Electronic Redesign of an Industrial Lift

URTZI AGIRRE ALBIZU
Abril de 2016
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 Rüben 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

Acknowledgements................................................................................................................................. 3  
Abstract .................................................................................................................................................... 4  
List of Contents ........................................................................................................................................ 5  
List of Figures ........................................................................................................................................... 7  
List of Tables............................................................................................................................................ 10  
List of Symbols and Abbreviations ........................................................................................................ 11  
1. Introduction......................................................................................................................................... 13  
   1.1. Company characterization ............................................................................................................... 14  
   1.2. The Problem ................................................................................................................................... 15  
       1.2.1. Company needs ....................................................................................................................... 16  
       1.2.2. Objectives .............................................................................................................................. 16  
   1.3. Methods ...................................................................................................................................... 16  
2. Background......................................................................................................................................... 18  
   2.1. System Description ......................................................................................................................... 19  
       2.1.1. Components ............................................................................................................................ 19  
       2.1.2. Auto-levelling .......................................................................................................................... 24  
       2.1.3. Hydraulic locks .......................................................................................................................... 24  
       2.1.4. Two speeds ............................................................................................................................. 25  
   2.2. Automation technologies ................................................................................................................. 26  
       2.2.1. PLC ......................................................................................................................................... 26  
       2.2.2. Microcontroller ....................................................................................................................... 27  
   2.3. State of the art ................................................................................................................................. 29  
3. Development....................................................................................................................................... 30  
   3.1. Problem analysis ............................................................................................................................. 31  
   3.2. Brainstorming & Preliminary drafts ............................................................................................... 32  
       3.2.1. Input interface ............................................................................................................................ 32  
       3.2.2. Output interface ....................................................................................................................... 33  
       3.2.3. Technologies ............................................................................................................................ 35  
   3.3. Selecting the best idea ..................................................................................................................... 38  
       3.3.1. Technology ............................................................................................................................... 38
3.3.2. Input interface ........................................................................................................ 39
3.3.3. Output interface .................................................................................................... 39

3.4. Developing the main idea ......................................................................................... 40
  3.4.1. Software .................................................................................................................. 40
  3.4.2. Hardware ................................................................................................................ 63
  3.4.3. Testing ..................................................................................................................... 65
  3.4.4. Safety ..................................................................................................................... 70

3.5. Budgeting .................................................................................................................... 70

3.6. Critical analysis and prospects for improvement .................................................... 71

3.7. Equipment instructions manual ............................................................................... 72
  3.7.1. Raspberry Pi raw configuration ............................................................................ 72
  3.7.2. Establish working program .................................................................................. 77
  3.7.3. Pin configuration .................................................................................................... 79

3.8. Maintenance Guidelines ........................................................................................... 82
  3.8.1. Assembly ............................................................................................................... 82
  3.8.2. Maintenance ......................................................................................................... 82

4. Concluding remarks ..................................................................................................... 84

5. Sources of information ............................................................................................... 86
  5.1. Books and scientific papers .................................................................................... 87
  5.2. Websites ................................................................................................................... 88

6. Appendixes .................................................................................................................. 89
List of Figures

Figure 1: ABER logo ............................................................................................................. 14
Figure 2: Elevator scheme ................................................................................................. 15
Figure 3: Gantt diagram ................................................................................................... 17
Figure 4: Hydraulic circuit ................................................................................................ 19
Figure 5: Discharge valve (www.contranspower.com) ...................................................... 19
Figure 6: Pressure limit valve (www.austworld.com) ......................................................... 20
Figure 7: Hydraulic cylinder (www.pedroroquet.com) ....................................................... 20
Figure 8: Pump (img.directindustry.es) .......................................................................... 20
Figure 9: Check valve (www.allprosperity.com) ................................................................. 21
Figure 10: Tank (aber.es) ................................................................................................... 21
Figure 11: Electric circuit ................................................................................................... 21
Figure 12: Limit switch (www.tracepartsonline.net) ......................................................... 22
Figure 13: Electromagnetic lock (products.dorma.com) .................................................... 22
Figure 14: Motor (roydisa.es) .......................................................................................... 22
Figure 15: Contactor (kiirux.com) ................................................................................... 23
Figure 16: Buttons (static.grainger.com) ........................................................................ 23
Figure 17: Transformer (www.cetronic.es) ........................................................................ 23
Figure 18: Auto-level sketch .......................................................................................... 24
Figure 19: Hydraulic locks sketch .................................................................................... 24
Figure 20: Two speeds scheme ....................................................................................... 25
Figure 21: PLC hardware .................................................................................................. 26
Figure 22: Ladder language example ................................................................................ 27
Figure 23: Microcontroller hardware ............................................................................... 28
Figure 24: Voltage divider ............................................................................................... 32
Figure 25: Optocoupler input .......................................................................................... 33
Figure 26: Transistor output ............................................................................................ 34
Figure 27: Optocoupler plus transistor output ................................................................. 34
Figure 28: Relay output .................................................................................................... 35
Figure 29: Raspberry Pi ..................................................................................................... 36
Figure 30: Class sensor example ..................................................................................... 40
Figure 31: Class actuator example .................................................................................. 41
Figure 32: Class floor example .............................................................................. 41
Figure 33: define_IOS() example ......................................................................... 42
Figure 34: GPIO_config() example ...................................................................... 42
Figure 35: Standard elevators fluxogram .............................................................. 44
Figure 36: Initialization of standard elevator ......................................................... 46
Figure 37: Button interrupt handler ...................................................................... 46
Figure 38: enable_int() function ......................................................................... 47
Figure 39: Standard elevator’s main program ....................................................... 48
Figure 40: Auto-level interrupt fluxogram ............................................................ 49
Figure 41: Auto-levelling interrupt handler ......................................................... 52
Figure 42: Button interrupt enabling function ...................................................... 52
Figure 43: Auto-levelling interrupts enabling functions ...................................... 52
Figure 44: Auto-levelling interrupts disabling functions ..................................... 53
Figure 45: Code changes for two speeds .............................................................. 54
Figure 46: Hydraulic lock elevator’s fluxogram .................................................... 55
Figure 47: Hydraulic lock elevator’s initialization ................................................ 58
Figure 48: Hydraulic lock elevator’s main program ............................................. 60
Figure 49: Changes in code for two speeds .......................................................... 62
Figure 50: Proposed input interface .................................................................... 63
Figure 51: Selected output interface .................................................................... 64
Figure 52: Jumper wires ....................................................................................... 64
Figure 53: Test layout ........................................................................................... 65
Figure 54: Testing screen ...................................................................................... 66
Figure 55: Hydraulic scheme and actual assembly .............................................. 67
Figure 56: System connection ............................................................................. 67
Figure 57: Compute module ................................................................................ 71
Figure 58: OS download ....................................................................................... 72
Figure 59: Win32 Disk Imager download ............................................................... 72
Figure 60: Win32 Disk Imager ............................................................................. 73
Figure 61: ipconfig ............................................................................................... 73
Figure 62: cmdline.txt ......................................................................................... 73
Figure 63: Network and Sharing Centre .............................................................. 74
Figure 64: Network properties ............................................................................ 74
Figure 65: Putty download ................................................................................... 74
Figure 66: Putty .................................................................................................... 75
Figure 67: RPi login........................................................................................................... 75
Figure 68: raspi-config....................................................................................................... 75
Figure 69: Remote desktop............................................................................................... 76
Figure 70: smb.conf modifications.................................................................................... 76
Figure 71: Win32 Disk imager download .......................................................................... 77
Figure 72: Image writing .................................................................................................... 77
Figure 73: program_autorun.py....................................................................................... 77
Figure 74: Modifiable path............................................................................................... 78
Figure 75: Enclosure......................................................................................................... 82
Figure 76: Conformal coating........................................................................................... 82
Figure 77: count.py script............................................................................................... 83
List of Tables

Table 1: Standard elevators IO number ............................................................................................................. 31
Table 2: Auto-level elevator IO number ............................................................................................................. 31
Table 3: Hydraulic locks elevator IO number .................................................................................................. 31
Table 4: Raspberry Pi characteristics ............................................................................................................. 37
Table 5: Technologies comparison summary ................................................................................................. 39
Table 6: Standard elevators inputs/outputs .................................................................................................. 45
Table 7: Auto-leveling elevator’s inputs/outputs ........................................................................................... 51
Table 8: Added outputs for two speeds .......................................................................................................... 53
Table 9: Hydraulic lock elevator’s inputs/outputs .......................................................................................... 57
Table 10: Added outputs for two speeds ....................................................................................................... 61
Table 11: Budget .............................................................................................................................................. 70
Table 12: SWOT analysis ............................................................................................................................... 71
List of Symbols and Abbreviations

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

![ABER logo](image)
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.

![Elevator scheme](attachment://elevator_scheme.png)

*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 through 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

![Gantt diagram]

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.

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

![Pressure limit valve](www.austworld.com)

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.

![Hydraulic cylinder](www.pedro-roquet.com)

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.

![Pump](img.directindustry.es)

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.

![Limit switch](www.tracepartsonline.net)

**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.

![Electromagnetic lock](products.dorma.com)

**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.

![Motor](roydisa.es)

**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.

![Contactor](kiirux.com)

Figure 15: Contactor (kiirux.com)

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

![Buttons](static.grainger.com)

Figure 16: Buttons (static.grainger.com)

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

![Transformer](www.cetronic.es)

Figure 17: Transformer (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.

![Auto-level sketch](image1.png)

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.

![Hydraulic locks sketch](image2.png)

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.

![Two speeds scheme](image)

**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

- 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.

![Ladder language example](image)

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

- 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 sets 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.

Figure 23: Microcontroller hardware
• 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 programmed 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**

- Problem Analysis
- Brainstorming and Preliminary drafts
- Selecting the best idea
- Developing the main idea
- Budgeting
- Critical Analysis and Prospects of Development
- Equipment Instruction Manual
- Maintenance Guidelines
3. Development

3.1. Problem analysis

Considering the company needs, the objectives and the system itself, the following points have 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. Below 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 elevator IO number**

<table>
<thead>
<tr>
<th>2 floors</th>
<th>1U 1D</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 floors</th>
<th>1U 1D</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>5</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

**Table 2: Auto-level elevator IO number**

<table>
<thead>
<tr>
<th>2 floors</th>
<th>1U 1D</th>
<th>1U 2D</th>
<th>2U 2D</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 floors</th>
<th>1U 1D</th>
<th>1U 2D</th>
<th>2U 2D</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>22</td>
<td>22</td>
</tr>
</tbody>
</table>

**Table 3: Hydraulic locks elevator IO number**

<table>
<thead>
<tr>
<th>2 floors</th>
<th>1U 1D</th>
<th>1U 2D</th>
<th>2U 2D</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 floors</th>
<th>1U 1D</th>
<th>1U 2D</th>
<th>2U 2D</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>8</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

- Flexibility: As the controller can be installed in different elevators and more innovations are planned 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](image)

- SW: The sensor
- R1 & R2: Resistors that divide the 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.

![Optocoupler input diagram](image)

- **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](image)

- 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](image)

- 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](image)

- 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 below.

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:

![Raspberry Pi](image)

**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>
<thead>
<tr>
<th>Table 4: Raspberry Pi characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Microcontroller</td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>On Board Network</td>
</tr>
<tr>
<td>Multitasking</td>
</tr>
<tr>
<td>Input voltage</td>
</tr>
<tr>
<td>Flash Memory</td>
</tr>
<tr>
<td>USB ports</td>
</tr>
<tr>
<td>Operative system</td>
</tr>
<tr>
<td>Integrated developing system (IDE)</td>
</tr>
</tbody>
</table>

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 programming knowledge. Hardware modifications needs new design of the PCB.
- PLC: The software can be changed and ladder programming 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>
<thead>
<tr>
<th></th>
<th>1º</th>
<th>2º</th>
<th>3º</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>PLC</td>
<td>Raspberry Pi</td>
<td>PCB</td>
</tr>
<tr>
<td>Innovation capabilities</td>
<td>Raspberry Pi</td>
<td>PCB</td>
<td>PLC</td>
</tr>
<tr>
<td>Price</td>
<td>PCB</td>
<td>Raspberry Pi</td>
<td>PLC</td>
</tr>
</tbody>
</table>

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.

```python
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)
```

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.

```python
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)
```

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.

```python
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)
```

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.

  o Outputs:
    ▪ P: Array that groups all floor objects
    ▪ motor: Actuator type object that represents the motor
    ▪ valve: Actuator type object that represents the valve
def define_IOs():
    # Function that configures all the objects of the program and assigns
    # 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)

    P[0].FC.port = 7
    P[0].door.port = 11
    P[0].button.port = 13
    P[0].lock.port = 15

    P[1].FC.port = 19
    P[1].door.port = 21
    P[1].button.port = 23
    P[1].lock.port = 29

    P[2].FC.port = 31
    P[2].door.port = 33
    P[2].button.port = 32
    P[2].lock.port = 35
    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.

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

def GPIO_config(P, motor, valve):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.door.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.button.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port, GPIO.OUT)

    GPIO.setup(motor.port, GPIO.OUT)
    GPIO.setup(valve.port, GPIO.OUT)
    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.
  o Inputs:
    ▪ P: Array that groups all floor objects
    ▪ motor: Object that represents the motor
    ▪ valve: Object that represents the valve
  o 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.

![Fluxogram of a standard elevator](image)

**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
o **p1.door**: Sensor type object that represents the limit switch of the door
  o **p1.button**: Sensor type object that represents the button of the floor
  o **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.
  o **p2.number**: Integer type variable that stores the floor number
  o **p2.FC**: Sensor type object that represents the limit switch of the floor
  o **p2.door**: Sensor type object that represents the limit switch of the door
  o **p2.button**: Sensor type object that represents the button of the floor
  o **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

<table>
<thead>
<tr>
<th>Input</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0 button</td>
<td>13</td>
</tr>
<tr>
<td>FC floor 0</td>
<td>7</td>
</tr>
<tr>
<td>Door closed 0</td>
<td>11</td>
</tr>
<tr>
<td>P1 button</td>
<td>23</td>
</tr>
<tr>
<td>FC floor 1</td>
<td>19</td>
</tr>
<tr>
<td>Door closed 1</td>
<td>21</td>
</tr>
<tr>
<td>P2 button</td>
<td>32</td>
</tr>
<tr>
<td>FC floor 2</td>
<td>31</td>
</tr>
<tr>
<td>Door closed 2</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor</td>
<td>37</td>
</tr>
<tr>
<td>Valve</td>
<td>38</td>
</tr>
<tr>
<td>Lock door 0</td>
<td>15</td>
</tr>
<tr>
<td>Lock door 1</td>
<td>29</td>
</tr>
<tr>
<td>Lock door 2</td>
<td>35</td>
</tr>
</tbody>
</table>

### 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 "destination" 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 "destination" to the floor number.
def initialization(P, motor, valve):
    # 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 descends 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
            if (actual < 0):
                while((P[0].door.value() and
                       P[1].door.value()) != 1):
                    pass
                for p in P:
                    p.lock.set_value(1)
            elif((P[0].FC.value() or
                  P[1].FC.value()) == 0):
                valve.set_value(1)
                valve.set_value(0)
            for p in P:
                if(p.FC.value() == 1):
                    actual = p.number
                    destination = actual
                    p.lock.set_value(0)
                break
            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:

def P1_handler(port):
    global destination
    destination = 0
    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.

```python
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 def disable_int(P):
155     # Disables interrupts
156     GPIO.remove_event_detect(P[0].button.port)
157     GPIO.remove_event_detect(P[1].button.port)
158     GPIO.remove_event_detect(P[2].button.port)
159     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.
## 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 opens the valve
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 global destination = define_IOs()

GPIO_config(P, motor, valve)

[actual, destination] = initialization(P, motor, valve)

enable_int(P)

while(1):

try:
    time.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):
                valve.set_value(1)
                motor.set_value(1)
            elif(actual < destination):
                motor.set_value(1)
                valve.set_value(0)
            else:
                actual = destination
        P[destination].lock.set_value(0)
        actual = destination
        enable_int(P)
    else:
        destination = actual
except:
    GPIO.cleanup()
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.

Variables

- \(P\): array that stores all floor objects
- \(p0\): floor type object that represents the first floor of the elevator
- \(p0.\text{number}\): Integer type variable that stores the floor number
  - \(p0.\text{FC}\): Sensor type object that represent the limit switch of the floor
  - \(p0.\text{FCA}\): Sensor type object that represent the upper limit switch for auto-levelling.
  - \(p0.\text{FCB}\): Sensor type object that represent the lower limit switch for auto-levelling.
  - \(p0.\text{door}\): Sensor type object that represents the limit switch of the door
  - \(p0.\text{button}\): Sensor type object that represents the button of the floor
o **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
  o **p1.number**: Integer type variable that stores the floor number
  o **p1.FC**: Sensor type object that represents the limit switch of the floor
  o **p1.FCA**: Sensor type object that represents the upper limit switch for auto-leveling.
  o **p1.FCB**: Sensor type object that represents the lower limit switch for auto-leveling.
  o **p1.door**: Sensor type object that represents the limit switch of the door
  o **p1.button**: Sensor type object that represents the button of the floor
  o **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
  o **p2.number**: Integer type variable that stores the floor number
  o **p2.FC**: Sensor type object that represents the limit switch of the floor
  o **p2.FCA**: Sensor type object that represents the upper limit switch for auto-leveling.
  o **p2.FCB**: Sensor type object that represents the lower limit switch for auto-leveling.
  o **p2.door**: Sensor type object that represents the limit switch of the door
  o **p2.button**: Sensor type object that represents the button of the floor
  o **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

<table>
<thead>
<tr>
<th>Input</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0 button</td>
<td>13</td>
</tr>
<tr>
<td>FC floor 0</td>
<td>7</td>
</tr>
<tr>
<td>FCA floor 0</td>
<td>8</td>
</tr>
<tr>
<td>FCB floor 0</td>
<td>10</td>
</tr>
<tr>
<td>Door closed 0</td>
<td>11</td>
</tr>
<tr>
<td>P1 button</td>
<td>23</td>
</tr>
<tr>
<td>FC floor 1</td>
<td>19</td>
</tr>
<tr>
<td>FCA floor 1</td>
<td>22</td>
</tr>
<tr>
<td>FCB floor 1</td>
<td>24</td>
</tr>
<tr>
<td>Door closed 1</td>
<td>21</td>
</tr>
<tr>
<td>P2 button</td>
<td>33</td>
</tr>
<tr>
<td>FC floor 2</td>
<td>29</td>
</tr>
<tr>
<td>FCA floor 2</td>
<td>36</td>
</tr>
<tr>
<td>FCB floor 2</td>
<td>38</td>
</tr>
<tr>
<td>Door closed 2</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor</td>
<td>15</td>
</tr>
<tr>
<td>Valve</td>
<td>18</td>
</tr>
<tr>
<td>Lock door 0</td>
<td>12</td>
</tr>
<tr>
<td>Lock door 1</td>
<td>26</td>
</tr>
<tr>
<td>Lock door 2</td>
<td>35</td>
</tr>
</tbody>
</table>

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.

```python
265 def AL0_up(port):
266     level = 0
267     AL_up(level)
268     return
269
270 def AL0_down(port):
271     level = 0
272     AL_down(level)
274     return
```
```python
def AL_up(piso):
    while (P[piso].FC.value() != 1):
        motor.salida(1)
        return

def AL_down(piso):
    while (P[piso].FC.value() != 1):
        valve.salida(1)
    valve.salida(0)
    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.

```python
def enable_int_general(P):
    # Enables button 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

Figure 42: Button interrupt enabling function
```

```python
def enable_AL(actual, P):
    # Enables autolevel interrupts depending on the floor number
    if (actual == 0):
        enable_int_0(P)
    if (actual == 1):
        enable_int_1(P)
    if (actual == 2):
        enable_int_2(P)

    return

def enable_int_0(P):
    # Enables Autolevel interrupts for floor 0
    GPIO.add_event_detect(P[0].FCA.port, GPIO.RISING, callback=AL0_down, bouncetime=500)
    GPIO.add_event_detect(P[0].FCB.port, GPIO.RISING, callback=AL0_up, bouncetime=500)

    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():

```
def disable_int(P):
    # Disable 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)
    GPIO.remove_event_detect(P[0].FCA.port)
    GPIO.remove_event_detect(P[0].FCB.port)
    GPIO.remove_event_detect(P[1].FCA.port)
    GPIO.remove_event_detect(P[1].FCB.port)
    GPIO.remove_event_detect(P[2].FCA.port)
    GPIO.remove_event_detect(P[2].FCB.port)
    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>
<thead>
<tr>
<th>Output</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor 1</td>
<td>15</td>
</tr>
<tr>
<td>Valve 1</td>
<td>18</td>
</tr>
<tr>
<td>Motor 2</td>
<td>16</td>
</tr>
<tr>
<td>Valve 2</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 8: Added outputs for two speeds

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.

```python
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
```

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 represents the limit switch of the floor
    - **p0.FCA**: Sensor type object that represents 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 represents the limit switch of the floor
    - **p1.FCA**: Sensor type object that represents 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 represents the limit switch of the floor
    - **p2.FCA**: Sensor type object that represents 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

<table>
<thead>
<tr>
<th>Input</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0 button</td>
<td>13</td>
</tr>
<tr>
<td>FC floor 0</td>
<td>7</td>
</tr>
<tr>
<td>FCA floor 0</td>
<td>8</td>
</tr>
<tr>
<td>Lock FCO 0</td>
<td>12</td>
</tr>
<tr>
<td>Lock FCI 0</td>
<td>16</td>
</tr>
<tr>
<td>Door closed 0</td>
<td>11</td>
</tr>
<tr>
<td>P1 button</td>
<td>23</td>
</tr>
<tr>
<td>FC floor 1</td>
<td>19</td>
</tr>
<tr>
<td>FCA floor 1</td>
<td>18</td>
</tr>
<tr>
<td>Lock FCO 1</td>
<td>24</td>
</tr>
<tr>
<td>Lock FCI 1</td>
<td>26</td>
</tr>
<tr>
<td>Door closed 1</td>
<td>21</td>
</tr>
<tr>
<td>P2 button</td>
<td>33</td>
</tr>
<tr>
<td>FC floor 2</td>
<td>29</td>
</tr>
<tr>
<td>FCA floor 2</td>
<td>28</td>
</tr>
<tr>
<td>Lock FCO 2</td>
<td>36</td>
</tr>
<tr>
<td>Lock FCI 2</td>
<td>38</td>
</tr>
<tr>
<td>Door closed 2</td>
<td>31</td>
</tr>
<tr>
<td>Output</td>
<td>Port</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Motor</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>motor</td>
<td>3</td>
</tr>
<tr>
<td>Valve</td>
<td>37</td>
</tr>
<tr>
<td>Lock door 0</td>
<td>15</td>
</tr>
<tr>
<td>Lock out 0</td>
<td>10</td>
</tr>
<tr>
<td>Lock door 1</td>
<td>21</td>
</tr>
<tr>
<td>Lock out 1</td>
<td>22</td>
</tr>
<tr>
<td>Lock door 2</td>
<td>35</td>
</tr>
<tr>
<td>Lock out 2</td>
<td>32</td>
</tr>
</tbody>
</table>
**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.

```python
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     # value of the actual floor. If the platform is not in any floor,
135     # it descends until it reaches one and sets "actual" and "destination"
136     # to the floor number.
137     actual = -1
138     for p in P:
139         if (p.FC.value() == 1):
140             actual = p.number
141             destination = actual
142         else:
143             while (p.puerta.value() != 1):
144                 pass
145             p.lock.set_value(1)
146             if (actual < 0):
147                 while((P[0].FCA.value() or
148                       P[1].FCA.value()) == 0):
149                     valve.set_value(1)
150                     valve.set_value(0)
151         for p in P:
152             if(p.FCA.value() == 1):
153                 while(p.lock_FCO.value() != 1):
154                     p.lock_out.set_value(1)
155                     p.lock_out.set_value(0)
156                 while(p.FC.value() != 1):
157                     valve.set_value(1)
158                     valve.set_value(0)
159             actual = p.number
160             destination = actual
161             p.lock.set_value(0)
162     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.
while enable_int:
    # Destination door is unlocked, "actual" value is
    # 1.-Platform elevates until 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():
        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)
                while(P[actual].lock_FCI.value() != 1):
                    P[actual].lock_out.set_value(0)
                if(P[destination].lock_FCO.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].lock_FCO.value() != 1):
                    motor.set_value(1)
                while(P[destination].FCA.value() != 1):
                    P[actual].lock_out.set_value(0)
                    P[destination].lock_out.set_value(1)
                if(P[destination].FC.value() != 1):
                    valve.set_value(1)
                    valve.set_value(0)
            except:
                GPIO.cleanup()
        except: raise

Figure 48: Hydraulic lock elevator's main program

ERASMUS PROJECT

URTZI AGIRRE ALBIZU
60 ELECTRONIC REDESIGN OF AN INDUSTRIAL LIFT
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

<table>
<thead>
<tr>
<th>Output</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>motor 1</td>
<td>3</td>
</tr>
<tr>
<td>Valve 1</td>
<td>5</td>
</tr>
<tr>
<td>Motor 2</td>
<td>37</td>
</tr>
<tr>
<td>Valve 2</td>
<td>40</td>
</tr>
</tbody>
</table>

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.

```python
if (actual > destination):
    while(P[actual].FCA.value() != 1):
        motors[0].set_value(1)
        motors[1].set_value(0)
        while(P[actual].lock_FCI.value() != 1):
            P[actual].lock_out.set_value(0)
        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):
            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)
```

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 suit 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:

![Proposed input interface](image)

*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.

![Selected output interface](image1)

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.

![Jumper wires](image2)

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.

![GPIO testing diagram]

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 were 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.

![Python 3.5.1 Shell]

```python
>>> import sys
>>> sys.path.append('/home/urzti/Repos/Industrial-Lift/Python/programs')
>>> import test

Elevator is in floor 0
inhibit destination--> 1
motor ON
FCG deactivated
FCI activated
motor OFF
Elevator is in floor 1
inhibit destination--> 0
valve ON
FCG deactivated
FCI activated
valve OFF
Elevator is in floor 0
inhibit destination--> 0

>>> |
```

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.
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.
#Hardware test

#Inputs:    FC1
#           FC2
#           virtual input

#outputs:
#           up
#           down

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

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)

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

##FUNCIONES

def define_I0s():
    # Function that configures all the objects of the program and assigns
    # the GPIO ports to each sensor and actuator
    p0 = floor(0)
    p1 = floor(1)
    P = [p0, p1]

    motor = actuator("motor", 18)
    up = actuator("up", 23)
    down = actuator("down", 25)
    valves = [up, down]

    P[0].FC.port = 17
    P[1].FC.port = 4

    return P, motor, valves
def GPIO_config(P, motor, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(motor.port, GPIO.OUT)
    GPIO.setup(valves[0].port, GPIO.OUT)
    GPIO.setup(valves[1].port, GPIO.OUT)
    return

##MAIN

global destino
[P, motor, valves] = define_IOs()
GPIO_config(P, motor, valves)
motor.output(1)
valves[0].output(1)
valves[1].output(1)
actual = 1
destino = 1
while(1):
    try:
        print("lift is on floor "+str(actual))
destino = int(input("introduce floor number--> "))
        if (actual != destino):
            while(P[destino].FC.value() != 0):
                if (actual < destino):
                    motor.output(0)
                else:
                    destino = actual
        elif(actual > destino):
            motor.output(0)
        else:
            destino = actual
except:
    GPIO.cleanup()
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

<table>
<thead>
<tr>
<th>Element</th>
<th>Quantity</th>
<th>Price</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi 2</td>
<td>1</td>
<td>40,00 €</td>
<td>40,00 €</td>
</tr>
<tr>
<td>Power supply</td>
<td>1</td>
<td>8,00 €</td>
<td>8,00 €</td>
</tr>
<tr>
<td>Output interface</td>
<td>1</td>
<td>20,00 €</td>
<td>20,00 €</td>
</tr>
<tr>
<td>microSD card</td>
<td>1</td>
<td>5,00 €</td>
<td>5,00 €</td>
</tr>
<tr>
<td>Jumper wires</td>
<td>1</td>
<td>1,00 €</td>
<td>1,00 €</td>
</tr>
<tr>
<td>Enclosure</td>
<td>1</td>
<td>15,00 €</td>
<td>15,00 €</td>
</tr>
<tr>
<td>Conformal coating</td>
<td>1</td>
<td>10,00 €</td>
<td>10,00 €</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td></td>
<td><strong>99,00 €</strong></td>
</tr>
</tbody>
</table>
3.6. Critical analysis and prospects for improvement

Table 12: SWOT analysis

<table>
<thead>
<tr>
<th>Strengths</th>
<th>Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Versatility: The product is capable of adapting to a lot of products made by the company</td>
<td>• Robustness: The designed product is divided in different parts, which have to be assembled together.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opportunities</th>
<th>Threats</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Innovation: Its innovation capabilities offers the opportunity to differentiate from the competition.</td>
<td>• Product life: As the hardware is manufactured by third parties, there is the possibility that they leave the production</td>
</tr>
</tbody>
</table>

As seen in the SWOT analysis, after the development of the project there are still some points to take care of. As future developments, I would recommend 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/output 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 specific 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.

![Compute module](image-url)
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.

![OS download](https://www.raspberrypi.org/downloads/raspbian/)

**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.

![Win32 Disk Imager download](https://sourceforge.net/projects/win32diskimager/)

**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.

![Win32 Disk Imager](image)

**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.

![ipconfig](image)

**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.

![cmdline.txt](image)

**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 connections click on the network that has Internet connection.

![Network and Sharing Centre](image)

**Figure 63: Network and Sharing Centre**

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

![Network properties](image)

**Figure 64: Network properties**

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

![Putty download](image)

**Download PuTTY**

Putty is an SSH and telnet client, developed with source code and without restrictions.

You can download Putty [here](https://www.putty.org).

**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.

![Putty](image)

**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’.

![RPi login](image)

**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.

![raspi-config](image)

**Figure 68 raspi-config**

After this, execute the following commands to check for updates.
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.

![Remote desktop](image.png)

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
  browsable=Yes
  writable=Yes
  only_guest=no
  create mask=0777
  directory mask=0777
  public=no
  valid users=pi
```

![smb.conf modifications](image.png)

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.

![Win32 Disk Imager download](image)

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.

![Image writing](image)

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.

```python
#!/usr/bin/python3
execfile("/home/pi/Programs/autorun2.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.

```python
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: First 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: First 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 became 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 of application, low cost and good waterproof performance.

Figure 76: Conformal coating

3.8.2. Maintenance
To help maintenance works, a program that registers 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.
import time

def count_usage(peripheric):
    data = time.localtime()
    year = str(data[0])
    mon = str(data[1])
    day = str(data[2])
    hour = str(data[3])
    mins = str(data[4])
    sec = str(data[5])
    file = open("F:\Proyecto ABER\Raspberry Pi\Programs\egister.txt", "a+")
    file.write(year + "/" + mon + "/" + day + ";" + hour + ":" +
                mins + ":" + sec + ";" + str(peripheric.name) + "\n")
    file.close()
    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 Peripherals.

**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


**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 Aellido (2013).** Titulo en cursiva. 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/>


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

<http://www.kevinmfodor.com/home/My-Blog/microcontrollerinputprotectiontechniques>

RENESAS (2013). How Photocouplers/Optocouplers are used.


<http://electronicdesign.com/embedded/engineer-s-guide-high-quality-pcb-design>


6. APPENDIXES
6. Programs

```python
#2P1S1DStandard

# Inputs: P0 button
# P1 button
# FC floor 0
# FC 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
import count
GPIO.setmode(GPIO.BOX)  # 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 assigns the GPIO ports to each sensor and actuator
    P0 = floor(0)
    P1 = floor(1)
    P = [P0, P1]

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

    P[0].FC.port = 7
    P[0].door.port = 11
```

6. Programas

```python
#2P1S1DStandard

# Input: P0 button
# P1 button
# FC floor 0
# FC floor 1
# door closed 0
# door closed 1

# Output: motor
# valve
# lock door 0
# lock door 1

import RPi.GPIO as GPIO  # import GPIO module
import time  # import time module
import count
GPIO.setmode(GPIO.BOX)  # 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 assigns the GPIO ports to each sensor and actuator
    P0 = floor(0)
    P1 = floor(1)
    P = [P0, P1]

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

    P[0].FC.port = 7
    P[0].door.port = 11
```
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)
def disable_int(P):
    # Disables interrupts
    GPIO.remove_event_detect(P[0].button.port)
    GPIO.remove_event_detect(P[1].button.port)
    return

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

P[0].FC.port = 7
P[0].door.port = 11
P[0].button.port = 13
P[0].lock.port = 15

def count_handler(port):
    for p in P:
        if p.FC.port == port:
            peri = p.FC
            break
        elif p.door.port == port:
            peri = p.door
            break
        elif p.button.port == port:
            peri = p.door
            break
        elif p.lock.port == port:
            peri = p.door
            break
        if motor.port == port:
            peri = motor
        elif valve.port == port:
            peri = valve
        count_usage(peri)
        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 opens the valve
# 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.

global destination
[P, motor, valve] = define_IOs()
GPIO_config(P, motor, valve)
[actual, destination] = initialization(P, motor, valve)
enable_int(P)

while(1):
    try:
        time.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):
                valve.set_value(1)
            elif(actual < destination):
                motor.set_value(1)
            else:
                valve.set_value(0)
                motor.set_value(0)
                P[destination].lock.set_value(0)
                actual = destination
                enable_int(P)
        else:
            destination = actual
    except:
        GPIO.cleanup()
raise
# ELECTRONIC REDESIGN OF AN INDUSTRIAL LIFT

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 assigns
    # the GPIO ports to each sensor and actuator
    p0 = floor(0)
    pl = floor(1)
    P = [p0, pl]
    motor = actuator("motor", 0)
    valve = actuator("valve", 0)
def AL_up(level):
    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].door.value() and
                       P[1].door.value()) != 1):
                    pass
                for p in P:
                    p.lock.set_value(1)
                while((P[0].FC.value() or
                       P[1].FC.value()) == 0):
                    valve.set_value(1)
                    valve.set_value(0)
            for p in P:
                if(p.FC.value() == 1):
                    actual = p.number
                    destination = actual
                break
            return actual, destination

def GPIO_config(P, motor, valve):
    # Function that sets the GPIO as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.door.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.button.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.FCA.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.FCB.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(motor.port, GPIO.OUT)
    GPIO.setup(valve.port, GPIO.OUT)
    return

def initialization(P, motor, valve):
    # 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 descends 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].door.value() and
                       P[1].door.value()) != 1):
                        pass
                    for p in P:
                        p.lock.set_value(1)
                    while((P[0].FC.value() or
                           P[1].FC.value()) == 0):
                        valve.set_value(1)
                        valve.set_value(0)
            for p in P:
                if(p.FC.value() == 1):
                    actual = p.number
                    destination = actual
                break
            return actual, destination

def AL_up(level):
while (P[level].FC.value() != 1):
    motor.set_value(1)
    return

def AL_down(level):
    while (P[level].FC.value() != 1):
        valve.set_value(1)
        return

def enable_int_general(P):
    # Enables button 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)
    return

def enable_int_0(P):
    # Enables Autolevel interrupts for floor 0
    GPIO.add_event_detect(P[0].FCA.port,
                           GPIO.RISING,
                           callback=AL0_down,
                           bouncetime=500)
    GPIO.add_event_detect(P[0].FCB.port,
                           GPIO.RISING,
                           callback=AL0_up,
                           bouncetime=500)
    return

def enable_int_1(P):
    # Enables Autolevel interrupts for floor 1
    GPIO.add_event_detect(P[1].FCA.port,
                           GPIO.RISING,
                           callback=AL1_down,
                           bouncetime=500)
    GPIO.add_event_detect(P[1].FCB.port,
                           GPIO.RISING,
                           callback=AL1_up,
                           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[0].FCA.port)
    GPIO.remove_event_detect(P[0].FCB.port)
    GPIO.remove_event_detect(P[1].FCA.port)
    GPIO.remove_event_detect(P[1].FCB.port)
    return

def enable_AL(actual, P):
    # Enables Autolevel interrupts depending on the floor number
    if (actual == 0):
        enable_int_0(P)
    if (actual == 1):
        enable_int_1(P)
    if (actual == 2):
        enable_int_2(P)
210     return 
211  
212 def P1_handler(port):
213     global destination
214     destination = 0
215     return 
216  
217 def P2_handler(port):
218     global destination
219     destination = 1
220     return 
221  
222 def AL0_up(port):
223     level = 0
224     AL_up(level)
225     return 
226  
227 def AL0_down(port):
228     level = 0
229     AL_down(level)
230     return 
231  
232 def AL1_up(port):
233     level = 1
234     AL_up(level)
235     return 
236  
237 def AL1_down(port):
238     level = 1
239     AL_down(level)
240     return 
241  
242 def AL0_up(port):
243     level = 0
244     AL_up(level)
245     return 
246  
247 def AL0_down(port):
248     level = 0
249     AL_down(level)
250     return 
251  
252 # MAIN
253 # Explanation: First configuration functions are executed and then
254 # it sets "destination" to the number of the floor pressed. If the door
255 # is not closed "destination" is set again to the actual value. If it is,
256 # the door is locked and interrupts are disabled until the movement
257 # is over. The movements is as follows:
258 # 1.-It checks if the destination floor is up or down
259 # 2.-If it is up activates the motor
260 # 3.-If it is down opens the valve
261 # 4.-When the destination limit switch is pressed the valve is
262 # closed and the motor is shut down.
263 # Destination door is unlocked, "actual" value is updated and
264 # interrupts are enabled again.
265 # Autolevel: When the elevator is at rest, if one of the actual floors
266 # autolevel limit switch is activated an interrupt occurs. Depending
267 # on if it is the upper or lower switch the valve or the motor
268 # is activated
269 
270 global destination
271 [P, motor, valve] = define_IOs()
272 GPIO_config(P, motor, valve)
273 [actual, destination] = initialization(P, motor, valve)
274 enable_int_general(P)
275 enable_AL(actual, P)
276 
277 while(1):
278     try:
279         time.sleep(0.3)
280         if (actual != destination and P[actual].door.value() == 1):
281             disable_int(P)
P[actual].lock.set_value(1)
while P[destination].FC.value() != 1:
    if (actual > destination):
        valve.set_value(1)
    elif (actual < destination):
        motor.set_value(1)
valve.set_value(0)
motor.set_value(0)
P[destination].lock.set_value(0)
actual = destination
enable_int_general(P)
enable_AL(actual, P)
else:
    destination = actual
except:
    GPIO.cleanup()
raise
import RPi.GPIO as GPIO # import GPIO module
import time # import time module

GPIO.setmode(GPIO.BOARD) # set pin references to BOARD

## CLASES

### floor:

class 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)

### sensor:

class sensor:
    def __init__(self, name, port):
        self.name = name
        self.port = port

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

### actuator:

class 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 assigns
    # the GPIO ports to each sensor and actuator
    p0 = floor(0)
    p1 = floor(1)
    P = [p0, p1]

e = actuator("motor", 0)
valve1 = actuator("valve1", 0)
valve2 = actuator("valve2", 0)

valves = [valve1, valve2]
def AL_up(level):
    while (P[level].FC.value() != 1):
        motor.set_value(1)

def initialization(p, motor, valves):
    # 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 desends 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].door.value() and
                P[1].door.value() != 1):
                    pass
            for p in P:
                p.lock.set_value(1)
            while ((P[0].FC.value() or
                P[1].FC.value() == 0):
                valves[0].set_value(1)
                valves[0].set_value(0)
            for p in P:
                if (p.FC.value() == 1):
                    actual = p.number
                    destination = actual
                    p.lock.set_value(0)
                    break
    return actual, destination

def AL_up(level):
    while (P[level].FC.value() != 1):
        motor.set_value(1)
def AL_down(level):
    while P[level].FC.value() != 1:
        valves[0].set_value(1)
        valves[1].set_value(0)
        return

def AL0_down(P):
    GPIO.add_event_detect(P[0].FCA.port, GPIO.RISING,
        callback=AL0_down,
        bouncetime=500)
    GPIO.add_event_detect(P[0].FCB.port, GPIO.RISING,
        callback=AL0_up,
        bouncetime=500)
    return

def enable_AL(actual, P):
    if (actual == 0):
        enable_int_0(P)
    if (actual == 1):
        enable_int_1(P)
    return

class P1_handler(dst, self):
    # Sets "destination" to the number of the floor the elevator must go
    destination = dst
    global destination

def enable_AL(actual, P):
    # Enables autolevel interrupts depending on the floor number
    if (actual == 0):
        enable_int_0(P)
    if (actual == 1):
        enable_int_1(P)
    return

# Disables interrupts
GPIO.remove_event_detect(P[0].button.port)
GPIO.remove_event_detect(P[1].button.port)
GPIO.remove_event_detect(P[0].FCA.port)
GPIO.remove_event_detect(P[0].FCB.port)
GPIO.remove_event_detect(P[1].FCA.port)
GPIO.remove_event_detect(P[1].FCB.port)
return

# Enables autolevel interrupts for floor 0
GPIO.add_event_detect(P[0].FCA.port, GPIO.RISING,
    callback=AL0_down,
    bouncetime=500)
GPIO.add_event_detect(P[0].FCB.port, GPIO.RISING,
    callback=AL0_up,
    bouncetime=500)
return

# Enables autolevel interrupts for floor 0
GPIO.add_event_detect(P[0].FCA.port, GPIO.RISING,
    callback=AL0_down,
    bouncetime=500)
GPIO.add_event_detect(P[0].FCB.port, GPIO.RISING,
    callback=AL0_up,
    bouncetime=500)
return

# Enables autolevel interrupts for floor 1
GPIO.add_event_detect(P[1].FCA.port, GPIO.RISING,
    callback=AL1_down,
    bouncetime=500)
GPIO.add_event_detect(P[1].FCB.port, GPIO.RISING,
    callback=AL1_up,
    bouncetime=500)
return

# Enables autolevel interrupts for floor 1
GPIO.add_event_detect(P[1].FCA.port, GPIO.RISING,
    callback=AL1_down,
    bouncetime=500)
GPIO.add_event_detect(P[1].FCB.port, GPIO.RISING,
    callback=AL1_up,
    bouncetime=500)
return

##INTERRUPCIONES
# Sets "destination" to the number of the floor the elevator must go
if (actual == 0):
    enable_int_0(P)
if (actual == 1):
    enable_int_1(P)
return
destination = 0
return
def P2_handler(self):
    global destination
    destination = 1
    return
def AL0_up(self):
    level = 0
    AL_up(level)
    return
def AL0_down(self):
    level = 0
    AL_down(level)
    return
def AL1_up(self):
    level = 1
    AL_up(level)
    return
def AL1_down(self):
    level = 1
    AL_down(level)
    return

##MAIN
# Explanation: First configuration functions are executed and then
# it enters the loop. When a button is pressed a interrupt occurs and
# if 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 vallae 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 interrupt occurs. Depending
# on if it is the upper or lower switch the valve or the motor
# is activated

global destination
[P, motor, valves] = define_I0s()
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].FCA.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
ERASMUS PROJECT
URTZI AGIRRE ALBIZU

#2P2S2DAuto-level

###CLASES

class 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:

def __init__(self, name, port):
    self.name = name
    self.port = port

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

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

---

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

##FUNCIONES

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

URTZI AGIRRE ALBIZU

motor1 = actuator("motor1", 15)
motor2 = actuator("motor2", 16)
valve1 = actuator("valve1", 18)
valve2 = actuator("valve2", 40)
valves = [valve1, valve2]
motors = [motor1, motor2]

P[0].FC.port = 7  # amarillo
P[0].door.port = 11  # azul
P[0].button.port = 13  # morado
P[0].lock.port = 15  # ------
P[0].FCA.port = 8  # naranja
P[0].FCB.port = 10  # verde
P[1].FC.port = 19  # amarillo
P[1].door.port = 21  # azul
P[1].button.port = 23  # morado
P[1].lock.port = 26  # ------
P[1].FCA.port = 22  # naranja
P[1].FCB.port = 24  # verde

return P, motors, valves

def GPIO_config(P, motors, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
for p in P:
    GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.lock.port,GPIO.IN)
    GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)

    GPIO.setup(motors[0].port,GPIO.OUT)
    GPIO.setup(motors[1].port,GPIO.OUT)
    GPIO.setup(valves[0].port,GPIO.OUT)
    GPIO.setup(valves[1].port,GPIO.OUT)

return

def initialization(P, motors, valves):
    # 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 desends 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].door.value() and
               P[1].door.value() != 1):
            pass
        for p in P:
            p.lock.set_value(1)
        while((P[0].FC.value() or
               P[1].FC.value()) == 0):
            valves[0].set_value(1)
            valves[1].set_value(0)
for p in P:
if(p.FC.value() == 1):
    actual = p.number
    destination = actual
break
return actual, destination

def AL_up(level):
    while(p[level].FC.value() != 1):
        motors[0].set_value(1)
        motors[1].set_value(0)
        return

def AL_down(level):
    while(p[level].FC.value() != 1):
        valves[0].set_value(1)
        valves[1].set_value(0)
        return

def enable_int_general(p):
    # Enables button 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)
    return

def enable_int_0(p):
    # Enables Autolevel interrupts for floor 0
    GPIO.add_event_detect(p[0].FCA.port,
        GPIO.RISING,
        callback=AL0_down,
        bouncetime=500)
    GPIO.add_event_detect(p[0].FCB.port,
        GPIO.RISING,
        callback=AL0_up,
        bouncetime=500)
    return

def enable_int_1(p):
    # Enables Autolevel interrupts for floor 1
    GPIO.add_event_detect(p[1].FCA.port,
        GPIO.RISING,
        callback=AL1_down,
        bouncetime=500)
    GPIO.add_event_detect(p[1].FCB.port,
        GPIO.RISING,
        callback=AL1_up,
        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[0].FCA.port)
    GPIO.remove_event_detect(p[0].FCB.port)
    GPIO.remove_event_detect(p[1].FCA.port)
    GPIO.remove_event_detect(p[1].FCB.port)
def enable_AL(actual, P):
    # Enables autolevel interrupts depending on the floor number
    if(actual == 0):
        enable_int_0(P)
    if(actual == 1):
        enable_int_1(P)
    return

##INTERFACIOS
# 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 AL0_up(port):
    level = 0
    AL_up(level)
    return

def AL0_down(port):
    level = 0
    AL_down(level)
    return

def AL1_up(port):
    level = 1
    AL_up(level)
    return

def AL1_down(port):
    level = 1
    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 motor1 and motor2 until FCB, then
#    deactivates motor1.
# 3.-If it is down opens the two valves until FCA, then close one
# 4.-When the destination limit switch is pressed the valve is
#    closed and the motor is shut down.
# 5.- 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 interrupt occurs. Depending
# on if it is the upper or lower switch the valve or the motor
# is activated

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

while():
    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[1].set_value(0)
i = 1
            elif(actual < destination):
                while(P[destination].FCB.value() != 1 and i == 0):
                    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)
    except:
        enable_int_general(P)
enable_AL(actual, P)
        GPIO.cleanup()
        raise

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
ERASMUS PROJECT  URTZI AGIRRE ALBIZU

#2P1S1DHydraulicLocks

# Inputs:  
P0 button
P1 button
FC floor 0
FCA floor 0
tranca FCO floor 0
tranca FCI floor 0
FC floor 1
FCA floor 1
tranca FCO floor 1
tranca FCI floor 1
door closed 0
door closed 1

# Outputs:  
motor
valve
lock door 0
lock door 1
lock_out floor
lock_out floor 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.puerta = sensor("puerta" + str(self.number), 0)
        self.boton = sensor("boton" + str(self.number), 0)
        self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
        self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)
        self.lock_out = actuator("lock_out" + 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 assigns
    # the GPIO ports to each sensor and actuator
    p0 = floor(0)
    pl = floor(1)
ERASMUS PROJECT

P = [p0, p1]
motor = actuator("motor", 0)
valve = actuator("valve", 0)
P[0].FC.port = 4
P[0].puerta.port = 17
P[0].boton.port = 27
P[0].lock.port = 0
P[0].FCA.port = 14
P[0].lock_out.port = 0
P[0].lock_FCO.port = 18
P[0].lock_FCI.port = 23
P[1].FC.port = 10
P[1].puerta.port = 9
P[1].boton.port = 11
P[1].lock.port = 0
P[1].FCA.port = 23
P[1].lock_out.port = 0
P[1].lock_FCO.port = 7
P[1].lock_FCI.port = 12
return P, motor, valve

def GPIO_config(P, motor, valve):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.puerta.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.boton.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.FCA.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock_out.port, GPIO.OUT)
        GPIO.setup(p.lock_FCO.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock_FCI.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(motor.port, GPIO.OUT)
        GPIO.setup(valve.port, GPIO.OUT)
        return

def initialization(P, motor, valve):
    # 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 descends 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.puerta.value() != 1):
                pass
            p.lock.set_value(1)
        if (actual < 0):
            while((P[0].FCA.value() or
                   p[1].FCA.value()) == 0):
                valve.set_value(1)
        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):
    valve.set_value(1)
    valve.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].boton.port, GPIO.FALLING, callback=P1_handler, bouncetime=500)
    GPIO.add_event_detect(P[1].boton.port, GPIO.FALLING, callback=P2_handler, bouncetime=500)

    return

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

##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 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.

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)
    while (P[actual].lock_FCI.value() != 1):
        P[actual].lock_out.set_value(0)
        P[destination].lock_out.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)
else:
    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):
        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)
    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)
    P[actual].lock.set_value(0)
    motor.set_value(0)
actual = destination
except:
    GPIO.cleanup()
raise
#2P1S2D Hydraulic Locks

# Inputs:
- P0 button
- P1 button
- FC floor 0
- FCA floor 0
- tranca FCO floor 0
- tranca FCI floor 0
- FC floor 1
- FCA floor 1
- tranca FCO floor 1
- tranca FCI floor 1
- door closed 0
- door closed 1

# Outputs:
- motor
- valve 1
- valve 2
- lock door 0
- lock door 1
- lock_out floor 0
- lock_out floor 1

```python
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.door = sensor("door" + str(self.number), 0)
        self.button = sensor("button" + str(self.number), 0)
        self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
        self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)
        self.lock_out = actuator("lock_out" + 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 assigns the GPIO ports to each sensor and actuator
    p0 = floor(0)
    p1 = floor(1)
    P = [p0, p1]
```
motor = actuator("motor", 0)
valve1 = actuator("valve1", 0)
valve2 = actuator("valve2", 0)
valves = [valve1, valve2]

P[0].FC.port = 4
P[0].door.port = 17
P[0].button.port = 27
P[0].lock.port = 0
P[0].FCA.port = 14
P[0].lock_out.port = 0
P[0].lock_FCO.port = 18
P[0].lock_FCI.port = 23

P[1].FC.port = 10
P[1].door.port = 9
P[1].button.port = 11
P[1].lock.port = 0
P[1].FCA.port = 23
P[1].lock_out.port = 0
P[1].lock_FCO.port = 7
P[1].lock_FCI.port = 12

return P, motor, valves

def GPIO_config(P, motor, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    # configuracion GPIOs
    GPIO.setup(p.door.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.button.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.lock.port, GPIO.OUT)
    GPIO.setup(p.FCA.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.lock_out.port, GPIO.OUT)
    GPIO.setup(p.lock_FCO.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.lock_FCI.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(motor.port, GPIO.OUT)
    GPIO.setup(valve1.port, GPIO.OUT)
    GPIO.setup(valve2.port, GPIO.OUT)
    return

def initialization(P, motor, valves):
    # 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 desends 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)
            valve.set_value(0)
        for p in P:
            if(p.FC.value() == 1):
                while(p.lock_FCO.value() != 1):
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)
    return

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

##INTERRUPCIONES
# Sets "destination" to the number of the floor the elevator must go

##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 until FCA
# 2.-Hydraulic lock is retracted
# 3.-Destination hydraulic lock is extracted and activate valve1 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 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.

global destination

while(1):
    try:
        if (actual != destination and P[actual].door.value() == 1):
            disable_int(P)

return actual, destination
P[actual].lock.set_value(1)
if (actual > destination):
    while(P[actual].FCA.value() != 1):
        motor.set_value(1)
    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)
    while(P[destination].lock.F.value() !value):  
        enable_int(P)
        actual = destination
        motor.set_value(0)
    P[destination].lock.set_value(0)
except:
    GPIO.cleanup()
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.door = sensor("door" + str(self.number), 0)
        self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
        self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)
        self.lock_out = actuator("lock_out" + 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 assigns
    p0 = floor(0)
ERASMUS PROJECT

URTZI AGIRRE ALBIZU

pl = floor(1)
P = [p0, pl]
motor1 = actuator("motor1", 3)
motor2 = actuator("motor2", 5)
valve1 = actuator("valve1", 37)
valve2 = actuator("valve2", 40)
valves = [valve1, valve2]
motors = [motor1, motor2]
P[0].FC.port = 7
P[0].door.port = 11
P[0].button.port = 13
P[0].lock.port = 15
P[0].FCA.port = 8
P[0].lock_out.port = 10
P[0].lock_FCO.port = 12
P[0].lock_FCI.port = 16
P[1].FC.port = 19
P[1].door.port = 21
P[1].button.port = 23
P[1].lock.port = 27
P[1].FCA.port = 18
P[1].lock_out.port = 22
P[1].lock_FCO.port = 24
P[1].lock_FCI.port = 26
return P, motors, valves

def GPIO_config(P, motors, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port,GPIO.OUT)
        GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock_out.port,GPIO.OUT)
        GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(motor1.port,GPIO.OUT)
    GPIO.setup(motor2.port,GPIO.OUT)
    GPIO.setup(valve1.port,GPIO.OUT)
    GPIO.setup(valve2.port,GPIO.OUT)
    return

def initialization(P, motors, valves):
    # 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 descends 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)
```python
for p in P:
    if (p.FCA.value() == 1):
        while (p.lock_FCO.value() != 1):
            p.lock_out.set_value(1)
            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)
    return

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

# MAIN
```

**#INTERRUPCIONES**

Sets "destination" to the number of the floor the elevator must go

```
def P1_handler(port):
    # 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 until FCA
    #   2.-Hydraulic lock is retracted
    #   3.-Destination hydraulic lock is extracted and activate valve1 and
    #      valve2 until FCA, then deactivate valve2.
    #   4.-When the platform arrives to the destination valve1 is closed
    global destination
    destination = 0
    return

def P2_handler(port):
    # Explanation: 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,
    #   2.-Hydraulic lock is retracted
    #   3.-motor1 is deactivated when it arrives to destinations FCA
    #   4.-Destination hydraulic lock is extracted and valve1 open
    #   5.-When the platform arrives to the destination valve1 is closed
```

```
actual = p.number
destination = actual
p.lock.set_value(0)
return actual, destination
```
# Destination door is unlocked, "actual" value is
# updated and interrupts are enabled again.

```python
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[1].set_value(0)
            while(P[actual].lock_FCI.value() != 1):
                P[actual].lock_out.set_value(0)
                P[destination].lock_out.set_value(1)
            if(P[destination].FCA.value() == 1):
                while(P[destination].FCA.value() != 1):
                    pass
                valves[1].set_value(0)
            while(P[destination].FCA.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):
                motors[0].set_value(1)
                motors[1].set_value(1)
            while(P[destination].FCA.value() != 1):
                pass
            while(P[destination].lock_FCO.value() != 1):
                motors[0].set_value(0)
                P[actual].lock_out.set_value(0)
            while(P[destination].lock_FCO.value() != 1):
                pass
            while(P[destination].FCA.value() != 1):
                pass
        else:
            pass
    except:
        GPIO.cleanup()
        raise
```

URTZI AGIRRE ALBIZU

ERASMUS PROJECT

ELECTRONIC REDESIGN OF AN INDUSTRIAL LIFT
001 #3P1S1DStandard

002 
003 #Inputs:    P0 button
004 #           P1 button
005 #           P2 button
006 #           FC floor 0
007 #           FC floor 1
008 #           FC floor 2
009 #           door closed 0
010 #           door closed 1
011 #           door closed 2
012 #
013 #outputs:   motor
014 #           valve
015 #           lock door 0
016 #           lock door 1
017 #           lock door 2
018
019 import RPi.GPIO as GPIO   # import GPIO module
020 import time               # import time module
021 GPIO.setmode(GPIO.BOARD)  # set pin references to BOARD
022
023
024 ##CLASES
025
026 # Class that represents all the sensors and actuators of a floor
027 class floor:
028     def __init__(self, number):
029         self.number = number
030         self.FC = sensor("FC\" + str(self.number), 0)
031         self.door = sensor("door\" + str(self.number), 0)
032         self.button = sensor("button\" + str(self.number), 0)
033         self.lock = actuator("lock\" + str(self.number), 0)
034
035 # Class that represents a sensor
036 class 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)
043
044 # Class that represents an actuator
045 class actuator:
046     def __init__(self, name, port):
047         self.name = name
048         self.port = port
049
050     def set_value(self, valor):
051         if(valor == 1):
052             GPIO.output(self.port, 0)
053         elif(valor == 0):
054             GPIO.output(self.port, 1)
055
056 ##FUNCIONES
057
058 def define_IOs():
059     # Function that configures all the objects of the program and assigns
060     # the GPIO ports to each sensor and actuator
061     p0 = floor(0)
062     p1 = floor(1)
063     p2 = floor(2)
064     P = [p0, p1, p2]
065
066     motor = actuator("motor", 37)
067     valve = actuator("valve", 38)
def enable_int(P):
# Enables interrupts

def initialization(P, motor, valve):
# 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 descends 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].door.value() and
               P[1].door.value()) != 1):
            pass
        for p in P:
            p.lock.set_value(1)
        while((P[0].FC.value() or
               P[1].FC.value()) == 0):
            valve.set_value(1)
        valve.set_value(0)
    for p in P:
        if(p.FC.value() == 1):
            actual = p.number
            destination = actual
            break
return actual, destination

def enable_int(P):
# Enables interrupts
**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

    # 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. 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 opens the valve
    # 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.
    global destination
    [P, motor, valve] = define_IOS()
    GPIO_config(P, motor, valve)
    [actual, destination] = initialization(P, motor, valve)
    enable_int(P)
    while(1):
        try:
            time.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):
    valve.set_value(1)
elif (actual < destination):
    motor.set_value(1)

valve.set_value(0)
motor.set_value(0)
P[destination].lock.set_value(0)

actual = destination

else:
    destination = actual

except:
    GPIO.cleanup()
#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
- 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

```python
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 assigns
    # the GPIO ports to each sensor and actuator
```
p0 = floor(0)
pl = floor(1)
p2 = floor(2)
P = [p0, pl, p2]
motor = actuator("motor", 0)
valve = actuator("valve", 0)
P[0].FC.port = 4
P[0].door.port = 17
P[0].button.port = 27
P[0].lock.port = 0
P[0].FCA.port = 14
P[0].FCB.port = 15
P[1].FC.port = 10
P[1].door.port = 9
P[1].button.port = 11
P[1].lock.port = 0
P[1].FCA.port = 25
P[1].FCB.port = 8
P[2].FC.port = 5
P[2].door.port = 6
P[2].button.port = 13
P[2].lock.port = 0
P[2].FCA.port = 16
P[2].FCB.port = 20
return P, motor, valve

def GPIO_config(P, motor, valve):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port,GPIO.OUT)
        GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(motor.port,GPIO.OUT)
        GPIO.setup(valve.port,GPIO.OUT)
        return

def initialization(P, motor, valve):
    # 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 desends 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].door.value() and
                            P[1].door.value() != 1):
                        pass
                    for p in P:
                        p.lock.set_value(1)
                        while((P[0].FC.value() or
for p in P:
    if(p.FC.value() == 1):
        actual = p.number
        destination = actual
        p.lock.set_value(0)
        break
return actual, destination

def AL_up(level):
    while(P[level].FC.value() != 1):
        motor.set_value(1)
        motor.set_value(0)
    return

def AL_down(level):
    while(P[level].FC.value() != 1):
        valve.set_value(1)
        valve.set_value(0)
    return

def enable_int_general(P):
    # Enables button 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 enable_int_0(P):
    # Enables Autolevel interrupts for floor 0
    GPIO.add_event_detect(P[0].FCA.port,
                          GPIO.RISING,
                          callback=AL0_down,
                          bouncetime=500)
    GPIO.add_event_detect(P[0].FCB.port,
                          GPIO.RISING,
                          callback=AL0_up,
                          bouncetime=500)
    return

def enable_int_1(P):
    # Enables Autolevel interrupts for floor 1
    GPIO.add_event_detect(P[1].FCA.port,
                          GPIO.RISING,
                          callback=AL1_down,
                          bouncetime=500)
    GPIO.add_event_detect(P[1].FCB.port,
                          GPIO.RISING,
                          callback=AL1_up,
                          bouncetime=500)
    return

def enable_int_2(P):
    # Enables Autolevel interrupts for floor 2
    GPIO.add_event_detect(P[2].FCA.port,
def disable_int(P):
    GPIO.remove_event_detect(P[0].button.port)
    GPIO.remove_event_detect(P[1].button.port)
    GPIO.remove_event_detect(P[2].button.port)
    GPIO.remove_event_detect(P[0].FCA.port)
    GPIO.remove_event_detect(P[0].FCB.port)
    GPIO.remove_event_detect(P[1].FCA.port)
    GPIO.remove_event_detect(P[1].FCB.port)
    GPIO.remove_event_detect(P[2].FCA.port)
    GPIO.remove_event_detect(P[2].FCB.port)
    return

def enable_AL(actual, P):
    if(actual == 0):
        enable_int_0(P)
    if(actual == 1):
        enable_int_1(P)
    if(actual == 2):
        enable_int_2(P)
    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

def AL0_up(port):
    level = 0
    AL_up(level)
    return

def AL0_down(port):
    level = 0
    AL_down(level)
    return

def AL1_up(port):
    level = 1
```python
def AL1_down(port):
    level = 1
    AL_down(level)
    return

def AL2_up(port):
    level = 2
    AL_up(level)
    return

def AL2_down(port):
    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 opens the valve
# 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 interrupt occurs. Depending
# on if it is the upper or lower switch the valve or the motor
# is activated

[P, motor, valve] = define_IOS()
GPIO_config(P, motor, valve)
[actual, destination] = initialization(P, motor, valve)
enable_int_general(P)
enable_AL(actual, P)

while(1):
    try:
        time.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):
                    valve.set_value(1)
                    else:
                    if (actual < destination):
                        motor.set_value(1)
                        else:
                        valve.set_value(0)
                        motor.set_value(0)
                        P[destination].lock.set_value(0)
                        actual = destination
                        else:
                            destination = actual
                            GPIO.cleanup()
                            raise
```

This code snippet deals with the electronic redesign of an industrial lift. It includes functions for moving up (AL_up) and down (AL_down) floors, as well as auto-leveling features when the elevator is at rest. The code is designed to handle various scenarios such as enabling and disabling interrupts, locking the door, and activating the motor or valve based on the floor destination.
#3P1S2DAuto-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 1
#          valve 2
#          lock door 0
#          lock door 1
#          lock door 2

import RPi.GPIO as GPIO
import time
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 assigns
    # the GPIO ports to each sensor and actuator
    p0 = floor(0)
    p1 = floor(1)
ERASMUS PROJECT

p2 = floor(2)
P = [p0, p1, p2]
motor = actuator("motor", 0)
valve1 = actuator("valve1", 0)
valve2 = actuator("valve2", 0)
valves = [valve1, valve2]
P[0].FC.port = 4
P[0].door.port = 17
P[0].button.port = 27
P[0].lock.port = 0
P[0].FCA.port = 14
P[0].FCB.port = 15
P[1].FC.port = 10
P[1].door.port = 9
P[1].button.port = 11
P[1].lock.port = 0
P[1].FCA.port = 25
P[1].FCB.port = 8
P[2].FC.port = 5
P[2].door.port = 6
P[2].button.port = 13
P[2].lock.port = 0
P[2].FCA.port = 16
P[2].FCB.port = 20
return P, motor, valves

def GPIO_config(P, motor, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port,GPIO.OUT)
        GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.FCB.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(motor.port,GPIO.OUT)
        GPIO.setup(valve1.port,GPIO.OUT)
        GPIO.setup(valve2.port,GPIO.OUT)
        return

def initialization(P, motor, valves):
    # 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 descends 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].FC.value() and
                       P[1].FC.value()) == 1):
                    pass
            for p in P:
                p.lock.set_value(1)
                while((P[0].FC.value() or
                       P[1].FC.value()) == 0):
valves[0].set_value(1)
valves[0].set_value(0)

for p in P:
    if(p.FC.value() == 1):
        actual = p.number
        destination = actual
        p.lock.set_value(0)
        break
return actual, destination

def AL_up(level):
    while(P[level].FC.value() != 1):
        motor.set_value(1)
        motor.set_value(0)
    return

def AL_down(level):
    while(P[level].FC.value() != 1):
        valves[0].set_value(1)
        valves[1].set_value(0)
        valves[0].set_value(0)
    return

def enable_int_general(p):
    # Enables button 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 enable_int_0(p):
    # Enables Autolevel interrupts for floor 0
    GPIO.add_event_detect(P[0].FCA.port, GPIO.RISING,
                           callback=AL0_down,
                           bouncetime=500)
    GPIO.add_event_detect(P[0].FCB.port, GPIO.RISING,
                           callback=AL0_up,
                           bouncetime=500)
    return

def enable_int_1(p):
    # Enables Autolevel interrupts for floor 1
    GPIO.add_event_detect(P[1].FCA.port, GPIO.RISING,
                           callback=AL1_down,
                           bouncetime=500)
    GPIO.add_event_detect(P[1].FCB.port, GPIO.RISING,
                           callback=AL1_up,
                           bouncetime=500)
    return

def enable_int_2(p):
    # Enables Autolevel interrupts for floor 2
    GPIO.add_event_detect(P[2].FCA.port, GPIO.RISING,
                           callback=AL2_down,
                           bouncetime=500)
GPIO.add_event_detect(P[2].FCB.port, GPIO.RISING, callback=AL2_up, 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)
    GPIO.remove_event_detect(P[0].FCA.port)
    GPIO.remove_event_detect(P[1].FCA.port)
    GPIO.remove_event_detect(P[0].FCB.port)
    GPIO.remove_event_detect(P[1].FCB.port)
    GPIO.remove_event_detect(P[2].FCB.port)

    return

def enable_AL(actual, P):
    # Enables autolevel interrupts depending on the floor number
    if(actual == 0):
        enable_int_0(P)
    if(actual == 1):
        enable_int_1(P)
    if(actual == 2):
        enable_int_2(P)

    return

##INTERRUPCIONES

# Sets "destination" to the number of the floor the elevator must go

def P1_handler(self):
    global destination
    destination = 0
    return

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

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

def AL0_up(self):
    level = 0
    AL_up(level)
    return

def AL0_down(self):
    level = 0
    AL_down(level)
    return

def AL1_up(self):
    level = 1
    AL_up(level)
    return

def AL1_down(self):
    level = 1
    AL_down(level)
    return

def AL2_up(self):
level = 2

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 interrupt 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].FCA.value() == 1):
            if (actual > destination):
                while(P[destination].FCA.value() == 1 and i == 0)
                    valves[0].set_value(1)
                    valves[1].set_value(1)
                    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)
    except:
        GPIO.cleanup() 
        raise

134 ELECTRONIC REDESIGN OF AN INDUSTRIAL LIFT
import RPi.GPIO as GPIO # import GPIO module
import time # import time module
GPIO.setmode(GPIO.BORDER) # 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 assigns
p0 = floor(0)
p1 = floor(1)
p2 = floor(2)
P = [p0, p1, p2]

motor1 = actuator("motor1", 15)
motor2 = actuator("motor2", 16)
valve1 = actuator("valve1", 18)
valve2 = actuator("valve2", 40)
valves = [valve1, valve2]
motors = [motor1, motor2]

P[0].FC.port = 7 # amarillo
P[0].door.port = 11 # azul
P[0].button.port = 13 # morado
P[0].lock.port = 15 # -----
P[0].FCA.port = 8 # naranja
P[0].FCB.port = 10 # verde

P[1].FC.port = 19 # amarillo
P[1].door.port = 21 # azul
P[1].button.port = 23 # morado
P[1].lock.port = 26 # -----
P[1].FCA.port = 22 # naranja
P[1].FCB.port = 24 # verde

P[2].FC.port = 29 # amarillo
P[2].door.port = 31 # azul
P[2].button.port = 33 # morado
P[2].lock.port = 35 # -----
P[2].FCA.port = 36 # naranja
P[2].FCB.port = 38 # verde
return P, motors, valves

def GPIO_config(P, motors, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.door.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.button.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port, GPIO.OUT)
        GPIO.setup(p.FCA.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.FCB.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(motors[0].port, GPIO.OUT)
        GPIO.setup(motors[1].port, GPIO.OUT)
        GPIO.setup(valves[0].port, GPIO.OUT)
        GPIO.setup(valves[1].port, GPIO.OUT)
    return

def initialization(P, motors, valves):
    # 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 descends 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
def AL_up(level):
    while(P[level].FC.value() != 1):
        motors[0].set_value(1)
        motors[1].set_value(0)
        motors[0].set_value(0)
        return

def AL_down(level):
    while(P[level].FC.value() != 1):
        valves[0].set_value(1)
        valves[1].set_value(0)
        return

def enable_int_general(p):
    # Enables button 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)

    def enable_int_0(p):
        # Enables Autolevel interrupts for floor 0
        GPIO.add_event_detect(P[0].FCB.port,
            GPIO.RISING,
            callback=AL0_down,
            bouncetime=500)
        GPIO.add_event_detect(P[0].FCA.port,
            GPIO.RISING,
            callback=AL0_up,
            bouncetime=500)
        return

    def enable_int_1(p):
        # Enables Autolevel interrupts for floor 1
        GPIO.add_event_detect(P[1].FCB.port,
            GPIO.RISING,
            callback=AL1_down,
            bouncetime=500)
        return
def enable_int_2(P):
    # Enables Autolevel interrupts for floor 2
    GPIO.add_event_detect(P[2].FCA.port,
        GPIO.RISING,
        callback=AL2_down,
        bouncetime=500)
return

def enable_AL(actual, P):
    # Enables autolevel interrupts depending on the floor number
    if(actual == 0):
        enable_int_0(P)
    if(actual == 1):
        enable_int_1(P)
    if(actual == 2):
        enable_int_2(P)
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

def AL0_up(port):
    level = 0
    AL_up(level)
    return
```python
def AL0_down(port):
    level = 0
    AL_down(level)
    return

def AL1_up(port):
    level = 1
    AL_up(level)
    return

def AL1_down(port):
    level = 1
    AL_down(level)
    return

def AL2_up(port):
    level = 2
    AL_up(level)
    return

def AL2_down(port):
    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 motor1 and motor2 until FCB, then
#     deactivates motor1.
# 3.-If it is down opens the two valves until FCA, then close one
# 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 interrupt occurs. Depending
# on if it is the upper or lower switch the valve or the motor
# is activated
global destination
P, motors, valves = define_IOs()
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)
                        i = 1
                elif(actual < destination):
```

---

ERASMUS PROJECT

URTZI AGIRRE ALBIZU

139 ELECTRONIC REDESIGN OF AN INDUSTRIAL LIFT
while \(P[\text{destination}].\text{PCB}.\text{value}() \neq 1 \text{ and } i == 0\):
    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)
\(P[\text{destination}].\text{lock}.\text{set_value}(0)\)
actual = destination
i = 0
enable_int_general(P)
enable_AL(actual, P)
except:
    GPIO.cleanup()
raise
#3P1S1DHydraulicLocks

```python
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.puerta = sensor("puerta" + str(self.number), 0)
        self.boton = sensor("boton" + str(self.number), 0)
        self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
        self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
        self.lock = actuator("lock" + str(self.number), 0)
        self.lock_out = actuator("lock_out" + 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)
```

# Inputs:
- P0 button
- P1 button
- P2 button
- FC floor 0
- FCA floor 0
- tranca FCO floor 0
- tranca FCI floor 0
- FC floor 1
- FCA floor 1
- tranca FCO floor 1
- tranca FCI floor 1
- FC floor 2
- FCA floor 2
- tranca FCO floor 2
- tranca FCI floor 2
- door closed 0
- door closed 1
- door closed 2

# Outputs:
- motor
- valve
- lock door 0
- lock door 1
- lock door 2
- lock_out floor 0
- lock_out floor 1
- lock_out floor 2
##FUNCIONES

```python
# define_IOs():
# Function that configures all the objects of the program and assigns
# the GPIO ports to each sensor and actuator
p0 = floor(0)
p1 = floor(1)
p2 = floor(2)
P = [p0, p1, p2]
motor = actuator("motor", 0)
valve = actuator("valve", 0)
P[0].FC.port = 4
P[0].puerta.port = 17
P[0].boton.port = 27
P[0].lock.port = 0
P[0].FCA.port = 14
P[0].lock_out.port = 0
P[0].lock_FCO.port = 18
P[0].lock_FCI.port = 23
P[1].FC.port = 10
P[1].puerta.port = 9
P[1].boton.port = 11
P[1].lock.port = 0
P[1].FCA.port = 23
P[1].lock_out.port = 0
P[1].lock_FCO.port = 7
P[1].lock_FCI.port = 12
P[2].FC.port = 5
P[2].puerta.port = 6
P[2].boton.port = 13
P[2].lock.port = 0
P[2].FCA.port = 16
P[2].lock_out.port = 0
P[2].lock_FCO.port = 21
P[2].lock_FCI.port = 26
return P, motor, valve
```

```python
def GPIO_config(P, motor, valve):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.puerta.port, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.boton.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port,GPIO.OUT)
        GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock_out.port,GPIO.OUT)
        GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(motor.port,GPIO.OUT)
        GPIO.setup(valve.port,GPIO.OUT)
    return
```

```python
def initialization(P, motor, valve):
    # 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 desends 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.puerta.value() != 1):
        pass
    p.lock.set_value(1)
if (actual < 0):
    while((P[0].FCA.value() or P[1].FCA.value()) == 0):
        valve.set_value(1)
    valve.set_value(0)
for p in P:
    if(p.FCA.value() == 1):
        while(p.lock_FC0.value() != 1):
            p.lock_out.set_value(1)
        p.lock_out.set_value(0)
        while(p.FC.value() != 1):
            valve.set_value(1)
        valve.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].boton.port, GPIO.FALLING, callback=P1_handler, bouncetime=500)
    GPIO.add_event_detect(P[1].boton.port, GPIO.FALLING, callback=P2_handler, bouncetime=500)
    GPIO.add_event_detect(P[2].boton.port, GPIO.FALLING, callback=P3_handler, bouncetime=500)
    return

def disable_int(P):
    # Disables interrupts
    GPIO.remove_event_detect(P[0].boton.port)
    GPIO.remove_event_detect(P[1].boton.port)
    GPIO.remove_event_detect(P[2].boton.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 raises until FCA
# 2.-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.

global destination

[P, motor, valve] = define_IOs()
GPIO_config(P, motor, valve)
[actual, destination] = initialization(P, motor, valve)
enable_int(P)

while():
    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)
                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)
        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)
                    while(P[destination].lock_FCI.value() != 1):
                        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()
ERASMUS PROJECT
URTZI AGIRRE ALBIZU

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

036 #CLASSES

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)

056 class sensor:
057 # Class that represents a sensor
058     def __init__(self, name, port):
059         self.name = name
060         self.port = port
061     def value(self):
062         return GPIO.input(self.port)

069 class actuator:
070 # Class that represents an actuator
071     def __init__(self, name, port):
072         self.name = name
073         self.port = port
074     def set_value(self, valor):
075         if(valor == 1):
076             GPIO.output(self.port, 0)
077         elif(valor == 0):
078             GPIO.output(self.port, 1)
##FUNCIONES

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

    motor = actuator("motor", 0)
    valve1 = actuator("valve1", 0)
    valve2 = actuator("valve2", 0)
    valves = [valve1, valve2]

    P[0].FC.port = 4
    P[0].door.port = 17
    P[0].button.port = 27
    P[0].lock.port = 0
    P[0].FCA.port = 14
    P[0].lock_out.port = 0
    P[0].lock_FCO.port = 18
    P[0].lock_FCI.port = 23

    P[1].FC.port = 10
    P[1].door.port = 9
    P[1].button.port = 11
    P[1].lock.port = 0
    P[1].FCA.port = 23
    P[1].lock_out.port = 0
    P[1].lock_FCO.port = 7
    P[1].lock_FCI.port = 12

    P[2].FC.port = 5
    P[2].door.port = 6
    P[2].button.port = 13
    P[2].lock.port = 0
    P[2].FCA.port = 16
    P[2].lock_out.port = 0
    P[2].lock_FCO.port = 21
    P[2].lock_FCI.port = 26

    return P, motor, valves

def GPIO_config(P, motor, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program

    for p in P:
        GPIO.setup(p.FC.port, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    # configuracion GPIOs
    GPIO.setup(p.door.port, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.button.port, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.lock.port, GPIO.OUT)
    GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.lock_out.port,GPIO.OUT)
    GPIO.setup(p.lock_FCO.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(p.lock_FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)

    GPIO.setup(motor.port,GPIO.OUT)
    GPIO.setup(valve1.port,GPIO.OUT)
    GPIO.setup(valve2.port,GPIO.OUT)

    return

def initialization(P, motor, valves):
    # 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 descends 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)
        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):
            if (actual < 0):
                while((P[0].FCA.value()) or (P[1].FCA.value()) == 0):
                    valves[0].set_value(1)
                    valve.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
ERASMUS PROJECT  URTZI AGIRRE ALBIZU
148 ELECTRONIC REDESIGN OF AN INDUSTRIAL LIFT

# 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 until FCA
# 2.-Hydraulic lock is retracted
# 3.-Destination hydraulic lock is extracted and activate valvel and
# valve2 until FCA, then deactivate valve2.
# 4.-When the platform arrives to the destination valvel 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.

global destination
[P, motor, valves] = define_IOS()
GPIO_config()
[actual, destination] = initialization(P, motor, valves)
enable_int(P)

while():
    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].FC.value() != 1):
                                pass
                                pass
                                while(P[destination].FC.value() != 1):
                                    pass
                                    pass
                                    while(P[destination].lock_FCI.value() != 1):
                                        while(P[destination].lock_out.set_value(0)
                                            while(P[destination].lock_FCO.value() != 1):
                                                while(P[destination].lock_out.set_value(0)
                                                    while(P[destination].lock_FCO.value() != 1):
                                                        while(P[destination].lock_out.set_value(0)
                                                            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
ERASMUS PROJECT  
URTZI AGIRRE ALBIZU

#3P2S2DHydraulicLocks

# Inputs:  P0 button  
P1 button  
P2 button  
FC floor 0  
FCA floor 0  
tranca FCO floor 0  
tranca FCI floor 0  
FC floor 1  
FCA floor 1  
tranca FCO floor 1  
tranca FCI floor 1  
FC floor 2  
FCA floor 2  
tranca FCO floor 2  
tranca FCI floor 2  
door closed 0  
door closed 1  
door closed 2

# Outputs:  motor 1  
motor 2  
valve 1  
valve 2  
lock door 0  
lock door 1  
lock door 2  
lock out floor 0  
lock out floor 1  
lock out floor 2

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

#CLASSES

class 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.door = sensor("door" + str(self.number), 0)
        self.button = sensor("button" + str(self.number), 0)
        self.lock_FCO = sensor("lock_FCO" + str(self.number), 0)
        self.lock_FCI = sensor("lock_FCI" + str(self.number), 0)
        self.lock_out = actuator("lock_out" + str(self.number), 0)

    class sensor:
        def __init__(self, name, port):
            self.name = name
            self.port = port
        def value(self):
            return GPIO.heart(self.port)

    class 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):
def initialization(P, motors, valves):
    # Function that set the initial conditions of the system
    p0 = floor(0)
    p1 = floor(1)
    p2 = floor(2)
    P = [p0, p1, p2]
    motor1 = actuator("motor1", 3)
    motor2 = actuator("motor2", 5)
    valve1 = actuator("valve1", 37)
    valve2 = actuator("valve2", 40)
    valves = [valve1, valve2]
    motors = [motor1, motor2]
    P[0].FC.port = 7
    P[0].door.port = 11
    P[0].button.port = 13
    P[0].lock.port = 15
    P[0].FCA.port = 8
    P[0].lock_out.port = 10
    P[0].lock_FCQ.port = 12
    P[0].lock_FCI.port = 16
    P[1].FC.port = 19
    P[1].door.port = 21
    P[1].button.port = 23
    P[1].lock.port = 27
    P[1].FCA.port = 18
    P[1].lock_out.port = 22
    P[1].lock_FCQ.port = 24
    P[1].lock_FCI.port = 26
    P[2].FC.port = 29
    P[2].door.port = 31
    P[2].button.port = 33
    P[2].lock.port = 35
    P[2].FCA.port = 28
    P[2].lock_out.port = 32
    P[2].lock_FCQ.port = 36
    P[2].lock_FCI.port = 38
    return P, motors, valves

def GPIO_config(P, motors, valves):
    # Function that sets the GPIOs as inputs or outputs
    # Its inputs are all the sensors and actuators of the program
    for p in P:
        GPIO.setup(p.FC.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.door.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.button.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock.port,GPIO.OUT)
        GPIO.setup(p.FCA.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.lock_out.port,GPIO.OUT)
        GPIO.setup(p.FCQ.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(p.FCI.port,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(motor1.port,GPIO.OUT)
        GPIO.setup(motor2.port,GPIO.OUT)
        GPIO.setup(valve1.port,GPIO.OUT)
        GPIO.setup(valve2.port,GPIO.OUT)
    return

def initialization(P, motors, valves):
    # Function that set the initial conditions of the system
    p0 = floor(0)
    p1 = floor(1)
    p2 = floor(2)
    P = [p0, p1, p2]
    motor1 = actuator("motor1", 3)
    motor2 = actuator("motor2", 5)
    valve1 = actuator("valve1", 37)
    valve2 = actuator("valve2", 40)
    valves = [valve1, valve2]
    motors = [motor1, motor2]
    P[0].FC.port = 7
    P[0].door.port = 11
    P[0].button.port = 13
    P[0].lock.port = 15
    P[0].FCA.port = 8
    P[0].lock_out.port = 10
    P[0].lock_FCQ.port = 12
    P[0].lock_FCI.port = 16
    P[1].FC.port = 19
    P[1].door.port = 21
    P[1].button.port = 23
    P[1].lock.port = 27
    P[1].FCA.port = 18
    P[1].lock_out.port = 22
    P[1].lock_FCQ.port = 24
    P[1].lock_FCI.port = 26
    P[2].FC.port = 29
    P[2].door.port = 31
    P[2].button.port = 33
    P[2].lock.port = 35
    P[2].FCA.port = 28
    P[2].lock_out.port = 32
    P[2].lock_FCQ.port = 36
    P[2].lock_FCI.port = 38
    return P, motors, valves
# 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 descends 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
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 until FCA
# 2.-Hydraulic lock is retracted
# 3.-Destination hydraulic lock is extracted and activate valve1 and
#   valve2 until FCA, then deactivate valve2.
# 4.-When the platform arrives to the destination valve1 is closed
# 5.-When the platform arrives to the destination valve1 is closed
# 6.-motor1 and motor2 are activated until destination FC,
#   then motor2 is deactivated
# 7.-(no oil) Then valve2 until FCA, then deactivate valve2.
# If the destination is up, the movement is as follows:
# 1.-motor1 and motor2 are activated until destination FC,
# 2.-Hydraulic lock is retracted
# 3.-motor1 is deactivated when it arrives to destinations FCA
# 4.-Destination hydraulic lock is extracted and valve1 open
# 5.-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[0].set_value(0)
            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
            while(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() \neq 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() \neq 1):
    valves[0].set_value(1)
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(0)
except:
    GPIO.cleanup()
raise