. Sharing files

In the RPi's desktop, open the terminal and type the first command to install samba and then the next to configure it. After the second command is entered a text file will open.

```
pi@raspberrypi:~ $ sudo apt-get install samba samba-common-bin
pi@raspberrypi:~ $ sudo leafpad /etc/samba/smb.conf
```

In the end of the text file you have to modify the text to be like the one on the image.

```
[pihome]
   comment= Pi Home
   path=/home/pi
   browseable=Yes
   writeable=Yes
   only guest=no
   create mask=0777
   directory mask=0777
   public=no
   valid users=pi
```

Figure 70: smb.conf modifications

To finish the configuration of Samba you have to type the following command, and after it will ask for the password twice.

```
pi@raspberrypi:~ $ smbpasswd -a pi
```

3.7.2.Stablish working program

1. Write image on SD card

First step is to write “RaspberryPi_image.img” on the SD card. A program like win32 disk imager is needed. To download it, just go to the download page shown below.

Figure 71: Win32 Disk imager download

Now, open Win32 Disk Imager and write the image in the SD card. Choose the path of the image and be careful with the drive letter. Be sure that is referred to the SD card reader. When the configuration is done press write.

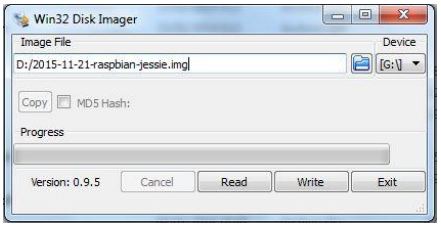


Figure 72: Image writing

2. Choose elevator program

To choose what program will be executed, ones the image is written, open “program_autorun.py” using a text editor.

```
1  #!/usr/bin/python3
2  execfile("/home/pi/Programs/autorun2.py")
3
4  ##2 floors
5  ##
6  ##/home/pi/Programs/2 floors/2Pnormal.py
7  ##/home/pi/Programs/2 floors/2P1S1DFin_curso.py
8  ##/home/pi/Programs/2 floors/2P1S1DTrancas.py
9  ##/home/pi/Programs/2 floors/2P1S2DFin_curso.py
10 ##/home/pi/Programs/2 floors/2P1S2DTrancas.py
11 ##/home/pi/Programs/2 floors/2P2S2DFin_curso.py
12 ##/home/pi/Programs/2 floors/2P2S2DTrancas.py
13 ##
14 ##3 floors
15 ##
16 ##/home/pi/Programs/3 floors/3Pnormal.py
17 ##/home/pi/Programs/3 floors/3P1S1DFin_curso.py
18 ##/home/pi/Programs/3 floors/3P1S1DTrancas.py
19 ##/home/pi/Programs/3 floors/3P1S2DFin_curso.py
20 ##/home/pi/Programs/3 floors/3P1S2DTrancas.py
21 ##/home/pi/Programs/3 floors/3P2S2DFin_curso.py
22 ##/home/pi/Programs/3 floors/3P2S2DTrancas.py
```

Figure 73: program_autorun.py

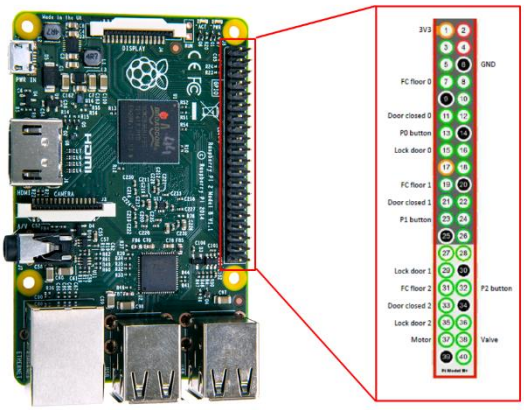
There, change the program directory to the program wanted. There is a list of available programs. It is highly recommended to maintain it actualized if any new programs are made.

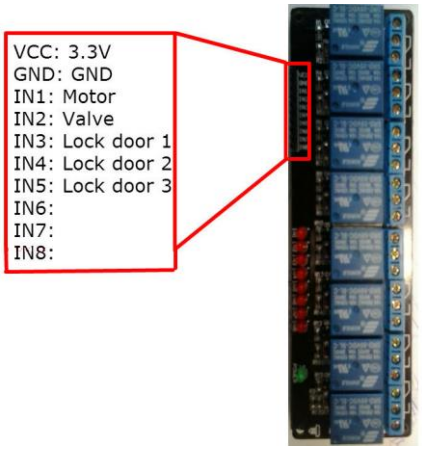
```
1  #!/usr/bin/python3
2  execfile("/enter/here/wanted/program.py")
```

Figure 74: Modifiable path

3.7.3. Pin configuration

3.7.3.1. Standard elevator





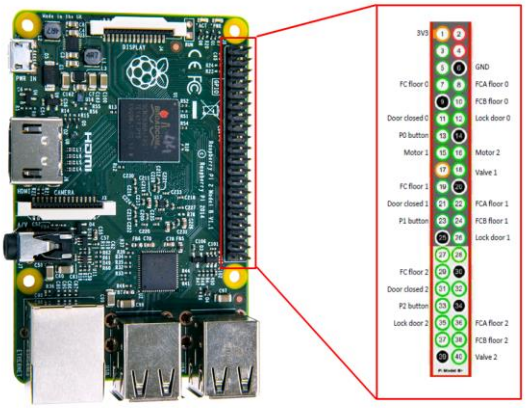
Inputs

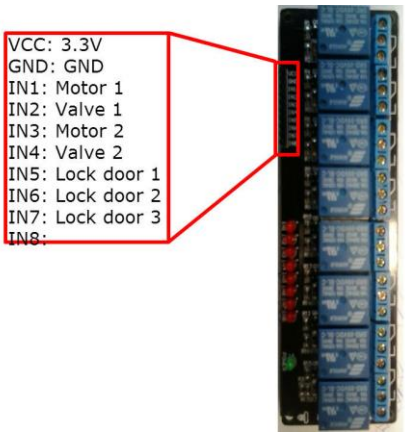
- P0 button: First floor button
- FC floor 0: First floor limit switch
- Door closed 0: First floor door limit switch
- P1 button: Second floor button
- FC floor 1: Second floor limit switch
- Door closed 1: Second floor door limit switch
- P2 button: Third floor button
- FC floor 2: Third floor limit switch
- Door closed 2: Third floor door limit switch

Outputs

- Motor: Motor contactor signal
- Valve: Valve signal
- Lock door 0: First floor electric lock
- Lock door 1: Second floor electric lock
- Lock door 2: Third floor electric lock

3.7.3.2. Auto-level elevator





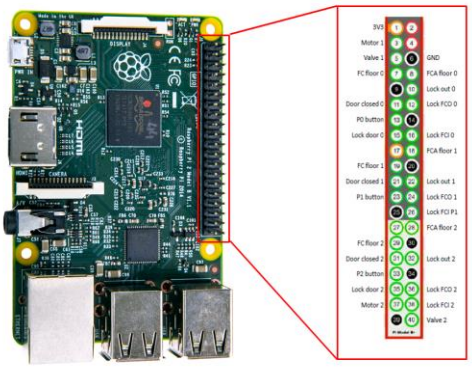
Inputs

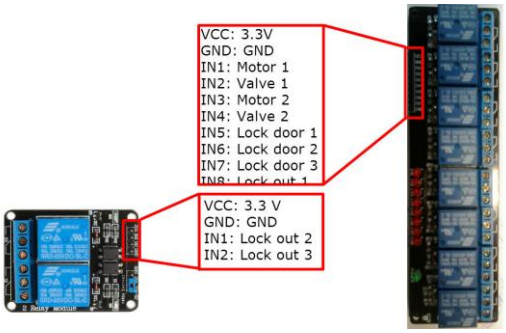
- P0 button: First floor button
- FC floor 0: First floor limit switch
- FCA floor 0: Fist floor auto level upper limit switch
- FCB floor 0: First floor auto level lower limit switch
- Door closed 0: First floor door limit switch
- P1 button: Second floor button
- FC floor 1: Second floor limit switch
- FCA floor 1:Second floor auto level upper limit switch
- FCB floor 1: Second floor auto level lower limit switch
- Door closed 1: Second floor door limit switch
- P2 button: Third floor button
- FC floor 2: Third floor limit switch
- FCA floor 2: Third floor auto level upper limit switch
- FCB floor 2: Second floor auto level lower limit switch
- Door closed 2: Third floor door limit switch

Outputs

- Motor 1: Motor contactor signal
- Motor 2: Second pump valve signal
- Valve 1: First valve signal
- Valve 2: Second valve signal
- Lock door 0: First floor electric lock
- Lock door 1: Second floor electric lock
- Lock door 2: Third floor electric lock

3.7.3.3. Hydraulic locks elevator





Inputs

- P0 button: First floor button
- FC floor 0: First floor limit switch
- FCA floor 0: Fist floor auto level upper limit switch
- Door closed 0: First floor door limit switch
- Lock FCO 0: Hydraulic lock out limit switch
- Lock FCI 0: Hydraulic lock in limit switch
- P1 button: Second floor button
- FC floor 1: Second floor limit switch
- FCA floor 1: Second floor auto level upper limit switch
- Door closed 1: Second floor door limit switch
- Lock FCO 1: Hydraulic lock out limit switch
- Lock FCI 1: Hydraulic lock in limit switch
- P2 button: Third floor button
- FC floor 2: Third floor limit switch
- FCA floor 2: Third floor auto level upper limit switch
- Door closed 2: Third floor door limit switch
- Lock FCO 2: Hydraulic lock out limit switch
- Lock FCI 2: Hydraulic lock in limit switch

Outputs

- Motor 1: Motor contactor signal
- Motor 2: Second pump valve signal
- Valve 1: First valve signal
- Valve 2: Second valve signal
- Lock door 0: First floor electric lock
- Lock door 1: Second floor electric lock
- Lock door 2: Third floor electric lock
- Lock out 1: First floor hydraulic lock set signal
- Lock out 2: Second floor hydraulic lock set signal
- Lock out 3: Third floor hydraulic lock set signal

3.8. Maintenance Guidelines

3.8.1. Assembly

To install the board in the distribution panel, first idea was to put it into an enclosure with an IP grade that protects against water. After searching, this option become harder than it looked, because of the need for external connections.

The second solution was to use a normal enclosure without waterproof protection prepared to be mounted in DIN rails, and then cover the board with conformal coating.

Figure 75: Enclosure

For the coating, a silicone one has been selected, due to its ease aplication, low cost and good waterproof performance.

Figure 76: Conformal coating

3.8.2. Maintenance

To help maintenance works, a program that register the use of system sensors and actuators was developed. This program saves in a text file the date, hour and peripheral name that was activated. This log can be exported to a spreadsheet and be used to know when the life of a peripheral is about to end, or analyze how and when is the elevator working.

The program consists in a function which is called when an interrupt occurs. Then it opens the log file and writes a new line with the data. Finally, it closes the file.

```
01 import time
02
03 def count_usage(peripheric):
04     data = time.localtime()
05     year = str(data[0])
06     mon = str(data[1])
07     day = str(data[2])
08     hour = str(data[3])
09     mins = str(data[4])
10     sec = str(data[5])
11     file = open("F:\Proyecto ABER\Raspberry Pi\Programs\egister.txt",
12                 "a+")
13
14     file.write(year + "/" + mon + "/" + day + ";" + hour + ":" +
15                mins + ":" + sec + ";" + str(peripheric.name) + "\n")
16
17     file.close()
18     return
```

Figure 77: count.py script

4. CONCLUDING REMARKS

4. Concluding remarks

After concluding the project, the results can be considered satisfactory. The board has been tested with real equipment and it had good performance, although still has to be seen how it works in the long run. Considering the objectives stated at the beginning of the work, it has been possible to standardize the control system and making it adaptable to new designs and it has potential to develop new features.

Apart from the developed product itself, the conclusions of the experience have been truly satisfactory. It was a great opportunity and a challenge to work in a foreign company in a foreign language, which offers a more realistic project that what usually is done in universities. I have improved my technical skills in electronics in which I was inexperienced, and also professional and personal skills. This is one of those opportunities that really serves for the future development of a career.

5. SOURCES OF INFORMATION

5. Bibliography and others sources of information

5.1. Books and scientific papers

DOGAN IBRAHIM. *Using LEDs, LCDs and GLCDs in Microcontroller Projects.* WILEY

BROADCOM CORPORATION (2012). *BCM2835 ARM Peripherials.*

GERT VAN LOO (2014). *ARM Quad A7 core.*

SILICON LABORATORIES INC (2013). *Developing Reliable Isolation Circuits: When to use digital isolation vs an optocoupler.*

AENOR (1999). *Normas de seguridad para la construcción de ascensores.* UNE-EN 81-2

POORVI BEHRE (2013). *Congestion-Free Elevator Control Using Microcontroller.*

MD. ANWARUL KABIR. *Design and Development of a Microcontroller-Based Cargo Lift Control System.* East West University

SANDAR HTAY, SU SU YI MON. *Implementation of PLC Based Elevator Control System.* International Journal of Electronics and Computer Science Engineering

NOMBRE APELLIDO (2013). *Titulo en cursive.* Editorial

5.2. Websites

TECMIKRO. *Solución de problemas y errores con los microcontroladores PIC.* <http://programarpicenc.com/articulos/solucion-de-problemas-y-errores-con-los-microcontroladores-pic/>

CAMBRIDGE MA (2006). *Microcontroller interfacing.* < http://cq.cx/interface.pl>

UNIFIED MICROSYSTEMS. *Microcontroller interfacing.* <http://www.w9xt.com/page_microdesign_pt1_intro.html>

KEVIN FODOR (2012). *Microcontroller Input Protection Techniques.* <http://www.kevinmfodor.com/home/My-Blog/microcontrollerinputprotectiontechniques>

RENESAS (2013). *How Photocouplers/Optocouplers are used.* <http://www.renesas.com/products/opto/technology/usage/index.jsp>

SIGNAL CONSULTING (2014). *Guard Traces.* <http://www.sigcon.com/Pubs/news/15_02.htm>

NICHOLAUS SMITH (2013). *The Engineer's Guide to high-quality PCB design.* <http://electronicdesign.com/embedded/engineer-s-guide-high-quality-pcb-design>

MAXIM INTEGRATED (2012). *Successful PCB Grounding with mixed-signal chips- follow the path of least impedance.* <https://www.maximintegrated.com/en/app-notes/index.mvp/id/5450>

DR. KELVIN T. ERICKSON (2010). *Programmable logic controllers: Hardware, software architecture.* https://www.isa.org/standards-publications/isa-publications/intech-magazine/2010/december/automation-basics-programmable-logic-controllers-hardware-software-architecture/

6. APPENDIXES

6. Programs

```
001 #2P1S1DStandard
002 #
003 #Inputs:     P0 button
004 #            P1 button
005 #            FC floor 0
006 #            FC floor 1
007 #            door closed 0
008 #            door closed 1
009 #
010 #outputs:    motor
011 #            valve
012 #            lock door 0
013 #            lock door 1
014
015 import RPi.GPIO as GPIO     # import GPIO module
016 import time                 # import time module
017 import count
018 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
019
020
021 ##CLASES
022 class floor:
023 # Class that represents all the sensors and actuators of a floor
024     def __init__(self, number):
025         self.number = number
026         self.FC = sensor("FC" + str(self.number), 0)
027         self.door = sensor("door" + str(self.number), 0)
028         self.button = sensor("button" + str(self.number), 0)
029         self.lock = actuator("lock" + str(self.number), 0)
030
031
032 class sensor:
033 # Class that represents a sensor
034     def __init__(self, name, port):
035         self.name = name
036         self.port = port
037
038     def value(self):
039         return GPIO.input(self.port)
040
041
042 class actuator:
043 # Class that represents an actuator
044     def __init__(self, name, port):
045         self.name = name
046         self.port = port
047
048     def set_value(self, valor):
049         if(valor == 1):
050             GPIO.output(self.port, 0)
051         elif(valor == 0):
052             GPIO.output(self.port, 1)
053
054
055 ##FUNCIONES
056 def define_IOs():
057 # Function that configures all the objects of the program and asigns
058 # the GPIO ports to each sensor and actuator
059     p0 = floor(0)
060     p1 = floor(1)
061     P = [p0, p1]
062
063     motor = actuator("motor", 37)
064     valve = actuator("valve", 38)
065
066     P[0].FC.port = 7
067     P[0].door.port = 11
```

```
068     P[0].button.port = 13
069     P[0].lock.port = 15
070
071     P[1].FC.port = 19
072     P[1].door.port = 21
073     P[1].button.port = 23
074     P[1].lock.port = 29
075     return P, motor, valve
076
077
078 def GPIO_config(P, motor, valve):
079 # Function that sets the GPIOs as inputs or outputs
080 # Its inputs are all the sensors and actuators of the program
081     for p in P:
082         GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
083         GPIO.setup(p.door.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
084         GPIO.setup(p.button.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
085         GPIO.setup(p.lock.port, GPIO.OUT)
086
087
088     GPIO.setup(motor.port, GPIO.OUT)
089     GPIO.setup(valve.port, GPIO.OUT)
090     return
091
092
093 def initialization(P, motor, valve):
094 # Function that set the initial conditions of the system
095 # Explanation: Sets "actual" to negative value and checks if the
096 # platform is in any of the floors. In that case sets "actual" to
097 # the value of the actual floor. If the platform is not in any floor,
098 # it desdends until it reaches one and sets "actual" and "destination"
099 # to the floor number.
100     actual = -1
101     for p in P:
102         if  (p.FC.value() == 1):
103             actual = p.number
104             destination = actual
105         else:
106             while(p.door.value() != 1):
107                 pass
108             p.lock.set_value(1)
109     if (actual < 0):
110         while((P[0].door.value() and
111                P[1].door.value()) != 1):
112             pass
113         for p in P:
114             p.lock.set_value(1)
115         while((P[0].FC.value() or
116                P[1].FC.value()) == 0):
117             valve.set_value(1)
118         valve.set_value(0)
119
120         for p in P:
121             if(p.FC.value() == 1):
122                 actual = p.number
123                 destination = actual
124                 p.lock.set_value(0)
125                 break
126     return actual, destination
127
128
129 def enable_int(P):
130 # Enables interrupts
131     GPIO.add_event_detect(P[0].button.port,
132                           GPIO.FALLING,
133                           callback=P1_handler,
134                           bouncetime=500)
135     GPIO.add_event_detect(P[1].button.port,
136                           GPIO.FALLING,
137                           callback=P2_handler,
```

```
138                             bouncetime=500)
139     return
140
141
142 def disable_int(P):
143 # Disables interrupts
144     GPIO.remove_event_detect(P[0].button.port)
145     GPIO.remove_event_detect(P[1].button.port)
146     return
147
148
149 ##INTERRUPCIONES
150 # Sets "destination" to the number of the floor the elevator must go
151 def P1_handler(port):
152     global destination
153     destination = 0
154     return
155
156
157 def P2_handler(port):
158     global destination
159     destination = 1
160     return
161
162     P[0].FC.port = 7
163     P[0].door.port = 11
164     P[0].button.port = 13
165     P[0].lock.port = 15
166
167
168 def count_handler(port):
169     for p in P:
170         if p.FC.port == port:
171             peri = p.FC
172             break
173         elif p.door.port == port:
174             peri = p.door
175             break
176         elif p.button.port == port:
177             peri = p.door
178             break
179         elif p.lock.port == port:
180             peri = p.door
181             break
182     if motor.port == port:
183         peri = motor
184     elif valve.port == port:
185         peri = valve
186
187     count_usage(peri)
188     return
189
190 ##MAIN
191 # Explanation: First configuration functions are executed and then
192 # enters the loop. When a button is pressed a interrupt occurs and
193 # it sets "destination" to the number of the floor pressed. If the door
194 # is not closed "destination" is set again to the actual value. If it is,
195 # the door is locked and interrupts are disabled until the movement
196 # is over. The movements is as follows:
197 #   1.-It checks if the destination floor is up or down
198 #   2.-If it is up activates the motor
199 #   3.-If it is down opens the valve
200 #   4.-When the destination limit switch is pressed the valve is
201 #      closed and the motor is shut down.
202 # Destination door is unlocked, "actual" value is updated and
203 # interrupts are enabled again.
204 global destination
205 [P, motor, valve] = define_IOs()
206 GPIO_config(P, motor, valve)
207 [actual, destination] = initialization(P, motor, valve)
```

```
208 enable_int(P)
209
210 while(1):
211     try:
212         time.sleep(0.3)
213         if (actual != destination and P[actual].door.value() == 1):
214             disable_int(P)
215             P[actual].lock.set_value(1)
216             while(P[destination].FC.value() != 1):
217                 if (actual > destination):
218                     valve.set_value(1)
219                 elif(actual < destination):
220                     motor.set_value(1)
221
222             valve.set_value(0)
223             motor.set_value(0)
224             P[destination].lock.set_value(0)
225             actual = destination
226             enable_int(P)
227         else:
228             destination = actual
229     except:
230         GPIO.cleanup()
231         raise
```

```
#2P1S1DAuto-level
#
#Inputs:    P0 button
#           P1 button
#           FC floor 0
#           FCA floor 0
#           FCB floor 0
#           FC floor 1
#           FCA floor 1
#           FCB floor 1
#           door closed 0
#           door closed 1
#
#outputs:   motor
#           valve
#           lock door 0
#           lock door 1

import RPi.GPIO as GPIO # import GPIO module
import time             # import time module
GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD


##CLASES
class floor:
# Class that represents all the sensors and actuators of a floor
    def __init__(self, number):
        self.number = number
        self.FC = sensor("FC" + str(self.number), 0)
        self.FCA = sensor("FCA" + str(self.number), 0)
        self.FCB = sensor("FCB" + str(self.number), 0)
        self.door = sensor("door" + str(self.number), 0)
        self.button = sensor("button" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)


class sensor:
# Class that represents a sensor
    def __init__(self, name, port):
        self.name = name
        self.port = port

    def set_value(self, valor):
        if(valor == 1):
            GPIO.output(self.port, 0)
        elif(valor == 0):
            GPIO.output(self.port, 1)


class actuator:
# Class that represents an actuator
    def __init__(self, name, port):
        self.name = name
        self.port = port

    def set_value(self, valor):
        GPIO.output(self.port, valor)


##FUNCIONES
def define_IOs():
# Function that configures all the objects of the program and asigns
# the GPIO ports to each sensor and actuator
    p0 = floor(0)
    p1 = floor(1)
    P = [p0, p1]

    motor = actuator("motor", 0)
    valve = actuator("valve", 0)
```

```
070
071     P[0].FC.port = 4
072     P[0].door.port = 17
073     P[0].button.port = 27
074     P[0].lock.port = 0
075     P[0].FCA.port = 14
076     P[0].FCB.port = 15
077
078     P[1].FC.port = 10
079     P[1].door.port = 9
080     P[1].button.port = 11
081     P[1].lock.port = 0
082     P[1].FCA.port = 25
083     P[1].FCB.port = 8
084     return P, motor, valve
085
086
087 def GPIO_config(P, motor, valve):
088 # Function that sets the GPIOs as inputs or outputs
089 # Its inputs are all the sensors and actuators of the program
090     for p in P:
091         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
092         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
093         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
094         GPIO.setup(p.lock.port,GPIO.OUT)
095         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
096         GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
097
098     GPIO.setup(motor.port,GPIO.OUT)
099     GPIO.setup(valve.port,GPIO.OUT)
100     return
101
102
103 def initialization(P, motor, valve):
104 # Function that set the initial conditions of the system
105 # Explanation: Sets "actual" to negative value and checks if the
106 # platform is in any of the floors. In that case sets "actual" to
107 # the value of the actual floor. If the platform is not in any floor,
108 # it desdends until it reaches one and sets "actual" and "destination"
109 # to the floor number.
110     actual = -1
111     for p in P:
112         if  (p.FC.value() == 1):
113             actual = p.number
114             destination = actual
115         else:
116             while(p.door.value() != 1):
117                 pass
118             p.lock.set_value(1)
119     if (actual < 0):
120         while((P[0].door.value() and
121                P[1].door.value()) != 1):
122             pass
123         for p in P:
124             p.lock.set_value(1)
125         while((P[0].FC.value() or
126                P[1].FC.value()) == 0):
127             valve.set_value(1)
128         valve.set_value(0)
129
130         for p in P:
131             if(p.FC.value() == 1):
132                 actual = p.number
133                 destination = actual
134                 p.lock.set_value(0)
135                 break
136     return actual, destination
137
138
139 def AL_up(level):
```

```
140     while(P[level].FC.value() != 1):
141         motor.set_value(1)
142     motor.set_value(0)
143     return
144
145
146 def AL_down(level):
147     while(P[level].FC.value() != 1):
148         valve.set_value(1)
149     valve.set_value(0)
150     return
151
152
153 def enable_int_general(P):
154 # Enables button interrupts
155     GPIO.add_event_detect(P[0].button.port,
156                           GPIO.FALLING,
157                           callback=P1_handler,
158                           bouncetime=500)
159     GPIO.add_event_detect(P[1].button.port,
160                           GPIO.FALLING,
161                           callback=P2_handler,
162                           bouncetime=500)
163     return
164
165 def enable_int_0(P):
166 # Enables Autolevel interrupts for floor 0
167     GPIO.add_event_detect(P[0].FCA.port,
168                           GPIO.RISING,
169                           callback=AL0_down,
170                           bouncetime=500)
171     GPIO.add_event_detect(P[0].FCB.port,
172                           GPIO.RISING,
173                           callback=AL0_up,
174                           bouncetime=500)
175     return
176
177 def enable_int_1(P):
178 # Enables Autolevel interrupts for floor 1
179     GPIO.add_event_detect(P[1].FCA.port,
180                           GPIO.RISING,
181                           callback=AL1_down,
182                           bouncetime=500)
183     GPIO.add_event_detect(P[1].FCB.port,
184                           GPIO.RISING,
185                           callback=AL1_up,
186                           bouncetime=500)
187     return
188
189
190 def disable_int(P):
191 # Disalbes interrupts
192     GPIO.remove_event_detect(P[0].button.port)
193     GPIO.remove_event_detect(P[1].button.port)
194
195     GPIO.remove_event_detect(P[0].FCA.port)
196     GPIO.remove_event_detect(P[0].FCB.port)
197
198     GPIO.remove_event_detect(P[1].FCA.port)
199     GPIO.remove_event_detect(P[1].FCB.port)
200     return
201
202 def enable_AL(actual, P):
203 # Enables autolevel interrupts depending on the floor number
204     if(actual == 0):
205         enable_int_0(P)
206     if(actual == 1):
207         enable_int_1(P)
208     if(actual == 2):
209         enable_int_2(P)
```

```
210     return
211
212 ##INTERRUPCIONES
213 # Sets "destination" to the number of the floor the elevator must go
214 def P1_handler(port):
215     global destination
216     destination = 0
217     return
218
219
220 def P2_handler(port):
221     global destination
222     destination = 1
223     return
224
225
226 def AL0_up(port):
227     level = 0
228     AL_up(level)
229     return
230
231
232 def AL0_down(port):
233     level = 0
234     AL_down(level)
235     return
236
237
238 def AL1_up(port):
239     level = 1
240     AL_up(level)
241     return
242
243
244 def AL1_down(port):
245     level = 1
246     AL_down(level)
247     return
248
249
250 ##MAIN
251 # Explanation: First configuration functions are executed and then
252 # enters the loop. When a button is pressed a interrupt occurs and
253 # it sets "destination" to the number of the floor pressed. If the door
254 # is not closed "destination" is set again to the actual value. If it is,
255 # the door is locked and interrupts are disabled until the movement
256 # is over. The movements is as follows:
257 #   1.-It checks if the destination floor is up or down
258 #   2.-If it is up activates the motor
259 #   3.-If it is down opens the valve
260 #   4.-When the destination limit switch is pressed the valve is
261 #      closed and the motor is shut down.
262 # Destination door is unlocked, "actual" value is updated and
263 # interrupts are enabled again.
264 # Autolevel: When the elevator is at rest, if one of the actual floors
265 # autolevel limit switch is activated an interupt occurs. Depending
266 # on if it is the upper or lower switch the valve or the motor
267 # is activated
268 global destination
269 [P, motor, valve] = define_IOs()
270 GPIO_config(P, motor, valve)
271 [actual, destination] = initialization(P, motor, valve)
272 enable_int_general(P)
273 enable_AL(actual, P)
274
275 while(1):
276     try:
277         time.sleep(0.3)
278         if (actual != destination and P[actual].door.value() == 1):
279             disable_int(P)
```

```
280             P[actual].lock.set_value(1)
281             while(P[destination].FC.value() != 1):
282                 if (actual > destination):
283                     valve.set_value(1)
284                 elif(actual < destination):
285                     motor.set_value(1)
286 
287             valve.set_value(0)
288             motor.set_value(0)
289             P[destination].lock.set_value(0)
290             actual = destination
291             enable_int_general(P)
292             enable_AL(actual, P)
293         else:
294             destination = actual
295     except:
296         GPIO.cleanup()
297         raise
```

```
001 #2P1S2DAuto-level
002 #
003 #Inputs:     P0 button
004 #            P1 button
005 #            FC floor 0
006 #            FCA floor 0
007 #            FCB floor 0
008 #            FC floor 1
009 #            FCA floor 1
010 #            FCB floor 1
011 #            door closed 0
012 #            door closed 1
013 #
014 #outputs:    motor
015 #            valve 1
016 #            valve 2
017 #            lock door 0
018 #            lock door 1
019
020 import RPi.GPIO as GPIO # import GPIO module
021 import time            # import time module
022 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
023
024 ##CLASES
025 class floor:
026 # Class that represents all the sensors and actuators of a floor
027     def __init__(self, number):
028         self.number = number
029         self.FC = sensor("FC" + str(self.number), 0)
030         self.FCA = sensor("FCA" + str(self.number), 0)
031         self.FCB = sensor("FCB" + str(self.number), 0)
032         self.door = sensor("door" + str(self.number), 0)
033         self.button = sensor("button" + str(self.number), 0)
034         self.lock = actuator("lock" + str(self.number), 0)
035
036 class sensor:
037 # Class that represents a sensor
038     def __init__(self,name,port):
039         self.name = name
040         self.port = port
041
042     def value(self):
043         return GPIO.input(self.port)
044
045 class actuator:
046 # Class that represents an actuator
047     def __init__(self, name, port):
048         self.name = name
049         self.port = port
050
051     def set_value(self, valor):
052         if(valor == 1):
053             GPIO.output(self.port, 0)
054         elif(valor == 0):
055             GPIO.output(self.port, 1)
056
057
058 ##FUNCIONES
059 def define_IOs():
060 # Function that configures all the objects of the program and asigns
061 # the GPIO ports to each sensor and actuator
062     p0 = floor(0)
063     p1 = floor(1)
064     P = [p0, p1]
065
066     motor = actuator("motor", 0)
067     valve1 = actuator("valve1", 0)
068     valve2 = actuator("valve2", 0)
069     valves = [valve1, valve2]
```

```
070
071     P[0].FC.port = 4
072     P[0].door.port = 17
073     P[0].button.port = 27
074     P[0].lock.port = 0
075     P[0].FCA.port = 14
076     P[0].FCB.port = 15
077
078     P[1].FC.port = 10
079     P[1].door.port = 9
080     P[1].button.port = 11
081     P[1].lock.port = 0
082     P[1].FCA.port = 25
083     P[1].FCB.port = 8
084     return P, motor, valves
085
086 def GPIO_config(P, motor, valves):
087 # Function that sets the GPIOs as inputs or outputs
088 # Its inputs are all the sensors and actuators of the program
089     for p in P:
090         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
091         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
092         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
093         GPIO.setup(p.lock.port,GPIO.OUT)
094         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
095         GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
096
097     GPIO.setup(motor.port,GPIO.OUT)
098     GPIO.setup(valve1.port,GPIO.OUT)
099     GPIO.setup(valve2.port,GPIO.OUT)
100     return
101
102 def initialization(P, motor, valves):
103 # Function that set the initial conditions of the system
104 # Explanation: Sets "actual" to negative value and checks if the
105 # platform is in any of the floors. In that case sets "actual" to
106 # the value of the actual floor. If the platform is not in any floor,
107 # it desdends until it reaches one and sets "actual" and "destination"
108 # to the floor number.
109     actual = -1
110     for p in P:
111         if  (p.FC.value() == 1):
112             actual = p.number
113             destination = actual
114         else:
115             while(p.door.value() != 1):
116                 pass
117             p.lock.set_value(1)
118     if (actual < 0):
119         while((P[0].door.value() and
120                P[1].door.value()) != 1):
121             pass
122         for p in P:
123             p.lock.set_value(1)
124         while((P[0].FC.value() or
125                P[1].FC.value()) == 0):
126             valves[0].set_value(1)
127         valves[0].set_value(0)
128
129         for p in P:
130             if(p.FC.value() == 1):
131                 actual = p.number
132                 destination = actual
133                 p.lock.set_value(0)
134                 break
135     return actual, destination
136
137 def AL_up(level):
138     while(P[level].FC.value() != 1):
139         motor.set_value(1)
```

```
140     motor.set_value(0)
141     return
142 
143 def AL_down(level):
144     while(P[level].FC.value() != 1):
145         valves[0].set_value(1)
146         valves[1].set_value(0)
147     valves[0].set_value(0)
148     return
149 
150 def enable_int_general(P):
151 # Enables button interrupts
152     GPIO.add_event_detect(P[0].button.port,
153                           GPIO.FALLING,
154                           callback=P1_handler,
155                           bouncetime=500)
156     GPIO.add_event_detect(P[1].button.port,
157                           GPIO.FALLING,
158                           callback=P2_handler,
159                           bouncetime=500)
160     return
161 
162 def enable_int_0(P):
163 # Enables Autolevel interrupts for floor 0
164     GPIO.add_event_detect(P[0].FCA.port,
165                           GPIO.RISING,
166                           callback=AL0_down,
167                           bouncetime=500)
168     GPIO.add_event_detect(P[0].FCB.port,
169                           GPIO.RISING,
170                           callback=AL0_up,
171                           bouncetime=500)
172     return
173 
174 def enable_int_1(P):
175 # Enables Autolevel interrupts for floor 1
176     GPIO.add_event_detect(P[1].FCA.port,
177                           GPIO.RISING,
178                           callback=AL1_down,
179                           bouncetime=500)
180     GPIO.add_event_detect(P[1].FCB.port,
181                           GPIO.RISING,
182                           callback=AL1_up,
183                           bouncetime=500)
184     return
185 
186 def disable_int(P):
187 # Disalbes interrupts
188     GPIO.remove_event_detect(P[0].button.port)
189     GPIO.remove_event_detect(P[1].button.port)
190 
191     GPIO.remove_event_detect(P[0].FCA.port)
192     GPIO.remove_event_detect(P[0].FCB.port)
193 
194     GPIO.remove_event_detect(P[1].FCA.port)
195     GPIO.remove_event_detect(P[1].FCB.port)
196     return
197 
198 def enable_AL(actual, P):
199 # Enables autolevel interrupts depending on the floor number
200     if(actual == 0):
201         enable_int_0(P)
202     if(actual == 1):
203         enable_int_1(P)
204     return
205 
206 ##INTERRUPCIONES
207 # Sets "destination" to the number of the floor the elevator must go
208 def P1_handler(self):
209     global destination
```

```
210     destination = 0
211     return
212 
213 def P2_handler(self):
214     global destination
215     destination = 1
216     return
217 
218 def AL0_up(self):
219     level = 0
220     AL_up(level)
221     return
222 
223 def AL0_down(self):
224     level = 0
225     AL_down(level)
226     return
227 
228 def AL1_up(self):
229     level = 1
230     AL_up(level)
231     return
232 
233 def AL1_down(self):
234     level = 1
235     AL_down(level)
236     return
237 
238 
239 ##MAIN
240 # Explanation: First configuration functions are executed and then
241 # enters the loop. When a button is pressed a interrupt occurs and
242 # it sets "destination" to the number of the floor pressed. If the door
243 # is not closed "destination" is set again to the actual value. If it is,
244 # the door is locked and interrupts are disabled until the movement
245 # is over. The movements is as follows:
246 #   1.-It checks if the destination floor is up or down
247 #   2.-If it is up activates the motor
248 #   3.-If it is down activates vale1 and valve2 until FCA,
249 #      then close valve2
250 #   4.-When the destination limit switch is pressed the valve is
251 #      closed and the motor is shut down.
252 # Destination door is unlocked, "actual" value is updated and
253 # interrupts are enabled again.
254 # Autolevel: When the elevator is at rest, if one of the actual floors
255 # autolevel limit switch is activated an interupt occurs. Depending
256 # on if it is the upper or lower switch the valve or the motor
257 # is activated
258 global destination
259 [P, motor, valves] = define_IOs()
260 GPIO_config(P, motor, valves)
261 [actual, destination] = initialization(P, motor, valves)
262 enable_int_general(P)
263 enable_AL(actual, P)
264 i = 0
265 
266 while(1):
267     try:
268         sleep(0.3)
269         if (actual != destination and P[actual].door.value() == 1):
270             disable_int(P)
271             P[actual].lock.set_value(1)
272             while(P[destination].FC.value() != 1):
273                 if (actual > destination):
274                     while(P[destination].FCA.value() != 1 and i == 0)
275                         valves[0].set_value(1)
276                         valves[1].set_value(1)
277                     valves[1].set_value(0)
278                     i = 1
279                 elif(actual < destination):
```

```
280                     motor.set_value(1)
281
282             valve[0].set_value(0)
283             motor.set_value(0)
284             P[destination].lock.set_value(0)
285             actual = destination
286             i = 0
287             enable_int_general(P)
288             enable_AL(actual, P)
289         else:
290             destination = actual
291         except:
292             GPIO.cleanup()
293             raise
```

```
#2P2S2DAuto-level
#
#Inputs:     P0 button
#            P1 button
#            FC floor 0
#            FCA floor 0
#            FCB floor 0
#            FC floor 1
#            FCA floor 1
#            FCB floor 1
#            door closed 0
#            door closed 1
#
#outputs:    motor 1
#            motor 2
#            valve 1
#            valve 2
#            lock door 0
#            lock door 1

import RPi.GPIO as GPIO # import GPIO module
import time             # import time module
GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD


##CLASES
class floor:
# Class that represents all the sensors and actuators of a floor
    def __init__(self, number):
        self.number = number
        self.FC = sensor("FC" + str(self.number), 0)
        self.FCA = sensor("FCA" + str(self.number), 0)
        self.FCB = sensor("FCB" + str(self.number), 0)
        self.door = sensor("door" + str(self.number), 0)
        self.button = sensor("button" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)


class sensor:
# Class that represents a sensor
    def __init__(self,name,port):
        self.name = name
        self.port = port

    def value(self):
        return GPIO.input(self.port)


class actuator:
# Class that represents an actuator
    def __init__(self, name, port):
        self.name = name
        self.port = port

    def set_value(self, valor):
        if(valor == 1):
            GPIO.output(self.port, 0)
        elif(valor == 0):
            GPIO.output(self.port, 1)


##FUNCIONES
def define_IOs():
# Function that configures all the objects of the program and asigns
# the GPIO ports to each sensor and actuator
    p0 = floor(0)
    p1 = floor(1)
    P = [p0, p1]

```

```
070     motor1 = actuator("motor1", 15)
071     motor2 = actuator("motor2", 16)
072     valve1 = actuator("valve1", 18)
073     valve2 = actuator("valve2", 40)
074     valves = [valve1, valve2]
075     motors = [motor1, motor2]
076
077     P[0].FC.port = 7            # amarillo
078     P[0].door.port = 11        # azul
079     P[0].button.port = 13        # morado
080     P[0].lock.port = 15         # -----
081     P[0].FCA.port = 8           # naranja
082     P[0].FCB.port = 10           # verde
083
084     P[1].FC.port = 19           # amarillo
085     P[1].door.port = 21         # azul
086     P[1].button.port = 23         # morado
087     P[1].lock.port = 26         # -----
088     P[1].FCA.port = 22           # naranja
089     P[1].FCB.port = 24           # verde
090
091     return P, motors, valves
092
093
094 def GPIO_config(P, motors, valves):
095 # Function that sets the GPIOs as inputs or outputs
096 # Its inputs are all the sensors and actuators of the program
097     for p in P:
098         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
099         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
100         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
101         GPIO.setup(p.lock.port,GPIO.OUT)
102         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
103         GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
104
105     GPIO.setup(motors[0].port,GPIO.OUT)
106     GPIO.setup(motors[1].port,GPIO.OUT)
107     GPIO.setup(valves[0].port,GPIO.OUT)
108     GPIO.setup(valves[1].port,GPIO.OUT)
109     return
110
111
112 def initialization(P, motors, valves):
113 # Function that set the initial conditions of the system
114 # Explanation: Sets "actual" to negative value and checks if the
115 # platform is in any of the floors. In that case sets "actual" to
116 # the value of the actual floor. If the platform is not in any floor,
117 # it desdends until it reaches one and sets "actual" and "destination"
118 # to the floor number.
119     actual = -1
120     for p in P:
121         if  (p.FC.value() == 1):
122             actual = p.number
123             destination = actual
124         else:
125             while(p.door.value() != 1):
126                 pass
127             p.lock.set_value(1)
128     if (actual < 0):
129         while((P[0].door.value() and
130                P[1].door.value()) != 1):
131             pass
132         for p in P:
133             p.lock.set_value(1)
134         while((P[0].FC.value() or
135                P[1].FC.value()) == 0):
136             valves[0].set_value(1)
137         valves[0].set_value(0)
138
139         for p in P:
```

```
140             if(p.FC.value() == 1):
141                 actual = p.number
142                 destination = actual
143                 break
144     return actual, destination
145
146
147 def AL_up(level):
148     while(P[level].FC.value() != 1):
149         motors[0].set_value(1)
150         motors[1].set_value(0)
151     motors[0].set_value(0)
152     return
153
154
155 def AL_down(level):
156     while(P[level].FC.value() != 1):
157         valves[0].set_value(1)
158         valves[1].set_value(0)
159     valves[0].set_value(0)
160     return
161
162
163 def enable_int_general(P):
164 # Enables button interrupts
165     GPIO.add_event_detect(P[0].button.port,
166                           GPIO.FALLING,
167                           callback=P1_handler,
168                           bouncetime=500)
169     GPIO.add_event_detect(P[1].button.port,
170                           GPIO.FALLING,
171                           callback=P2_handler,
172                           bouncetime=500)
173     return
174
175 def enable_int_0(P):
176 # Enables Autolevel interrupts for floor 0
177     GPIO.add_event_detect(P[0].FCA.port,
178                           GPIO.RISING,
179                           callback=AL0_down,
180                           bouncetime=500)
181     GPIO.add_event_detect(P[0].FCB.port,
182                           GPIO.RISING,
183                           callback=AL0_up,
184                           bouncetime=500)
185     return
186
187 def enable_int_1(P):
188     # Enables Autolevel interrupts for floor 1
189     GPIO.add_event_detect(P[1].FCA.port,
190                           GPIO.RISING,
191                           callback=AL1_down,
192                           bouncetime=500)
193     GPIO.add_event_detect(P[1].FCB.port,
194                           GPIO.RISING,
195                           callback=AL1_up,
196                           bouncetime=500)
197     return
198
199
200 def disable_int(P):
201 # Disalbes interrupts
202     GPIO.remove_event_detect(P[0].button.port)
203     GPIO.remove_event_detect(P[1].button.port)
204
205     GPIO.remove_event_detect(P[0].FCA.port)
206     GPIO.remove_event_detect(P[0].FCB.port)
207
208     GPIO.remove_event_detect(P[1].FCA.port)
209     GPIO.remove_event_detect(P[1].FCB.port)
```

```
210     return
211
212 def enable_AL(actual, P):
213 # Enables autolevel interrupts depending on the floor number
214     if(actual == 0):
215         enable_int_0(P)
216     if(actual == 1):
217         enable_int_1(P)
218     return
219
220 ##INTERRUPCIONES
221 # Sets "destination" to the number of the floor the elevator must go
222 def P1_handler(port):
223     global destination
224     destination = 0
225     return
226
227
228 def P2_handler(port):
229     global destination
230     destination = 1
231     return
232
233
234 def AL0_up(port):
235     level = 0
236     AL_up(level)
237     return
238
239
240 def AL0_down(port):
241     level = 0
242     AL_down(level)
243     return
244
245
246 def AL1_up(port):
247     level = 1
248     AL_up(level)
249     return
250
251
252 def AL1_down(port):
253     level = 1
254     AL_down(level)
255     return
256
257
258 ##MAIN
259 # Explanation: First configuration functions are executed and then
260 # enters the loop. When a button is pressed a interrupt occurs and
261 # it sets "destination" to the number of the floor pressed. If the door
262 # is not closed "destination" is set again to the actual value. If it is,
263 # the door is locked and interrupts are disabled until the movement
264 # is over. The movements is as follows:
265 #   1.-It checks if the destination floor is up or down
266 #   2.-If it is up activates motor1 and motor2 until FCB, then
267 #      deactivates motor1.
268 #   3.-If it is down opens the two valves until FCA, then close one
269 #   4.-When the destination limit switch is pressed the valve is
270 #      closed and the motor is shut down.
271 # Destination door is unlocked, "actual" value is updated and
272 # interrupts are enabled again.
273 # Autolevel: When the elevator is at rest, if one of the actual floors
274 # autolevel limit switch is activated an interupt occurs. Depending
275 # on if it is the upper or lower switch the valve or the motor
276 # is activated
277 global destination
278 [P, motors, valves] = define_IOs()
279 GPIO_config(P, motors, valves)
```

```
[actual, destination] = initialization(P, motors, valves)
enable_int_general(P)
enable_AL(actual, P)
i = 0

while(1):
    try:
        if (actual != destination and P[actual].door.value() == 1):
            disable_int(P)
            P[actual].lock.set_value(1)
            while(P[destination].FC.value() != 1):
                if (actual > destination):
                    while(P[destination].FCA.value() != 1 and i == 0):
                        valves[0].set_value(1)
                        valves[1].set_value(1)
                    valves[1].set_value(0)
                    i = 1
                elif(actual < destination):
                    while(P[destination].FCB.value() != 1 and i == 0):
                        motors[0].set_value(1)
                        motors[1].set_value(1)
                    motors[1].set_value(0)
                    i = 1

            valves[0].set_value(0)
            motors[0].set_value(0)
            P[destination].lock.set_value(0)
            actual = destination
            i = 0
            enable_int_general(P)
            enable_AL(actual, P)
    except:
        GPIO.cleanup()
        raise
```

```
001 #2P1S1DHydraulicLocks
002 #
003 #Inputs:    P0 button
004 #           P1 button
005 #           FC floor 0
006 #           FCA floor 0
007 #           tranca FCO floor 0
008 #           tranca FCI floor 0
009 #           FC floor 1
010 #           FCA floor 1
011 #           tranca FCO floor 1
012 #           tranca FCI floor 1
013 #           door closed 0
014 #           door closed 1
015 #
016 #outputs:   motor
017 #           valve
018 #           lock door 0
019 #           lock door 1
020 #           lock_out floor 0
021 #           lock_out floor 1
022
023 import RPi.GPIO as GPIO # import GPIO module
024 import time             # import time module
025 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
026
027 ##CLASES
028 class floor:
029 # Class that represents all the sensors and actuators of a floor
030     def __init__(self, number):
031         self.number = number
032         self.FC = sensor("FC" + str(self.number), 0)
033         self.FCA = sensor("FCA" + str(self.number), 0)
034         self.puerta = sensor("puerta" + str(self.number), 0)
035         self.boton = sensor("boton" + str(self.number), 0)
036         self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
037         self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
038         self.lock = actuator("lock" + str(self.number), 0)
039         self.lock_out = actuator("lock_out" + str(self.number), 0)
040
041
042 class sensor:
043 # Class that represents a sensor
044     def __init__(self,name,port):
045         self.name = name
046         self.port = port
047     def value(self):
048         return GPIO.input(self.port)
049
050
051 class actuator:
052 # Class that represents an actuator
053     def __init__(self,name,port):
054         self.name = name
055         self.port = port
056     def set_value(self, valor):
057         if(valor == 1):
058             GPIO.output(self.port, 0)
059         elif(valor == 0):
060             GPIO.output(self.port, 1)
061
062
063
064 ##FUNCIONES
065 def define_IOs():
066 # Function that configures all the objects of the program and asigns
067 # the GPIO ports to each sensor and actuator
068     p0 = floor(0)
069     p1 = floor(1)
```

```
070     P = [p0, p1]
071
072     motor = actuator("motor", 0)
073     valve = actuator("valve", 0)
074
075     P[0].FC.port = 4
076     P[0].puerta.port = 17
077     P[0].boton.port = 27
078     P[0].lock.port = 0
079     P[0].FCA.port = 14
080     P[0].lock_out.port = 0
081     P[0].lock_FCO.port = 18
082     P[0].lock_FCI.port = 23
083
084     P[1].FC.port = 10
085     P[1].puerta.port = 9
086     P[1].boton.port = 11
087     P[1].lock.port = 0
088     P[1].FCA.port = 23
089     P[1].lock_out.port = 0
090     P[1].lock_FCO.port = 7
091     P[1].lock_FCI.port = 12
092     return P, motor, valve
093
094
095 def GPIO_config(P, motor, valve):
096 # Function that sets the GPIOs as inputs or outputs
097 # Its inputs are all the sensors and actuators of the program
098     for p in P:
099         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
100         GPIO.setup(p.puerta.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
101         GPIO.setup(p.boton.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
102         GPIO.setup(p.lock.port,GPIO.OUT)
103         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
104         GPIO.setup(p.lock_out.port,GPIO.OUT)
105         GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
106         GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
107
108     GPIO.setup(motor.port,GPIO.OUT)
109     GPIO.setup(valve.port,GPIO.OUT)
110     return
111
112
113 def initialization(P, motor, valve):
114 # Function that set the initial conditions of the system
115 # Explanation: Sets "actual" to negative value and checks if the
116 # platform is in any of the floors. In that case sets "actual" to
117 # the value of the actual floor. If the platform is not in any floor,
118 # it desdends until it reaches one and sets "actual" and "destination"
119 # to the floor number.
120     actual = -1
121     for p in P:
122         if  (p.FC.value() == 1):
123             actual = p.number
124             destination = actual
125         else:
126             while(p.puerta.value() != 1):
127                 pass
128             p.lock.set_value(1)
129     if (actual < 0):
130         while((P[0].FCA.value() or
131                P[1].FCA.value()) == 0):
132             valve.set_value(1)
133         valve.set_value(0)
134
135         for p in P:
136             if(p.FCA.value() == 1):
137                 while(p.lock_FCO.value() != 1):
138                     p.lock_out.set_value(1)
139                 p.lock_out.set_value(0)
```

```
140                 while(p.FC.value() != 1):
141                     valve.set_value(1)
142                 valve.set_value(0)
143 
144                 actual = p.number
145                 destination = actual
146                 p.lock.set_value(0)
147     return actual, destination
148 
149 
150 def enable_int(P):
151 # Enables interrupts
152     GPIO.add_event_detect(P[0].boton.port,
153                           GPIO.FALLING,
154                           callback=P1_handler,
155                           bouncetime=500)
156     GPIO.add_event_detect(P[1].boton.port,
157                           GPIO.FALLING,
158                           callback=P2_handler,
159                           bouncetime=500)
160     return
161 
162 
163 def disable_int(P):
164 # Disables interrupts
165     GPIO.remove_event_detect(P[0].boton.port)
166     GPIO.remove_event_detect(P[1].boton.port)
167     return
168 
169 
170 ##INTERRUPCIONES
171 # Sets "destination" to the number of the floor the elevator must go
172 def P1_handler(port):
173     global destination
174     destination = 0
175     return
176 
177 
178 def P2_handler(port):
179     global destination
180     destination = 1
181     return
182 
183 
184 ##MAIN
185 # Explanation: First configuration functions are executed and then
186 # enters the loop. When a button is pressed a interrupt occurs and
187 # it sets "destination" to the number of the floor pressed. If the door
188 # is not closed "destination" is set again to the actual value. If it is,
189 # the door is locked and interrupts are disabled until the movement
190 # is over. If the destination is down, the movement is as follows:
191 #   1.-Platform elevates untill FCA
192 #   2.-Hydraulic lock is retracted
193 #   3.-Destination hidraulic lock is extracted and discharge valve open
194 #   4.-When the platform arrives to the destination the valve is closed
195 # If the destination is up, the movement is as follows:
196 #   1.-Platform elevates and Hydraulic lock is retracted
197 #   2.-Platform stops when it arrives to destinations FCA
198 #   3.-Destination hidraulic lock is extracted and discharge valve open
199 #   4.-When the platform arrives to the destination the valve is closed
200 # Destination door is unlocked, "actual" value is
201 # updated and interrupts are enabled again.
202 global destination
203 [P, motor, valve] = define_IOs()
204 GPIO_config(P, motor, valve)
205 [actual, destination] = initialization(P, motor, valve)
206 enable_int(P)
207 
208 while(1):
209     try:
```

```
210         if (actual != destination and P[actual].puerta.value() == 1):
211             disable_int(P)
212             P[actual].lock.set_value(1)
213             if (actual > destination):
214                 while(P[actual].FCA.value() != 1):
215                     motor.set_value(1)
216                 motor.set_value(0)
217                 while(P[actual].lock_FCI.value() != 1):
218                     P[actual].lock_out.set_value(0)
219                     P[destination].lock_out.set_value(1)
220                 valve.set_value(1)
221                 while(P[destination].lock_FCO.value() != 1):
222                     if(P[destination].FCA.value() == 1):
223                         valve.set_value(0)
224                 while(P[destination].FC.value() != 1):
225                     valve.set_value(1)
226                 valve.set_value(0)
227             elif(actual < destination):
228                 while(P[destination].lock_FCI.value() != 1):
229                     P[destination].lock_out.set_value(0)
230                 while(P[destination].FCA.value() != 1):
231                     motor.set_value(1)
232                 while(P[destination].lock_FCO.value() != 1):
233                     motor.set_value(0)
234                     P[actual].lock_out.set_value(0)
235                     P[destination].lock_out.set_value(1)
236                 while(P[destination].FC.value() != 1):
237                     valve.set_value(1)
238 
239             valve.set_value(0)
240             motor.set_value(0)
241             P[destination].lock.set_value(0)
242             actual = destination
243             enable_int(P)
244     except:
245         GPIO.cleanup()
246         raise
```

```
001 #2P1S2DHydraulicLocks
002 #
003 #Inputs:    P0 button
004 #           P1 button
005 #           FC floor 0
006 #           FCA floor 0
007 #           tranca FCO floor 0
008 #           tranca FCI floor 0
009 #           FC floor 1
010 #           FCA floor 1
011 #           tranca FCO floor 1
012 #           tranca FCI floor 1
013 #           door closed 0
014 #           door closed 1
015 #
016 #outputs:   motor
017 #           valve 1
018 #           valve 2
019 #           lock door 0
020 #           lock door 1
021 #           lock_out floor 0
022 #           lock_out floor 1
023
024 import RPi.GPIO as GPIO # import GPIO module
025 import time             # import time module
026 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
027
028 ##CLASES
029 class floor:
030 # Class that represents all the sensors and actuators of a floor
031     def __init__(self, number):
032         self.number = number
033         self.FC = sensor("FC" + str(self.number), 0)
034         self.FCA = sensor("FCA" + str(self.number), 0)
035         self.door = sensor("door" + str(self.number), 0)
036         self.button = sensor("button" + str(self.number), 0)
037         self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
038         self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
039         self.lock = actuator("lock" + str(self.number), 0)
040         self.lock_out = actuator("lock_out" + str(self.number), 0)
041
042
043 class sensor:
044 # Class that represents a sensor
045     def __init__(self,name,port):
046         self.name = name
047         self.port = port
048     def value(self):
049         return GPIO.input(self.port)
050
051 class actuator:
052 # Class that represents an actuator
053     def __init__(self,name,port):
054         self.name = name
055         self.port = port
056     def set_value(self, valor):
057         if(valor == 1):
058             GPIO.output(self.port, 0)
059         elif(valor == 0):
060             GPIO.output(self.port, 1)
061
062
063 ##FUNCIONES
064 def define_IOs():
065 # Function that configures all the objects of the program and asigns
066 # the GPIO ports to each sensor and actuator
067     p0 = floor(0)
068     p1 = floor(1)
069     P = [p0, p1]
```

```
070
071     motor = actuator("motor", 0)
072     valve1 = actuator("valve1", 0)
073     valve2 = actuator("valve2", 0)
074     valves = [valve1, valve2]
075
076     P[0].FC.port = 4
077     P[0].door.port = 17
078     P[0].button.port = 27
079     P[0].lock.port = 0
080     P[0].FCA.port = 14
081     P[0].lock_out.port = 0
082     P[0].lock_FCO.port = 18
083     P[0].lock_FCI.port = 23
084
085     P[1].FC.port = 10
086     P[1].door.port = 9
087     P[1].button.port = 11
088     P[1].lock.port = 0
089     P[1].FCA.port = 23
090     P[1].lock_out.port = 0
091     P[1].lock_FCO.port = 7
092     P[1].lock_FCI.port = 12
093
094     return P, motor, valves
095
096 def GPIO_config(P, motor, valves):
097 # Function that sets the GPIOs as inputs or outputs
098 # Its inputs are all the sensors and actuators of the program
099     for p in P:
100         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
# configuracion GPIOs
101         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
102         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
103         GPIO.setup(p.lock.port,GPIO.OUT)
104         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
105         GPIO.setup(p.lock_out.port,GPIO.OUT)
106         GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
107         GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
108
109     GPIO.setup(motor.port,GPIO.OUT)
110     GPIO.setup(valve1.port,GPIO.OUT)
111     GPIO.setup(valve2.port,GPIO.OUT)
112     return
113
114 def initialization(P, motor, valves):
115 # Function that set the initial conditions of the system
116 # Explanation: Sets "actual" to negative value and checks if the
117 # platform is in any of the floors. In that case sets "actual" to
118 # the value of the actual floor. If the platform is not in any floor,
119 # it desdends until it reaches one and sets "actual" and "destination"
120 # to the floor number.
121     actual = -1
122     for p in P:
123         if  (p.FC.value() == 1):
124             actual = p.number
125             destination = actual
126         else:
127             while(p.door.value() != 1):
128                 pass
129             p.lock.set_value(1)
130     if (actual < 0):
131         while((P[0].FCA.value() or
132                P[1].FCA.value()) == 0):
133             valves[0].set_value(1)
134         valve.set_value(0)
135
136         for p in P:
137             if(p.FCA.value() == 1):
138                 while(p.lock_FCO.value() != 1):
```

```
139                     p.lock_out.set_value(1)
140                 p.lock_out.set_value(0)
141                 while(p.FC.value() != 1):
142                     valves[0].set_value(1)
143                 valves[0].set_value(0)
144 
145                 actual = p.number
146                 destination = actual
147                 p.lock.set_value(0)
148     return actual, destination
149 
150 def enable_int(P):
151 # Enables interrupts
152     GPIO.add_event_detect(P[0].button.port,
153                           GPIO.FALLING,
154                           callback=P1_handler,
155                           bouncetime=500)
156     GPIO.add_event_detect(P[1].button.port,
157                           GPIO.FALLING,
158                           callback=P2_handler,
159                           bouncetime=500)
160     return
161 
162 def disable_int(P):
163 # Disables interrupts
164     GPIO.remove_event_detect(P[0].button.port)
165     GPIO.remove_event_detect(P[1].button.port)
166     return
167 
168 ##INTERRUPCIONES
169 # Sets "destination" to the number of the floor the elevator must go
170 def P1_handler(port):
171     global destination
172     destination = 0
173     return
174 
175 def P2_handler(port):
176     global destination
177     destination = 1
178     return
179 
180 ##MAIN
181 # Explanation: First configuration functions are executed and then
182 # enters the loop. When a button is pressed a interrupt occurs and
183 # it sets "destination" to the number of the floor pressed. If the door
184 # is not closed "destination" is set again to the actual value. If it is,
185 # the door is locked and interrupts are disabled until the movement
186 # is over. If the destination is down, the movement is as follows:
187 #   1.-Platform elevates untill FCA
188 #   2.-Hydraulic lock is retracted
189 #   3.-Destination hidraulic lock is extracted and activate vale1 and
190 #      valve2 until FCA, then deactivate valve2.
191 #   4.-When the platform arrives to the destination valve1 is closed
192 # If the destination is up, the movement is as follows:
193 #   1.-Platform elevates and Hydraulic lock is retracted
194 #   2.-Platform stops when it arrives to destinations FCA
195 #   3.-Destination hidraulic lock is extracted and discharge valve open
196 #   4.-When the platform arrives to the destination the valve is closed
197 # Destination door is unlocked, "actual" value is
198 # updated and interrupts are enabled again.
199 global destination
200 [P, motor, valves] = define_IOs()
201 GPIO_config(P, motor, valves)
202 [actual, destination] = initialization(P, motor, valves)
203 enable_int(P)
204 
205 while(1):
206     try:
207         if (actual != destination and P[actual].door.value() == 1):
208             disable_int(P)
```

```
209             P[actual].lock.set_value(1)
210             if (actual > destination):
211                 while(P[actual].FCA.value() != 1):
212                     motor.set_value(1)
213                 motor.set_value(0)
214                 while(P[actual].lock_FCI.value() != 1):
215                     P[actual].lock_out.set_value(0)
216                     P[destination].lock_out.set_value(1)
217                 valves[0].set_value(1)
218                 valves[1].set_value(1)
219                 while(P[destination].lock_FCO.value() != 1):
220                     if(P[destination].FCA.value() == 1):
221                         valves[0].set_value(0)
222                         valves[1].set_value(0)
223                 while(P[destination].FCA.value() != 1):
224                     pass
225                 valves[1].set_value(0)
226                 while(P[destination].FC.value() != 1):
227                     pass
228             elif(actual < destination):
229                 while(P[destination].lock_FCI.value() != 1):
230                     P[destination].lock_out.set_value(0)
231                 while(P[destination].FCA.value() != 1):
232                     motor.set_value(1)
233                 while(P[destination].lock_FCO.value() != 1):
234                     motor.set_value(0)
235                     P[actual].lock_out.set_value(0)
236                     P[destination].lock_out.set_value(1)
237                 while(P[destination].FC.value() != 1):
238                     valves[0].set_value(1)
239 
240             valves[0].set_value(0)
241             valves[1].set_value(0)
242             motor.set_value(0)
243             P[destination].lock.set_value(0)
244             actual = destination
245             enable_int(P)
246     except:
247         GPIO.cleanup()
248         raise
```

```
001 #2P2S2DHydraulicLocks
002 #
003 #Inputs:     P0 button
004 #            P1 button
005 #            FC floor 0
006 #            FCA floor 0
007 #            tranca FCO floor 0
008 #            tranca FCI floor 0
009 #            FC floor 1
010 #            FCA floor 1
011 #            tranca FCO floor 1
012 #            tranca FCI floor 1
013 #            door closed 0
014 #            door closed 1
015 #
016 #outputs:    motor 1
017 #            motor 2
018 #            valve 1
019 #            valve 2
020 #            lock door 0
021 #            lock door 1
022 #            lock_out floor 0
023 #            lock_out floor 1
024
025 import RPi.GPIO as GPIO # import GPIO module
026 import time             # import time module
027 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
028
029 ##CLASES
030 class floor:
031 # Class that represents all the sensors and actuators of a floor
032     def __init__(self, number):
033         self.number = number
034         self.FC = sensor("FC" + str(self.number), 0)
035         self.FCA = sensor("FCA" + str(self.number), 0)
036         self.door = sensor("door" + str(self.number), 0)
037         self.button = sensor("button" + str(self.number), 0)
038         self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
039         self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
040         self.lock = actuator("lock" + str(self.number), 0)
041         self.lock_out = actuator("lock_out" + str(self.number), 0)
042
043
044 class sensor:
045 # Class that represents a sensor
046     def __init__(self,name,port):
047         self.name = name
048         self.port = port
049     def value(self):
050         return GPIO.input(self.port)
051
052
053 class actuator:
054 # Class that represents an actuator
055     def __init__(self,name,port):
056         self.name = name
057         self.port = port
058     def set_value(self, valor):
059         if(valor == 1):
060             GPIO.output(self.port, 0)
061         elif(valor == 0):
062             GPIO.output(self.port, 1)
063
064
065 ##FUNCIONES
066 def define_IOs():
067 # Function that configures all the objects of the program and asigns
068 # the GPIO ports to each sensor and actuator
069     p0 = floor(0)
```

```
070     p1 = floor(1)
071     P = [p0, p1]
072
073     motor1 = actuator("motor1", 3)
074     motor2 = actuator("motor2", 5)
075     valve1 = actuator("valve1", 37)
076     valve2 = actuator("valve2", 40)
077     valves = [valve1, valve2]
078     motors = [motor1, motor2]
079
080     P[0].FC.port = 7
081     P[0].door.port = 11
082     P[0].button.port = 13
083     P[0].lock.port = 15
084     P[0].FCA.port = 8
085     P[0].lock_out.port = 10
086     P[0].lock_FCO.port = 12
087     P[0].lock_FCI.port = 16
088
089     P[1].FC.port = 19
090     P[1].door.port = 21
091     P[1].button.port = 23
092     P[1].lock.port = 27
093     P[1].FCA.port = 18
094     P[1].lock_out.port = 22
095     P[1].lock_FCO.port = 24
096     P[1].lock_FCI.port = 26
097     return P, motors, valves
098
099
100 def GPIO_config(P, motors, valves):
101 # Function that sets the GPIOs as inputs or outputs
102 # Its inputs are all the sensors and actuators of the program
103     for p in P:
104         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
105         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
106         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
107         GPIO.setup(p.lock.port,GPIO.OUT)
108         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
109         GPIO.setup(p.lock_out.port,GPIO.OUT)
110         GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
111         GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
112
113     GPIO.setup(motor1.port,GPIO.OUT)
114     GPIO.setup(motor2.port,GPIO.OUT)
115     GPIO.setup(valve1.port,GPIO.OUT)
116     GPIO.setup(valve2.port,GPIO.OUT)
117     return
118
119
120 def initialization(P, motors, valves):
121 # Function that set the initial conditions of the system
122 # Explanation: Sets "actual" to negative value and checks if the
123 # platform is in any of the floors. In that case sets "actual" to
124 # the value of the actual floor. If the platform is not in any floor,
125 # it desdends until it reaches one and sets "actual" and "destination"
126 # to the floor number.
127     actual = -1
128     for p in P:
129         if  (p.FC.value() == 1):
130             actual = p.number
131             destination = actual
132         else:
133             while(p.door.value() != 1):
134                 pass
135             p.lock.set_value(1)
136     if (actual < 0):
137         while((P[0].FCA.value() or
138                P[1].FCA.value()) == 0):
139             valves[0].set_value(1)
```

```
140         valves[0].set_value(0)
141
142         for p in P:
143             if(p.FCA.value() == 1):
144                 while(p.lock_FCO.value() != 1):
145                     p.lock_out.set_value(1)
146                 p.lock_out.set_value(0)
147                 while(p.FC.value() != 1):
148                     valves[0].set_value(1)
149                 valves[0].set_value(0)
150
151                 actual = p.number
152                 destination = actual
153                 p.lock.set_value(0)
154     return actual, destination
155
156
157 def enable_int(P):
158 # Enables interrupts
159     GPIO.add_event_detect(P[0].button.port,
160                           GPIO.FALLING,
161                           callback=P1_handler,
162                           bouncetime=500)
163     GPIO.add_event_detect(P[1].button.port,
164                           GPIO.FALLING,
165                           callback=P2_handler,
166                           bouncetime=500)
167     return
168
169
170 def disable_int(P):
171 # Disables interrupts
172     GPIO.remove_event_detect(P[0].button.port)
173     GPIO.remove_event_detect(P[1].button.port)
174     return
175
176
177 ##INTERRUPCIONES
178 # Sets "destination" to the number of the floor the elevator must go
179 def P1_handler(port):
180     global destination
181     destination = 0
182     return
183
184
185 def P2_handler(port):
186     global destination
187     destination = 1
188     return
189
190
191 ##MAIN
192 # Explanation: First configuration functions are executed and then
193 # enters the loop. When a button is pressed a interrupt occurs and
194 # it sets "destination" to the number of the floor pressed. If the door
195 # is not closed "destination" is set again to the actual value. If it is,
196 # the door is locked and interrupts are disabled until the movement
197 # is over. If the destination is down, the movement is as follows:
198 #   1.-Platform elevates untill FCA
199 #   2.-Hydraulic lock is retracted
200 #   3.-Destination hidraulic lock is extracted and activate vale1 and
201 #      valve2 until FCA, then deactivate valve2.
202 #   4.-When the platform arrives to the destination valve1 is closed
203 # If the destination is up, the movement is as follows:
204 #   1.-motor1 and motor2 are activated until destination FC,
205 #      then motor2 is deactivated
206 #   2.-Hydraulic lock is retracted
207 #   3.-motor1 is deactivated when it arrives to destinations FCA
208 #   4.-Destination hidraulic lock is extracted and valve1 open
209 #   5.-When the platform arrives to the destination valve1 is closed
```

```
# Destination door is unlocked, "actual" value is
# updated and interrupts are enabled again.
global destination
[P, motors, valves] = define_IOs()
GPIO_config(P, motors, valves)
[actual, destination] = initialization(P, motors, valves)
enable_int(P)

while(1):
    try:
        if (actual != destination and P[actual].door.value() == 1):
            disable_int(P)
            P[actual].lock.set_value(1)
            if (actual > destination):
                while(P[actual].FCA.value() != 1):
                    motors[0].set_value(1)
                motors[0].set_value(0)
                while(P[actual].lock_FCI.value() != 1):
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                valves[0].set_value(1)
                valves[1].set_value(1)
                while(P[destination].lock_FCO.value() != 1):
                    if(P[destination].FCA.value() == 1):
                        valves[0].set_value(0)
                        valves[1].set_value(0)
                while(P[destination].FCA.value() != 1):
                    pass
                valves[1].set_value(0)
                while(P[destination].FC.value() != 1):
                    pass
            elif(actual < destination):
                while(P[destination].lock_FCI.value() != 1):
                    P[destination].lock_out.set_value(0)
                while(P[destination].FC.value() != 1):
                    motors[0].set_value(1)
                    motors[1].set_value(1)
                motors[1].set_value(0)
                while(P[destination].FCA.value() != 1):
                    pass
                motors[0].set_value(0)
                while(P[destination].lock_FCO.value() != 1):
                    motors[0].set_value(0)
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                while(P[destination].FC.value() != 1):
                    valves[0].set_value(1)

            valves[0].set_value(0)
            valves[1].set_value(0)
            motors[0].set_value(0)
            motors[1].set_value(0)
            P[destination].lock.set_value(0)
            actual = destination
            enable_int(P)
    except:
        GPIO.cleanup()
        raise
```

```
#3P1S1DStandard
#
#Inputs:    P0 button
#           P1 button
#           P2 button
#           FC floor 0
#           FC floor 1
#           FC floor 2
#           door closed 0
#           door closed 1
#           door closed 2
#
#outputs:   motor
#           valve
#           lock door 0
#           lock door 1
#           lock door 2

import RPi.GPIO as GPIO     # import GPIO module
import time                 # import time module
GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD


##CLASES
class floor:
# Class that represents all the sensors and actuators of a floor
    def __init__(self, number):
        self.number = number
        self.FC = sensor("FC" + str(self.number), 0)
        self.door = sensor("door" + str(self.number), 0)
        self.button = sensor("button" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)


class sensor:
# Class that represents a sensor
    def __init__(self, name, port):
        self.name = name
        self.port = port

    def value(self):
        return GPIO.input(self.port)


class actuator:
# Class that represents an actuator
    def __init__(self, name, port):
        self.name = name
        self.port = port

    def set_value(self, valor):
        if(valor == 1):
            GPIO.output(self.port, 0)
        elif(valor == 0):
            GPIO.output(self.port, 1)


##FUNCIONES
def define_IOs():
# Function that configures all the objects of the program and asigns
# the GPIO ports to each sensor and actuator
    p0 = floor(0)
    p1 = floor(1)
    p2 = floor(2)
    P = [p0, p1, p2]

    motor = actuator("motor", 37)
    valve = actuator("valve", 38)

```

```
070     P[0].FC.port = 7
071     P[0].door.port = 11
072     P[0].button.port = 13
073     P[0].lock.port = 15
074
075     P[1].FC.port = 19
076     P[1].door.port = 21
077     P[1].button.port = 23
078     P[1].lock.port = 29
079
080     P[2].FC.port = 31
081     P[2].door.port = 33
082     P[2].button.port = 32
083     P[2].lock.port = 35
084     return P, motor, valve
085
086
087 def GPIO_config(P, motor, valve):
088 # Function that sets the GPIOs as inputs or outputs
089 # Its inputs are all the sensors and actuators of the program
090     for p in P:
091         GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
092         GPIO.setup(p.door.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
093         GPIO.setup(p.button.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
094         GPIO.setup(p.lock.port, GPIO.OUT)
095
096
097     GPIO.setup(motor.port, GPIO.OUT)
098     GPIO.setup(valve.port, GPIO.OUT)
099     return
100
101
102 def initialization(P, motor, valve):
103 # Function that set the initial conditions of the system
104 # Explanation: Sets "actual" to negative value and checks if the
105 # platform is in any of the floors. In that case sets "actual" to
106 # the value of the actual floor. If the platform is not in any floor,
107 # it desdends until it reaches one and sets "actual" and "destination"
108 # to the floor number.
109     actual = -1
110     for p in P:
111         if  (p.FC.value() == 1):
112             actual = p.number
113             destination = actual
114         else:
115             while(p.door.value() != 1):
116                 pass
117             p.lock.set_value(1)
118     if (actual < 0):
119         while((P[0].door.value() and
120                P[1].door.value()) != 1):
121             pass
122         for p in P:
123             p.lock.set_value(1)
124         while((P[0].FC.value() or
125                P[1].FC.value()) == 0):
126             valve.set_value(1)
127         valve.set_value(0)
128
129         for p in P:
130             if(p.FC.value() == 1):
131                 actual = p.number
132                 destination = actual
133                 p.lock.set_value(0)
134                 break
135     return actual, destination
136
137
138 def enable_int(P):
139 # Enables interrupts
```

```
140     GPIO.add_event_detect(P[0].button.port,
141                           GPIO.FALLING,
142                           callback=P1_handler,
143                           bouncetime=500)
144     GPIO.add_event_detect(P[1].button.port,
145                           GPIO.FALLING,
146                           callback=P2_handler,
147                           bouncetime=500)
148     GPIO.add_event_detect(P[2].button.port,
149                           GPIO.FALLING,
150                           callback=P3_handler,
151                           bouncetime=500)
152     return
153
154
155 def disable_int(P):
156 # Disables interrupts
157     GPIO.remove_event_detect(P[0].button.port)
158     GPIO.remove_event_detect(P[1].button.port)
159     GPIO.remove_event_detect(P[2].button.port)
160     return
161
162
163 ##INTERRUPCIONES
164 # Sets "destination" to the number of the floor the elevator must go
165 def P1_handler(port):
166     global destination
167     destination = 0
168     return
169
170
171 def P2_handler(port):
172     global destination
173     destination = 1
174     return
175
176
177 def P3_handler(port):
178     global destination
179     destination = 2
180     return
181
182
183 ##MAIN
184 # Explanation: First configuration functions are executed and then
185 # enters the loop. When a button is pressed a interrupt occurs and
186 # it sets "destination" to the number of the floor pressed. If the door
187 # is not closed "destination" is set again to the actual value. If it is,
188 # the door is locked and interrupts are disabled until the movement
189 # is over. The movements is as follows:
190 #   1.-It checks if the destination floor is up or down
191 #   2.-If it is up activates the motor
192 #   3.-If it is down opens the valve
193 #   4.-When the destination limit switch is pressed the valve is
194 #      closed and the motor is shut down.
195 # Destination door is unlocked, "actual" value is updated and
196 # interrupts are enabled again.
197 global destination
198 [P, motor, valve] = define_IOs()
199 GPIO_config(P, motor, valve)
200 [actual, destination] = initialization(P, motor, valve)
201 enable_int(P)
202
203 while(1):
204     try:
205         time.sleep(0.3)
206         if (actual != destination and P[actual].door.value() == 1):
207             disable_int(P)
208             P[actual].lock.set_value(1)
209             while(P[destination].FC.value() != 1):
```

```
210                 if (actual > destination):
211                     valve.set_value(1)
212                 elif(actual < destination):
213                     motor.set_value(1)
214 
215             valve.set_value(0)
216             motor.set_value(0)
217             P[destination].lock.set_value(0)
218             actual = destination
219             enable_int(P)
220         else:
221             destination = actual
222     except:
223         GPIO.cleanup()
224         raise
```

```
#3P1S1DAuto-level
#
#Inputs:    P0 button
#           P1 button
#           P2 button
#           FC floor 0
#           FCA floor 0
#           FCB floor 0
#           FC floor 1
#           FCA floor 1
#           FCB floor 1
#           FC floor 2
#           FCA floor 2
#           FCB floor 2
#           door closed 0
#           door closed 1
#           door closed 2
#
#outputs:   motor
#           valve
#           lock door 0
#           lock door 1
#           lock door 2

import RPi.GPIO as GPIO # import GPIO module
import time             # import time module
GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD


##CLASES
class floor:
# Class that represents all the sensors and actuators of a floor
    def __init__(self, number):
        self.number = number
        self.FC = sensor("FC" + str(self.number), 0)
        self.FCA = sensor("FCA" + str(self.number), 0)
        self.FCB = sensor("FCB" + str(self.number), 0)
        self.door = sensor("door" + str(self.number), 0)
        self.button = sensor("button" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)


class sensor:
# Class that represents a sensor
    def __init__(self, name, port):
        self.name = name
        self.port = port

    def set_value(self, valor):
        if(valor == 1):
            GPIO.output(self.port, 0)
        elif(valor == 0):
            GPIO.output(self.port, 1)


class actuator:
# Class that represents an actuator
    def __init__(self, name, port):
        self.name = name
        self.port = port

    def set_value(self, valor):
        GPIO.output(self.port, valor)


##FUNCIONES
def define_IOs():
# Function that configures all the objects of the program and asigns
# the GPIO ports to each sensor and actuator
```

```
070     p0 = floor(0)
071     p1 = floor(1)
072     p2 = floor(2)
073     P = [p0, p1, p2]
074
075     motor = actuator("motor", 0)
076     valve = actuator("valve", 0)
077
078     P[0].FC.port = 4
079     P[0].door.port = 17
080     P[0].button.port = 27
081     P[0].lock.port = 0
082     P[0].FCA.port = 14
083     P[0].FCB.port = 15
084
085     P[1].FC.port = 10
086     P[1].door.port = 9
087     P[1].button.port = 11
088     P[1].lock.port = 0
089     P[1].FCA.port = 25
090     P[1].FCB.port = 8
091
092     P[2].FC.port = 5
093     P[2].door.port = 6
094     P[2].button.port = 13
095     P[2].lock.port = 0
096     P[2].FCA.port = 16
097     P[2].FCB.port = 20
098     return P, motor, valve
099
100
101 def GPIO_config(P, motor, valve):
102 # Function that sets the GPIOs as inputs or outputs
103 # Its inputs are all the sensors and actuators of the program
104     for p in P:
105         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
106         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
107         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
108         GPIO.setup(p.lock.port,GPIO.OUT)
109         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
110         GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
111
112     GPIO.setup(motor.port,GPIO.OUT)
113     GPIO.setup(valve.port,GPIO.OUT)
114     return
115
116
117 def initialization(P, motor, valve):
118 # Function that set the initial conditions of the system
119 # Explanation: Sets "actual" to negative value and checks if the
120 # platform is in any of the floors. In that case sets "actual" to
121 # the value of the actual floor. If the platform is not in any floor,
122 # it desdends until it reaches one and sets "actual" and "destination"
123 # to the floor number.
124     actual = -1
125     for p in P:
126         if  (p.FC.value() == 1):
127             actual = p.number
128             destination = actual
129         else:
130             while(p.door.value() != 1):
131                 pass
132             p.lock.set_value(1)
133     if (actual < 0):
134         while((P[0].door.value() and
135                P[1].door.value()) != 1):
136             pass
137         for p in P:
138             p.lock.set_value(1)
139         while((P[0].FC.value() or
```

```
140                 P[1].FC.value()) == 0):
141             valve.set_value(1)
142         valve.set_value(0)
143 
144         for p in P:
145             if(p.FC.value() == 1):
146                 actual = p.number
147                 destination = actual
148                 p.lock.set_value(0)
149                 break
150     return actual, destination
151 
152 
153 def AL_up(level):
154     while(P[level].FC.value() != 1):
155         motor.set_value(1)
156     motor.set_value(0)
157     return
158 
159 
160 def AL_down(level):
161     while(P[level].FC.value() != 1):
162         valve.set_value(1)
163     valve.set_value(0)
164     return
165 
166 
167 def enable_int_general(P):
168 # Enables button interrupts
169     GPIO.add_event_detect(P[0].button.port,
170                           GPIO.FALLING,
171                           callback=P1_handler,
172                           bouncetime=500)
173     GPIO.add_event_detect(P[1].button.port,
174                           GPIO.FALLING,
175                           callback=P2_handler,
176                           bouncetime=500)
177     GPIO.add_event_detect(P[2].button.port,
178                           GPIO.FALLING,
179                           callback=P3_handler,
180                           bouncetime=500)
181     return
182 
183 def enable_int_0(P):
184 # Enables Autolevel interrupts for floor 0
185     GPIO.add_event_detect(P[0].FCA.port,
186                           GPIO.RISING,
187                           callback=AL0_down,
188                           bouncetime=500)
189     GPIO.add_event_detect(P[0].FCB.port,
190                           GPIO.RISING,
191                           callback=AL0_up,
192                           bouncetime=500)
193     return
194 
195 def enable_int_1(P):
196 # Enables Autolevel interrupts for floor 1
197     GPIO.add_event_detect(P[1].FCA.port,
198                           GPIO.RISING,
199                           callback=AL1_down,
200                           bouncetime=500)
201     GPIO.add_event_detect(P[1].FCB.port,
202                           GPIO.RISING,
203                           callback=AL1_up,
204                           bouncetime=500)
205     return
206 
207 def enable_int_2(P):
208 # Enables Autolevel interrupts for floor 2
209 GPIO.add_event_detect(P[1].FCA.port,
```

```
210                             GPIO.RISING,
211                             callback=AL2_down,
212                             bouncetime=500)
213     GPIO.add_event_detect(P[2].FCB.port,
214                             GPIO.RISING,
215                             callback=AL2_up,
216                             bouncetime=500)
217     return
218
219
220 def disable_int(P):
221 # Disalbes interrupts
222     GPIO.remove_event_detect(P[0].button.port)
223     GPIO.remove_event_detect(P[1].button.port)
224     GPIO.remove_event_detect(P[2].button.port)
225
226     GPIO.remove_event_detect(P[0].FCA.port)
227     GPIO.remove_event_detect(P[0].FCB.port)
228
229     GPIO.remove_event_detect(P[1].FCA.port)
230     GPIO.remove_event_detect(P[1].FCB.port)
231
232     GPIO.remove_event_detect(P[2].FCA.port)
233     GPIO.remove_event_detect(P[2].FCB.port)
234     return
235
236 def enable_AL(actual, P):
237 # Enables autolevel interrupts depending on the floor number
238     if(actual == 0):
239         enable_int_0(P)
240     if(actual == 1):
241         enable_int_1(P)
242     if(actual == 2):
243         enable_int_2(P)
244     return
245
246 ##INTERRUPCIONES
247 # Sets "destination" to the number of the floor the elevator must go
248 def P1_handler(port):
249     global destination
250     destination = 0
251     return
252
253
254 def P2_handler(port):
255     global destination
256     destination = 1
257     return
258
259
260 def P3_handler(port):
261     global destination
262     destination = 2
263     return
264
265
266 def AL0_up(port):
267     level = 0
268     AL_up(level)
269     return
270
271
272 def AL0_down(port):
273     level = 0
274     AL_down(level)
275     return
276
277
278 def AL1_up(port):
279     level = 1
```

```
280     AL_up(level)
281     return
282
283
284 def AL1_down(port):
285     level = 1
286     AL_down(level)
287     return
288
289
290 def AL2_up(port):
291     level = 2
292     AL_up(level)
293     return
294
295
296 def AL2_down(port):
297     level = 2
298     AL_down(level)
299     return
300
301
302 ##MAIN
303 # Explanation: First configuration functions are executed and then
304 # enters the loop. When a button is pressed a interrupt occurs and
305 # it sets "destination" to the number of the floor pressed. If the door
306 # is not closed "destination" is set again to the actual value. If it is,
307 # the door is locked and interrupts are disabled until the movement
308 # is over. The movements is as follows:
309 #   1.-It checks if the destination floor is up or down
310 #   2.-If it is up activates the motor
311 #   3.-If it is down opens the valve
312 #   4.-When the destination limit switch is pressed the valve is
313 #      closed and the motor is shut down.
314 # Destination door is unlocked, "actual" value is updated and
315 # interrupts are enabled again.
316 # Autolevel: When the elevator is at rest, if one of the actual floors
317 # autolevel limit switch is activated an interupt occurs. Depending
318 # on if it is the upper or lower switch the valve or the motor
319 # is activated
320 global destination
321 [P, motor, valve] = define_IOs()
322 GPIO_config(P, motor, valve)
323 [actual, destination] = initialization(P, motor, valve)
324 enable_int_general(P)
325 enable_AL(actual, P)
326
327 while(1):
328     try:
329         time.sleep(0.3)
330         if (actual != destination and P[actual].door.value() == 1):
331             disable_int(P)
332             P[actual].lock.set_value(1)
333             while(P[destination].FC.value() != 1):
334                 if (actual > destination):
335                     valve.set_value(1)
336                 elif(actual < destination):
337                     motor.set_value(1)
338
339             valve.set_value(0)
340             motor.set_value(0)
341             P[destination].lock.set_value(0)
342             actual = destination
343             enable_int_general(P)
344             enable_AL(actual, P)
345         else:
346             destination = actual
347     except:
348         GPIO.cleanup()
349         raise
```

```
001 #3P1S2DAuto-level
002 #
003 #Inputs:     P0 button
004 #            P1 button
005 #            P2 button
006 #            FC floor 0
007 #            FCA floor 0
008 #            FCB floor 0
009 #            FC floor 1
010 #            FCA floor 1
011 #            FCB floor 1
012 #            FC floor 2
013 #            FCA floor 2
014 #            FCB floor 2
015 #            door closed 0
016 #            door closed 1
017 #            door closed 2
018 #
019 #outputs:    motor
020 #            valve 1
021 #            valve 2
022 #            lock door 0
023 #            lock door 1
024 #            lock door 2
025
026 import RPi.GPIO as GPIO # import GPIO module
027 import time             # import time module
028 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
029
030 ##CLASES
031 class floor:
032 # Class that represents all the sensors and actuators of a floor
033     def __init__(self, number):
034         self.number = number
035         self.FC = sensor("FC" + str(self.number), 0)
036         self.FCA = sensor("FCA" + str(self.number), 0)
037         self.FCB = sensor("FCB" + str(self.number), 0)
038         self.door = sensor("door" + str(self.number), 0)
039         self.button = sensor("button" + str(self.number), 0)
040         self.lock = actuator("lock" + str(self.number), 0)
041
042 class sensor:
043 # Class that represents a sensor
044     def __init__(self,name,port):
045         self.name = name
046         self.port = port
047
048     def value(self):
049         return GPIO.input(self.port)
050
051 class actuator:
052 # Class that represents an actuator
053     def __init__(self, name, port):
054         self.name = name
055         self.port = port
056
057     def set_value(self, valor):
058         if(valor == 1):
059             GPIO.output(self.port, 0)
060         elif(valor == 0):
061             GPIO.output(self.port, 1)
062
063
064 ##FUNCIONES
065 def define_IOs():
066 # Function that configures all the objects of the program and asigns
067 # the GPIO ports to each sensor and actuator
068     p0 = floor(0)
069     p1 = floor(1)
```

```
070     p2 = floor(2)
071     P = [p0, p1, p2]
072
073     motor = actuator("motor", 0)
074     valve1 = actuator("valve1", 0)
075     valve2 = actuator("valve2", 0)
076     valves = [valve1, valve2]
077
078     P[0].FC.port = 4
079     P[0].door.port = 17
080     P[0].button.port = 27
081     P[0].lock.port = 0
082     P[0].FCA.port = 14
083     P[0].FCB.port = 15
084
085     P[1].FC.port = 10
086     P[1].door.port = 9
087     P[1].button.port = 11
088     P[1].lock.port = 0
089     P[1].FCA.port = 25
090     P[1].FCB.port = 8
091
092     P[2].FC.port = 5
093     P[2].door.port = 6
094     P[2].button.port = 13
095     P[2].lock.port = 0
096     P[2].FCA.port = 16
097     P[2].FCB.port = 20
098     return P, motor, valves
099
100 def GPIO_config(P, motor, valves):
101 # Function that sets the GPIOs as inputs or outputs
102 # Its inputs are all the sensors and actuators of the program
103     for p in P:
104         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
105         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
106         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
107         GPIO.setup(p.lock.port,GPIO.OUT)
108         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
109         GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
110
111     GPIO.setup(motor.port,GPIO.OUT)
112     GPIO.setup(valve1.port,GPIO.OUT)
113     GPIO.setup(valve2.port,GPIO.OUT)
114     return
115
116 def initialization(P, motor, valves):
117 # Function that set the initial conditions of the system
118 # Explanation: Sets "actual" to negative value and checks if the
119 # platform is in any of the floors. In that case sets "actual" to
120 # the value of the actual floor. If the platform is not in any floor,
121 # it desdends until it reaches one and sets "actual" and "destination"
122 # to the floor number.
123     actual = -1
124     for p in P:
125         if  (p.FC.value() == 1):
126             actual = p.number
127             destination = actual
128         else:
129             while(p.door.value() != 1):
130                 pass
131             p.lock.set_value(1)
132     if (actual < 0):
133         while((P[0].door.value() and
134                P[1].door.value()) != 1):
135             pass
136         for p in P:
137             p.lock.set_value(1)
138         while((P[0].FC.value() or
139                P[1].FC.value()) == 0):
```

```
140             valves[0].set_value(1)
141         valves[0].set_value(0)
142 
143         for p in P:
144             if(p.FC.value() == 1):
145                 actual = p.number
146                 destination = actual
147                 p.lock.set_value(0)
148                 break
149     return actual, destination
150 
151 def AL_up(level):
152     while(P[level].FC.value() != 1):
153         motor.set_value(1)
154     motor.set_value(0)
155     return
156 
157 def AL_down(level):
158     while(P[level].FC.value() != 1):
159         valves[0].set_value(1)
160         valves[1].set_value(0)
161     valves[0].set_value(0)
162     return
163 
164 def enable_int_general(P):
165 # Enables button interrupts
166     GPIO.add_event_detect(P[0].button.port,
167                           GPIO.FALLING,
168                           callback=P1_handler,
169                           bouncetime=500)
170     GPIO.add_event_detect(P[1].button.port,
171                           GPIO.FALLING,
172                           callback=P2_handler,
173                           bouncetime=500)
174     GPIO.add_event_detect(P[2].button.port,
175                           GPIO.FALLING,
176                           callback=P3_handler,
177                           bouncetime=500)
178     return
179 
180 def enable_int_0(P):
181 # Enables Autolevel interrupts for floor 0
182     GPIO.add_event_detect(P[0].FCA.port,
183                           GPIO.RISING,
184                           callback=AL0_down,
185                           bouncetime=500)
186     GPIO.add_event_detect(P[0].FCB.port,
187                           GPIO.RISING,
188                           callback=AL0_up,
189                           bouncetime=500)
190     return
191 
192 def enable_int_1(P):
193 # Enables Autolevel interrupts for floor 1
194     GPIO.add_event_detect(P[1].FCA.port,
195                           GPIO.RISING,
196                           callback=AL1_down,
197                           bouncetime=500)
198     GPIO.add_event_detect(P[1].FCB.port,
199                           GPIO.RISING,
200                           callback=AL1_up,
201                           bouncetime=500)
202     return
203 
204 def enable_int_2(P):
205 # Enables Autolevel interrupts for floor 2
206     GPIO.add_event_detect(P[2].FCA.port,
207                           GPIO.RISING,
208                           callback=AL2_down,
209                           bouncetime=500)
```

```
210     GPIO.add_event_detect(P[2].FCB.port,
211                           GPIO.RISING,
212                           callback=AL2_up,
213                           bouncetime=500)
214     return
215
216 def disable_int(P):
217 # Disalbes interrupts
218     GPIO.remove_event_detect(P[0].button.port)
219     GPIO.remove_event_detect(P[1].button.port)
220     GPIO.remove_event_detect(P[2].button.port)
221
222     GPIO.remove_event_detect(P[0].FCA.port)
223     GPIO.remove_event_detect(P[0].FCB.port)
224
225     GPIO.remove_event_detect(P[1].FCA.port)
226     GPIO.remove_event_detect(P[1].FCB.port)
227
228     GPIO.remove_event_detect(P[2].FCA.port)
229     GPIO.remove_event_detect(P[2].FCB.port)
230     return
231
232 def enable_AL(actual, P):
233 # Enables autolevel interrupts depending on the floor number
234     if(actual == 0):
235         enable_int_0(P)
236     if(actual == 1):
237         enable_int_1(P)
238     if(actual == 2):
239         enable_int_2(P)
240     return
241
242 ##INTERRUPCIONES
243 # Sets "destination" to the number of the floor the elevator must go
244 def P1_handler(self):
245     global destination
246     destination = 0
247     return
248
249 def P2_handler(self):
250     global destination
251     destination = 1
252     return
253
254 def P3_handler(self):
255     global destination
256     destination = 2
257     return
258
259 def AL0_up(self):
260     level = 0
261     AL_up(level)
262     return
263
264 def AL0_down(self):
265     level = 0
266     AL_down(level)
267     return
268
269 def AL1_up(self):
270     level = 1
271     AL_up(level)
272     return
273
274 def AL1_down(self):
275     level = 1
276     AL_down(level)
277     return
278
279 def AL2_up(self):
```

```
    level = 2
    AL_up(level)
    return

def AL2_down(self):
    level = 2
    AL_down(level)
    return

##MAIN
# Explanation: First configuration functions are executed and then
# enters the loop. When a button is pressed a interrupt occurs and
# it sets "destination" to the number of the floor pressed. If the door
# is not closed "destination" is set again to the actual value. If it is,
# the door is locked and interrupts are disabled until the movement
# is over. The movements is as follows:
#   1.-It checks if the destination floor is up or down
#   2.-If it is up activates the motor
#   3.-If it is down activates vale1 and valve2 until FCA,
#      then close valve2
#   4.-When the destination limit switch is pressed the valve is
#      closed and the motor is shut down.
# Destination door is unlocked, "actual" value is updated and
# interrupts are enabled again.
# Autolevel: When the elevator is at rest, if one of the actual floors
# autolevel limit switch is activated an interupt occurs. Depending
# on if it is the upper or lower switch the valve or the motor
# is activated
global destination
[P, motor, valves] = define_IOs()
GPIO_config(P, motor, valves)
[actual, destination] = initialization(P, motor, valves)
enable_int_general(P)
enable_AL(actual, P)
i = 0

while(1):
    try:
        sleep(0.3)
        if (actual != destination and P[actual].door.value() == 1):
            disable_int(P)
            P[actual].lock.set_value(1)
            while(P[destination].FC.value() != 1):
                if (actual > destination):
                    while(P[destination].FCA.value() != 1 and i == 0)
                        valves[0].set_value(1)
                        valves[1].set_value(1)
                    valves[1].set_value(0)
                    i = 1
                elif(actual < destination):
                    motor.set_value(1)

            valve[0].set_value(0)
            motor.set_value(0)
            P[destination].lock.set_value(0)
            actual = destination
            i = 0
            enable_int_general(P)
            enable_AL(actual, P)
        else:
            destination = actual
        except:
            GPIO.cleanup()
            raise
```

```
001 #3P2S2DAuto-level
002 #
003 #Inputs:     P0 button
004 #            P1 button
005 #            P2 button
006 #            FC floor 0
007 #            FCA floor 0
008 #            FCB floor 0
009 #            FC floor 1
010 #            FCA floor 1
011 #            FCB floor 1
012 #            FC floor 2
013 #            FCA floor 2
014 #            FCB floor 2
015 #            door closed 0
016 #            door closed 1
017 #            door closed 2
018 #
019 #outputs:    motor 1
020 #            motor 2
021 #            valve 1
022 #            valve 2
023 #            lock door 0
024 #            lock door 1
025 #            lock door 2
026
027 import RPi.GPIO as GPIO # import GPIO module
028 import time             # import time module
029 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
030
031
032 ##CLASES
033 class floor:
034 # Class that represents all the sensors and actuators of a floor
035     def __init__(self, number):
036         self.number = number
037         self.FC = sensor("FC" + str(self.number), 0)
038         self.FCA = sensor("FCA" + str(self.number), 0)
039         self.FCB = sensor("FCB" + str(self.number), 0)
040         self.door = sensor("door" + str(self.number), 0)
041         self.button = sensor("button" + str(self.number), 0)
042         self.lock = actuator("lock" + str(self.number), 0)
043
044
045 class sensor:
046 # Class that represents a sensor
047     def __init__(self,name,port):
048         self.name = name
049         self.port = port
050
051     def value(self):
052         return GPIO.input(self.port)
053
054
055 class actuator:
056 # Class that represents an actuator
057     def __init__(self, name, port):
058         self.name = name
059         self.port = port
060
061     def set_value(self, valor):
062         if(valor == 1):
063             GPIO.output(self.port, 0)
064         elif(valor == 0):
065             GPIO.output(self.port, 1)
066
067
068 ##FUNCIONES
069 def define_IOs():
```

```
070 # Function that configures all the objects of the program and asigns
071 # the GPIO ports to each sensor and actuator
072     p0 = floor(0)
073     p1 = floor(1)
074     p2 = floor(2)
075     P = [p0, p1, p2]
076 
077     motor1 = actuator("motor1", 15)
078     motor2 = actuator("motor2", 16)
079     valve1 = actuator("valve1", 18)
080     valve2 = actuator("valve2", 40)
081     valves = [valve1, valve2]
082     motors = [motor1, motor2]
083 
084     P[0].FC.port = 7            # amarillo
085     P[0].door.port = 11       # azul
086     P[0].button.port = 13        # morado
087     P[0].lock.port = 15        # -----
088     P[0].FCA.port = 8          # naranja
089     P[0].FCB.port = 10          # verde
090 
091     P[1].FC.port = 19           # amarillo
092     P[1].door.port = 21        # azul
093     P[1].button.port = 23         # morado
094     P[1].lock.port = 26        # -----
095     P[1].FCA.port = 22          # naranja
096     P[1].FCB.port = 24           # verde
097 
098     P[2].FC.port = 29           # amarillo
099     P[2].door.port = 31        # azul
100     P[2].button.port = 33         # morado
101     P[2].lock.port = 35        # -----
102     P[2].FCA.port = 36          # naranja
103     P[2].FCB.port = 38          # verde
104     return P, motors, valves
105 
106 
107 def GPIO_config(P, motors, valves):
108 # Function that sets the GPIOs as inputs or outputs
109 # Its inputs are all the sensors and actuators of the program
110     for p in P:
111         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
112         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
113         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
114         GPIO.setup(p.lock.port,GPIO.OUT)
115         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
116         GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
117 
118     GPIO.setup(motors[0].port,GPIO.OUT)
119     GPIO.setup(motors[1].port,GPIO.OUT)
120     GPIO.setup(valves[0].port,GPIO.OUT)
121     GPIO.setup(valves[1].port,GPIO.OUT)
122     return
123 
124 
125 def initialization(P, motors, valves):
126 # Function that set the initial conditions of the system
127 # Explanation: Sets "actual" to negative value and checks if the
128 # platform is in any of the floors. In that case sets "actual" to
129 # the value of the actual floor. If the platform is not in any floor,
130 # it desdends until it reaches one and sets "actual" and "destination"
131 # to the floor number.
132     actual = -1
133     for p in P:
134         if  (p.FC.value() == 1):
135             actual = p.number
136             destination = actual
137         else:
138             while(p.door.value() != 1):
139                 pass
```

```
140             p.lock.set_value(1)
141     if (actual < 0):
142         while((P[0].door.value() and
143                P[1].door.value()) != 1):
144             pass
145         for p in P:
146             p.lock.set_value(1)
147         while((P[0].FC.value() or
148                P[1].FC.value()) == 0):
149             valves[0].set_value(1)
150         valves[0].set_value(0)
151 
152         for p in P:
153             if(p.FC.value() == 1):
154                 actual = p.number
155                 destination = actual
156                 break
157     return actual, destination
158 
159 
160 def AL_up(level):
161     while(P[level].FC.value() != 1):
162         motors[0].set_value(1)
163         motors[1].set_value(0)
164     motors[0].set_value(0)
165     return
166 
167 
168 def AL_down(level):
169     while(P[level].FC.value() != 1):
170         valves[0].set_value(1)
171         valves[1].set_value(0)
172     valves[0].set_value(0)
173     return
174 
175 
176 def enable_int_general(P):
177 # Enables button interrupts
178     GPIO.add_event_detect(P[0].button.port,
179                           GPIO.FALLING,
180                           callback=P1_handler,
181                           bouncetime=500)
182     GPIO.add_event_detect(P[1].button.port,
183                           GPIO.FALLING,
184                           callback=P2_handler,
185                           bouncetime=500)
186     GPIO.add_event_detect(P[2].button.port,
187                           GPIO.FALLING,
188                           callback=P3_handler,
189                           bouncetime=500)
190     return
191 
192 def enable_int_0(P):
193 # Enables Autolevel interrupts for floor 0
194     GPIO.add_event_detect(P[0].FCA.port,
195                           GPIO.RISING,
196                           callback=AL0_down,
197                           bouncetime=500)
198     GPIO.add_event_detect(P[0].FCB.port,
199                           GPIO.RISING,
200                           callback=AL0_up,
201                           bouncetime=500)
202     return
203 
204 def enable_int_1(P):
205     # Enables Autolevel interrupts for floor 1
206     GPIO.add_event_detect(P[1].FCA.port,
207                           GPIO.RISING,
208                           callback=AL1_down,
209                           bouncetime=500)
```

```
210     GPIO.add_event_detect(P[1].FCB.port,
211                           GPIO.RISING,
212                           callback=AL1_up,
213                           bouncetime=500)
214     return
215
216 def enable_int_2(P):
217 # Enables Autolevel interrupts for floor 2
218     GPIO.add_event_detect(P[2].FCA.port,
219                           GPIO.RISING,
220                           callback=AL2_down,
221                           bouncetime=500)
222     GPIO.add_event_detect(P[2].FCB.port,
223                           GPIO.RISING,
224                           callback=AL2_up,
225                           bouncetime=500)
226     return
227
228
229 def disable_int(P):
230 # Disalbes interrupts
231     GPIO.remove_event_detect(P[0].button.port)
232     GPIO.remove_event_detect(P[1].button.port)
233     GPIO.remove_event_detect(P[2].button.port)
234
235     GPIO.remove_event_detect(P[0].FCA.port)
236     GPIO.remove_event_detect(P[0].FCB.port)
237
238     GPIO.remove_event_detect(P[1].FCA.port)
239     GPIO.remove_event_detect(P[1].FCB.port)
240
241     GPIO.remove_event_detect(P[2].FCA.port)
242     GPIO.remove_event_detect(P[2].FCB.port)
243     return
244
245 def enable_AL(actual, P):
246 # Enables autolevel interrupts depending on the floor number
247     if(actual == 0):
248         enable_int_0(P)
249     if(actual == 1):
250         enable_int_1(P)
251     if(actual == 2):
252         enable_int_2(P)
253     return
254
255 ##INTERRUPCIONES
256 # Sets "destination" to the number of the floor the elevator must go
257 def P1_handler(port):
258     global destination
259     destination = 0
260     return
261
262
263 def P2_handler(port):
264     global destination
265     destination = 1
266     return
267
268
269 def P3_handler(port):
270     global destination
271     destination = 2
272     return
273
274
275 def AL0_up(port):
276     level = 0
277     AL_up(level)
278     return
279
```

```
280
281 def AL0_down(port):
282     level = 0
283     AL_down(level)
284     return
285
286
287 def AL1_up(port):
288     level = 1
289     AL_up(level)
290     return
291
292
293 def AL1_down(port):
294     level = 1
295     AL_down(level)
296     return
297
298 def AL2_up(port):
299     level = 2
300     AL_up(level)
301     return
302
303
304 def AL2_down(port):
305     level = 2
306     AL_down(level)
307     return
308
309
310 ##MAIN
311 # Explanation: First configuration functions are executed and then
312 # enters the loop. When a button is pressed a interrupt occurs and
313 # it sets "destination" to the number of the floor pressed. If the door
314 # is not closed "destination" is set again to the actual value. If it is,
315 # the door is locked and interrupts are disabled until the movement
316 # is over. The movements is as follows:
317 #   1.-It checks if the destination floor is up or down
318 #   2.-If it is up activates motor1 and motor2 until FCB, then
319 #      deactivates motor1.
320 #   3.-If it is down opens the two valves until FCA, then close one
321 #   4.-When the destination limit switch is pressed the valve is
322 #      closed and the motor is shut down.
323 # Destination door is unlocked, "actual" value is updated and
324 # interrupts are enabled again.
325 # Autolevel: When the elevator is at rest, if one of the actual floors
326 # autolevel limit switch is activated an interupt occurs. Depending
327 # on if it is the upper or lower switch the valve or the motor
328 # is activated
329 global destination
330 [P, motors, valves] = define_IOs()
331 GPIO_config(P, motors, valves)
332 [actual, destination] = initialization(P, motors, valves)
333 enable_int_general(P)
334 enable_AL(actual, P)
335 i = 0
336
337 while(1):
338     try:
339         if (actual != destination and P[actual].door.value() == 1):
340             disable_int(P)
341             P[actual].lock.set_value(1)
342             while(P[destination].FC.value() != 1):
343                 if (actual > destination):
344                     while(P[destination].FCA.value() != 1 and i == 0):
345                         valves[0].set_value(1)
346                         valves[1].set_value(1)
347                     valves[1].set_value(0)
348                     i = 1
349                 elif(actual < destination):
```

```
350                     while(P[destination].FCB.value() != 1 and i == 0):
351                         motors[0].set_value(1)
352                         motors[1].set_value(1)
353                     motors[1].set_value(0)
354                     i = 1
355 
356             valves[0].set_value(0)
357             motors[0].set_value(0)
358             P[destination].lock.set_value(0)
359             actual = destination
360             i = 0
361             enable_int_general(P)
362             enable_AL(actual, P)
363     except:
364         GPIO.cleanup()
365         raise
```

```
001 #3P1S1DHydraulicLocks
002 #
003 #Inputs:    P0 button
004 #           P1 button
005 #           P2 button
006 #           FC floor 0
007 #           FCA floor 0
008 #           tranca FCO floor 0
009 #           tranca FCI floor 0
010 #           FC floor 1
011 #           FCA floor 1
012 #           tranca FCO floor 1
013 #           tranca FCI floor 1
014 #           FC floor 2
015 #           FCA floor 2
016 #           tranca FCO floor 2
017 #           tranca FCI floor 2
018 #           door closed 0
019 #           door closed 1
020 #           door closed 2
021 #
022 #outputs:   motor
023 #           valve
024 #           lock door 0
025 #           lock door 1
026 #           lock door 2
027 #           lock_out floor 0
028 #           lock_out floor 1
029 #           lock_out floor 2
030
031 import RPi.GPIO as GPIO # import GPIO module
032 import time              # import time module
033 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
034
035 ##CLASES
036 class floor:
037 # Class that represents all the sensors and actuators of a floor
038     def __init__(self, number):
039         self.number = number
040         self.FC = sensor("FC" + str(self.number), 0)
041         self.FCA = sensor("FCA" + str(self.number), 0)
042         self.puerta = sensor("puerta" + str(self.number), 0)
043         self.boton = sensor("boton" + str(self.number), 0)
044         self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
045         self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
046         self.lock = actuator("lock" + str(self.number), 0)
047         self.lock_out = actuator("lock_out" + str(self.number), 0)
048
049
050 class sensor:
051 # Class that represents a sensor
052     def __init__(self,name,port):
053         self.name = name
054         self.port = port
055     def value(self):
056         return GPIO.input(self.port)
057
058
059 class actuator:
060 # Class that represents an actuator
061     def __init__(self,name,port):
062         self.name = name
063         self.port = port
064     def set_value(self, valor):
065         if(valor == 1):
066             GPIO.output(self.port, 0)
067         elif(valor == 0):
068             GPIO.output(self.port, 1)
069
```

```
070
071
072 ##FUNCIONES
073 def define_IOs():
074 # Function that configures all the objects of the program and asigns
075 # the GPIO ports to each sensor and actuator
076     p0 = floor(0)
077     p1 = floor(1)
078     p2 = floor(2)
079     P = [p0, p1, p2]
080
081     motor = actuator("motor", 0)
082     valve = actuator("valve", 0)
083
084     P[0].FC.port = 4
085     P[0].puerta.port = 17
086     P[0].boton.port = 27
087     P[0].lock.port = 0
088     P[0].FCA.port = 14
089     P[0].lock_out.port = 0
090     P[0].lock_FCO.port = 18
091     P[0].lock_FCI.port = 23
092
093     P[1].FC.port = 10
094     P[1].puerta.port = 9
095     P[1].boton.port = 11
096     P[1].lock.port = 0
097     P[1].FCA.port = 23
098     P[1].lock_out.port = 0
099     P[1].lock_FCO.port = 7
100     P[1].lock_FCI.port = 12
101
102     P[2].FC.port = 5
103     P[2].puerta.port = 6
104     P[2].boton.port = 13
105     P[2].lock.port = 0
106     P[2].FCA.port = 16
107     P[2].lock_out.port = 0
108     P[2].lock_FCO.port = 21
109     P[2].lock_FCI.port = 26
110     return P, motor, valve
111
112
113 def GPIO_config(P, motor, valve):
114 # Function that sets the GPIOs as inputs or outputs
115 # Its inputs are all the sensors and actuators of the program
116     for p in P:
117         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
118         GPIO.setup(p.puerta.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
119         GPIO.setup(p.boton.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
120         GPIO.setup(p.lock.port,GPIO.OUT)
121         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
122         GPIO.setup(p.lock_out.port,GPIO.OUT)
123         GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
124         GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
125
126     GPIO.setup(motor.port,GPIO.OUT)
127     GPIO.setup(valve.port,GPIO.OUT)
128     return
129
130
131 def initialization(P, motor, valve):
132 # Function that set the initial conditions of the system
133 # Explanation: Sets "actual" to negative value and checks if the
134 # platform is in any of the floors. In that case sets "actual" to
135 # the value of the actual floor. If the platform is not in any floor,
136 # it desdends until it reaches one and sets "actual" and "destination"
137 # to the floor number.
138     actual = -1
139     for p in P:
```

```
140         if  (p.FC.value() == 1):
141             actual = p.number
142             destination = actual
143         else:
144             while(p.puerta.value() != 1):
145                 pass
146             p.lock.set_value(1)
147     if (actual < 0):
148         while((P[0].FCA.value() or
149                P[1].FCA.value()) == 0):
150             valve.set_value(1)
151         valve.set_value(0)
152 
153         for p in P:
154             if(p.FCA.value() == 1):
155                 while(p.lock_FCO.value() != 1):
156                     p.lock_out.set_value(1)
157                 p.lock_out.set_value(0)
158                 while(p.FC.value() != 1):
159                     valve.set_value(1)
160                 valve.set_value(0)
161 
162                 actual = p.number
163                 destination = actual
164                 p.lock.set_value(0)
165     return actual, destination
166 
167 
168 def enable_int(P):
169 # Enables interrupts
170     GPIO.add_event_detect(P[0].boton.port,
171                           GPIO.FALLING,
172                           callback=P1_handler,
173                           bouncetime=500)
174     GPIO.add_event_detect(P[1].boton.port,
175                           GPIO.FALLING,
176                           callback=P2_handler,
177                           bouncetime=500)
178     GPIO.add_event_detect(P[2].boton.port,
179                           GPIO.FALLING,
180                           callback=P3_handler,
181                           bouncetime=500)
182     return
183 
184 
185 def disable_int(P):
186 # Disables interrupts
187     GPIO.remove_event_detect(P[0].boton.port)
188     GPIO.remove_event_detect(P[1].boton.port)
189     GPIO.remove_event_detect(P[2].boton.port)
190     return
191 
192 
193 ##INTERRUPCIONES
194 # Sets "destination" to the number of the floor the elevator must go
195 def P1_handler(port):
196     global destination
197     destination = 0
198     return
199 
200 
201 def P2_handler(port):
202     global destination
203     destination = 1
204     return
205 
206 
207 def P3_handler(port):
208     global destination
209     destination = 2
```

```
210     return
211
212 ##MAIN
213 # Explanation: First configuration functions are executed and then
214 # enters the loop. When a button is pressed a interrupt occurs and
215 # it sets "destination" to the number of the floor pressed. If the door
216 # is not closed "destination" is set again to the actual value. If it is,
217 # the door is locked and interrupts are disabled until the movement
218 # is over. If the destination is down, the movement is as follows:
219 #   1.-Platform elevates untill FCA
220 #   2.-Hydraulic lock is retracted
221 #   3.-Destination hidraulic lock is extracted and discharge valve open
222 #   4.-When the platform arrives to the destination the valve is closed
223 # If the destination is up, the movement is as follows:
224 #   1.-Platform elevates and Hydraulic lock is retracted
225 #   2.-Platform stops when it arrives to destinations FCA
226 #   3.-Destination hidraulic lock is extracted and discharge valve open
227 #   4.-When the platform arrives to the destination the valve is closed
228 # Destination door is unlocked, "actual" value is
229 # updated and interrupts are enabled again.
230 global destination
231 [P, motor, valve] = define_IOs()
232 GPIO_config(P, motor, valve)
233 [actual, destination] = initialization(P, motor, valve)
234 enable_int(P)
235
236 while(1):
237     try:
238         if (actual != destination and P[actual].puerta.value() == 1):
239             disable_int(P)
240             P[actual].lock.set_value(1)
241             if (actual > destination):
242                 while(P[actual].FCA.value() != 1):
243                     motor.set_value(1)
244                 motor.set_value(0)
245                 while(P[actual].lock_FCI.value() != 1):
246                     P[actual].lock_out.set_value(0)
247                     P[destination].lock_out.set_value(1)
248                 valve.set_value(1)
249                 while(P[destination].lock_FCO.value() != 1):
250                     if(P[destination].FCA.value() == 1):
251                         valve.set_value(0)
252                 while(P[destination].FC.value() != 1):
253                     valve.set_value(1)
254                 valve.set_value(0)
255             elif(actual < destination):
256                 while(P[destination].lock_FCI.value() != 1):
257                     P[destination].lock_out.set_value(0)
258                 while(P[destination].FCA.value() != 1):
259                     motor.set_value(1)
260                 while(P[destination].lock_FCO.value() != 1):
261                     motor.set_value(0)
262                     P[actual].lock_out.set_value(0)
263                     P[destination].lock_out.set_value(1)
264                 while(P[destination].FC.value() != 1):
265                     valve.set_value(1)
266
267             valve.set_value(0)
268             motor.set_value(0)
269             P[destination].lock.set_value(0)
270             actual = destination
271             enable_int(P)
272     except:
273         GPIO.cleanup()
274         raise
```

```
001 #3P1S2DHydraulicLocks
002 #
003 #Inputs:     P0 button
004 #            P1 button
005 #            P2 button
006 #            FC floor 0
007 #            FCA floor 0
008 #            tranca FCO floor 0
009 #            tranca FCI floor 0
010 #            FC floor 1
011 #            FCA floor 1
012 #            tranca FCO floor 1
013 #            tranca FCI floor 1
014 #            FC floor 2
015 #            FCA floor 2
016 #            tranca FCO floor 2
017 #            tranca FCI floor 2
018 #            door closed 0
019 #            door closed 1
020 #            door closed 2
021 #
022 #outputs:    motor
023 #            valve 1
024 #            valve 2
025 #            lock door 0
026 #            lock door 1
027 #            lock door 2
028 #            lock_out floor 0
029 #            lock_out floor 1
030 #            lock_out floor 2
031
032 import RPi.GPIO as GPIO # import GPIO module
033 import time              # import time module
034 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
035
036 ##CLASES
037 class floor:
038 # Class that represents all the sensors and actuators of a floor
039     def __init__(self, number):
040         self.number = number
041         self.FC = sensor("FC" + str(self.number), 0)
042         self.FCA = sensor("FCA" + str(self.number), 0)
043         self.door = sensor("door" + str(self.number), 0)
044         self.button = sensor("button" + str(self.number), 0)
045         self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
046         self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
047         self.lock = actuator("lock" + str(self.number), 0)
048         self.lock_out = actuator("lock_out" + str(self.number), 0)
049
050
051 class sensor:
052 # Class that represents a sensor
053     def __init__(self,name,port):
054         self.name = name
055         self.port = port
056     def value(self):
057         return GPIO.input(self.port)
058
059 class actuator:
060 # Class that represents an actuator
061     def __init__(self,name,port):
062         self.name = name
063         self.port = port
064     def set_value(self, valor):
065         if(valor == 1):
066             GPIO.output(self.port, 0)
067         elif(valor == 0):
068             GPIO.output(self.port, 1)
069
```

```
070
071 ##FUNCIONES
072 def define_IOs():
073 # Function that configures all the objects of the program and asigns
074 # the GPIO ports to each sensor and actuator
075     p0 = floor(0)
076     p1 = floor(1)
077     p2 = floor(2)
078     P = [p0, p1, p2]
079
080     motor = actuator("motor", 0)
081     valve1 = actuator("valve1", 0)
082     valve2 = actuator("valve2", 0)
083     valves = [valve1, valve2]
084
085     P[0].FC.port = 4
086     P[0].door.port = 17
087     P[0].button.port = 27
088     P[0].lock.port = 0
089     P[0].FCA.port = 14
090     P[0].lock_out.port = 0
091     P[0].lock_FCO.port = 18
092     P[0].lock_FCI.port = 23
093
094     P[1].FC.port = 10
095     P[1].door.port = 9
096     P[1].button.port = 11
097     P[1].lock.port = 0
098     P[1].FCA.port = 23
099     P[1].lock_out.port = 0
100     P[1].lock_FCO.port = 7
101     P[1].lock_FCI.port = 12
102
103     P[2].FC.port = 5
104     P[2].door.port = 6
105     P[2].button.port = 13
106     P[2].lock.port = 0
107     P[2].FCA.port = 16
108     P[2].lock_out.port = 0
109     P[2].lock_FCO.port = 21
110     P[2].lock_FCI.port = 26
111     return P, motor, valves
112
113 def GPIO_config(P, motor, valves):
114 # Function that sets the GPIOs as inputs or outputs
115 # Its inputs are all the sensors and actuators of the program
116     for p in P:
117         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
# configuracion GPIOs
118         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
119         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
120         GPIO.setup(p.lock.port,GPIO.OUT)
121         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
122         GPIO.setup(p.lock_out.port,GPIO.OUT)
123         GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
124         GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
125
126     GPIO.setup(motor.port,GPIO.OUT)
127     GPIO.setup(valve1.port,GPIO.OUT)
128     GPIO.setup(valve2.port,GPIO.OUT)
129     return
130
131 def initialization(P, motor, valves):
132 # Function that set the initial conditions of the system
133 # Explanation: Sets "actual" to negative value and checks if the
134 # platform is in any of the floors. In that case sets "actual" to
135 # the value of the actual floor. If the platform is not in any floor,
136 # it desdends until it reaches one and sets "actual" and "destination"
137 # to the floor number.
138     actual = -1
```

```
    for p in P:
        if  (p.FC.value() == 1):
            actual = p.number
            destination = actual
        else:
            while(p.door.value() != 1):
                pass
            p.lock.set_value(1)
    if (actual < 0):
        while((P[0].FCA.value() or
               P[1].FCA.value()) == 0):
            valves[0].set_value(1)
        valve.set_value(0)

        for p in P:
            if(p.FCA.value() == 1):
                while(p.lock_FCO.value() != 1):
                    p.lock_out.set_value(1)
                p.lock_out.set_value(0)
                while(p.FC.value() != 1):
                    valves[0].set_value(1)
                valves[0].set_value(0)

                actual = p.number
                destination = actual
                p.lock.set_value(0)
    return actual, destination

def enable_int(P):
# Enables interrupts
    GPIO.add_event_detect(P[0].button.port,
                          GPIO.FALLING,
                          callback=P1_handler,
                          bouncetime=500)
    GPIO.add_event_detect(P[1].button.port,
                          GPIO.FALLING,
                          callback=P2_handler,
                          bouncetime=500)
    GPIO.add_event_detect(P[2].button.port,
                          GPIO.FALLING,
                          callback=P3_handler,
                          bouncetime=500)
    return

def disable_int(P):
# Disables interrupts
    GPIO.remove_event_detect(P[0].button.port)
    GPIO.remove_event_detect(P[1].button.port)
    GPIO.remove_event_detect(P[2].button.port)
    return

##INTERRUPCIONES
# Sets "destination" to the number of the floor the elevator must go
def P1_handler(port):
    global destination
    destination = 0
    return

def P2_handler(port):
    global destination
    destination = 1
    return

def P3_handler(port):
    global destination
    destination = 2
    return

##MAIN
# Explanation: First configuration functions are executed and then
```

```
# enters the loop. When a button is pressed a interrupt occurs and
# it sets "destination" to the number of the floor pressed. If the door
# is not closed "destination" is set again to the actual value. If it is,
# the door is locked and interrupts are disabled until the movement
# is over. If the destination is down, the movement is as follows:
#   1.-Platform elevates untill FCA
#   2.-Hydraulic lock is retracted
#   3.-Destination hidraulic lock is extracted and activate vale1 and
#      valve2 until FCA, then deactivate valve2.
#   4.-When the platform arrives to the destination valve1 is closed
# If the destination is up, the movement is as follows:
#   1.-Platform elevates and Hydraulic lock is retracted
#   2.-Platform stops when it arrives to destinations FCA
#   3.-Destination hidraulic lock is extracted and discharge valve open
#   4.-When the platform arrives to the destination the valve is closed
# Destination door is unlocked, "actual" value is
# updated and interrupts are enabled again.
global destination
[P, motor, valves] = define_IOs()
GPIO_config(P, motor, valves)
[actual, destination] = initialization(P, motor, valves)
enable_int(P)

while(1):
    try:
        if (actual != destination and P[actual].door.value() == 1):
            disable_int(P)
            P[actual].lock.set_value(1)
            if (actual > destination):
                while(P[actual].FCA.value() != 1):
                    motor.set_value(1)
                motor.set_value(0)
                while(P[actual].lock_FCI.value() != 1):
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                valves[0].set_value(1)
                valves[1].set_value(1)
                while(P[destination].lock_FCO.value() != 1):
                    if(P[destination].FCA.value() == 1):
                        valves[0].set_value(0)
                        valves[1].set_value(0)
                while(P[destination].FCA.value() != 1):
                    pass
                valves[1].set_value(0)
                while(P[destination].FC.value() != 1):
                    pass
            elif(actual < destination):
                while(P[destination].lock_FCI.value() != 1):
                    P[destination].lock_out.set_value(0)
                while(P[destination].FCA.value() != 1):
                    motor.set_value(1)
                while(P[destination].lock_FCO.value() != 1):
                    motor.set_value(0)
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                while(P[destination].FC.value() != 1):
                    valves[0].set_value(1)

            valves[0].set_value(0)
            valves[1].set_value(0)
            motor.set_value(0)
            P[destination].lock.set_value(0)
            actual = destination
            enable_int(P)
    except:
        GPIO.cleanup()
        raise
```

```
001 #3P2S2DHydraulicLocks
002 #
003 #Inputs:    P0 button
004 #           P1 button
005 #           P2 button
006 #           FC floor 0
007 #           FCA floor 0
008 #           tranca FCO floor 0
009 #           tranca FCI floor 0
010 #           FC floor 1
011 #           FCA floor 1
012 #           tranca FCO floor 1
013 #           tranca FCI floor 1
014 #           FC floor 2
015 #           FCA floor 2
016 #           tranca FCO floor 2
017 #           tranca FCI floor 2
018 #           door closed 0
019 #           door closed 1
020 #           door closed 2
021 #
022 #outputs:   motor 1
023 #           motor 2
024 #           valve 1
025 #           valve 2
026 #           lock door 0
027 #           lock door 1
028 #           lock door 2
029 #           lock_out floor 0
030 #           lock_out floor 1
031 #           lock_out floor 2
032
033 import RPi.GPIO as GPIO # import GPIO module
034 import time              # import time module
035 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
036
037 ##CLASES
038 class floor:
039 # Class that represents all the sensors and actuators of a floor
040     def __init__(self, number):
041         self.number = number
042         self.FC = sensor("FC" + str(self.number), 0)
043         self.FCA = sensor("FCA" + str(self.number), 0)
044         self.door = sensor("door" + str(self.number), 0)
045         self.button = sensor("button" + str(self.number), 0)
046         self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
047         self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
048         self.lock = actuator("lock" + str(self.number), 0)
049         self.lock_out = actuator("lock_out" + str(self.number), 0)
050
051
052 class sensor:
053 # Class that represents a sensor
054     def __init__(self,name,port):
055         self.name = name
056         self.port = port
057     def value(self):
058         return GPIO.input(self.port)
059
060
061 class actuator:
062 # Class that represents an actuator
063     def __init__(self,name,port):
064         self.name = name
065         self.port = port
066     def set_value(self, valor):
067         if(valor == 1):
068             GPIO.output(self.port, 0)
069         elif(valor == 0):
```

```
070             GPIO.output(self.port, 1)
071
072
073 ##FUNCIONES
074 def define_IOs():
075 # Function that configures all the objects of the program and asigns
076 # the GPIO ports to each sensor and actuator
077     p0 = floor(0)
078     p1 = floor(1)
079     p2 = floor(2)
080     P = [p0, p1, p2]
081
082     motor1 = actuator("motor1", 3)
083     motor2 = actuator("motor2", 5)
084     valve1 = actuator("valve1", 37)
085     valve2 = actuator("valve2", 40)
086     valves = [valve1, valve2]
087     motors = [motor1, motor2]
088
089     P[0].FC.port = 7
090     P[0].door.port = 11
091     P[0].button.port = 13
092     P[0].lock.port = 15
093     P[0].FCA.port = 8
094     P[0].lock_out.port = 10
095     P[0].lock_FCO.port = 12
096     P[0].lock_FCI.port = 16
097
098     P[1].FC.port = 19
099     P[1].door.port = 21
100     P[1].button.port = 23
101     P[1].lock.port = 27
102     P[1].FCA.port = 18
103     P[1].lock_out.port = 22
104     P[1].lock_FCO.port = 24
105     P[1].lock_FCI.port = 26
106
107     P[2].FC.port = 29
108     P[2].door.port = 31
109     P[2].button.port = 33
110     P[2].lock.port = 35
111     P[2].FCA.port = 28
112     P[2].lock_out.port = 32
113     P[2].lock_FCO.port = 36
114     P[2].lock_FCI.port = 38
115     return P, motors, valves
116
117
118 def GPIO_config(P, motors, valves):
119 # Function that sets the GPIOs as inputs or outputs
120 # Its inputs are all the sensors and actuators of the program
121     for p in P:
122         GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
123         GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
124         GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
125         GPIO.setup(p.lock.port,GPIO.OUT)
126         GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
127         GPIO.setup(p.lock_out.port,GPIO.OUT)
128         GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
129         GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
130
131     GPIO.setup(motor1.port,GPIO.OUT)
132     GPIO.setup(motor2.port,GPIO.OUT)
133     GPIO.setup(valve1.port,GPIO.OUT)
134     GPIO.setup(valve2.port,GPIO.OUT)
135     return
136
137
138 def initialization(P, motors, valves):
139 # Function that set the initial conditions of the system
```

```
# Explanation: Sets "actual" to negative value and checks if the
# platform is in any of the floors. In that case sets "actual" to
# the value of the actual floor. If the platform is not in any floor,
# it desdends until it reaches one and sets "actual" and "destination"
# to the floor number.
    actual = -1
    for p in P:
        if  (p.FC.value() == 1):
            actual = p.number
            destination = actual
        else:
            while(p.door.value() != 1):
                pass
            p.lock.set_value(1)
    if (actual < 0):
        while((P[0].FCA.value() or
               P[1].FCA.value()) == 0):
            valves[0].set_value(1)
        valves[0].set_value(0)

        for p in P:
            if(p.FCA.value() == 1):
                while(p.lock_FCO.value() != 1):
                    p.lock_out.set_value(1)
                p.lock_out.set_value(0)
                while(p.FC.value() != 1):
                    valves[0].set_value(1)
                valves[0].set_value(0)

                actual = p.number
                destination = actual
                p.lock.set_value(0)
    return actual, destination


def enable_int(P):
# Enables interrupts
    GPIO.add_event_detect(P[0].button.port,
                          GPIO.FALLING,
                          callback=P1_handler,
                          bouncetime=500)
    GPIO.add_event_detect(P[1].button.port,
                          GPIO.FALLING,
                          callback=P2_handler,
                          bouncetime=500)
    GPIO.add_event_detect(P[2].button.port,
                          GPIO.FALLING,
                          callback=P3_handler,
                          bouncetime=500)
    return


def disable_int(P):
# Disables interrupts
    GPIO.remove_event_detect(P[0].button.port)
    GPIO.remove_event_detect(P[1].button.port)
    GPIO.remove_event_detect(P[2].button.port)
    return


##INTERRUPCIONES
# Sets "destination" to the number of the floor the elevator must go
def P1_handler(port):
    global destination
    destination = 0
    return


def P2_handler(port):
    global destination
```

```
    destination = 1
    return


def P3_handler(port):
    global destination
    destination = 2
    return


##MAIN
# Explanation: First configuration functions are executed and then
# enters the loop. When a button is pressed a interrupt occurs and
# it sets "destination" to the number of the floor pressed. If the door
# is not closed "destination" is set again to the actual value. If it is,
# the door is locked and interrupts are disabled until the movement
# is over. If the destination is down, the movement is as follows:
#   1.-Platform elevates untill FCA
#   2.-Hydraulic lock is retracted
#   3.-Destination hidraulic lock is extracted and activate vale1 and
#      valve2 until FCA, then deactivate valve2.
#   4.-When the platform arrives to the destination valve1 is closed
# If the destination is up, the movement is as follows:
#   1.-motor1 and motor2 are activated until destination FC,
#      then motor2 is deactivated
#   2.-Hydraulic lock is retracted
#   3.-motor1 is deactivated when it arrives to destinations FCA
#   4.-Destination hidraulic lock is extracted and valve1 open
#   5.-When the platform arrives to the destination valve1 is closed
# Destination door is unlocked, "actual" value is
# updated and interrupts are enabled again.
global destination
[P, motors, valves] = define_IOs()
GPIO_config(P, motors, valves)
[actual, destination] = initialization(P, motors, valves)
enable_int(P)

while(1):
    try:
        if (actual != destination and P[actual].door.value() == 1):
            disable_int(P)
            P[actual].lock.set_value(1)
            if (actual > destination):
                while(P[actual].FCA.value() != 1):
                    motors[0].set_value(1)
                motors[0].set_value(0)
                while(P[actual].lock_FCI.value() != 1):
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                valves[0].set_value(1)
                valves[1].set_value(1)
                while(P[destination].lock_FCO.value() != 1):
                    if(P[destination].FCA.value() == 1):
                        valves[0].set_value(0)
                        valves[1].set_value(0)
                while(P[destination].FCA.value() != 1):
                    pass
                valves[1].set_value(0)
                while(P[destination].FC.value() != 1):
                    pass
            elif(actual < destination):
                while(P[destination].lock_FCI.value() != 1):
                    P[destination].lock_out.set_value(0)
                while(P[destination].FC.value() != 1):
                    motors[0].set_value(1)
                    motors[1].set_value(1)
                motors[1].set_value(0)
                while(P[destination].FCA.value() != 1):
                    pass
                motors[0].set_value(0)
```

```
280                 while(P[destination].lock_FCO.value() != 1):
281                     motors[0].set_value(0)
282                     P[actual].lock_out.set_value(0)
283                     P[destination].lock_out.set_value(1)
284                 while(P[destination].FC.value() != 1):
285                     valves[0].set_value(1)
286 
287             valves[0].set_value(0)
288             valves[1].set_value(0)
289             motors[0].set_value(0)
290             motors[1].set_value(0)
291             P[destination].lock.set_value(0)
292             actual = destination
293             enable_int(P)
294     except:
295         GPIO.cleanup()
296         raise
```