Desenvolupament d’un sistema personalitzable de control d’una porta d’accés

21 de juny de 2019

Roger Porta Batalla

Índex

Introducció

Aquest treball neix d'un projecte personal consistent en crear un sistema personalitzable que permeti realitzar diferents accions com a reacció a l'obertura d'una porta d'accés.

Objectiu

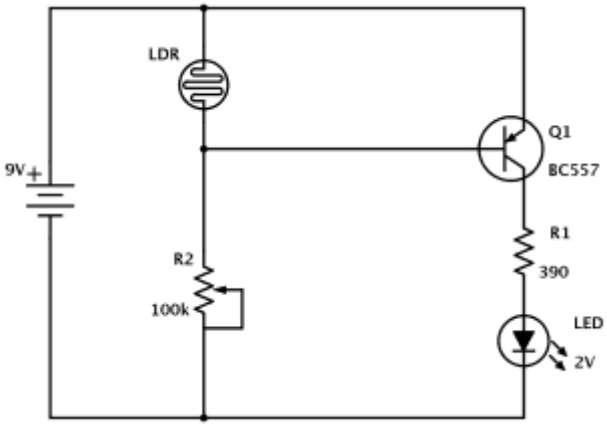
L'objectiu del projecte és realitzar les següents accions com a reacció a l'obertura de la porta sota control:

- Registrar en un fitxer la data i hora a la que s'ha obert la porta. Registrar també la data i hora en què s'ha tornat a tancar
- Avaluar la lluminositat a l'estança a la qual la porta dóna accés i, si és inferior a un determinat valor, encendre el llum d'aquesta estança. Un cop es tanqui la porta, el llum es mantindrà encès durant un interval de temps personalitzable. Transcorregut aquest interval de temps s'apagarà automàticament.
- Notificar via correu electrònic l'obertura de la porta fora de les hores autoritzades

Hardware

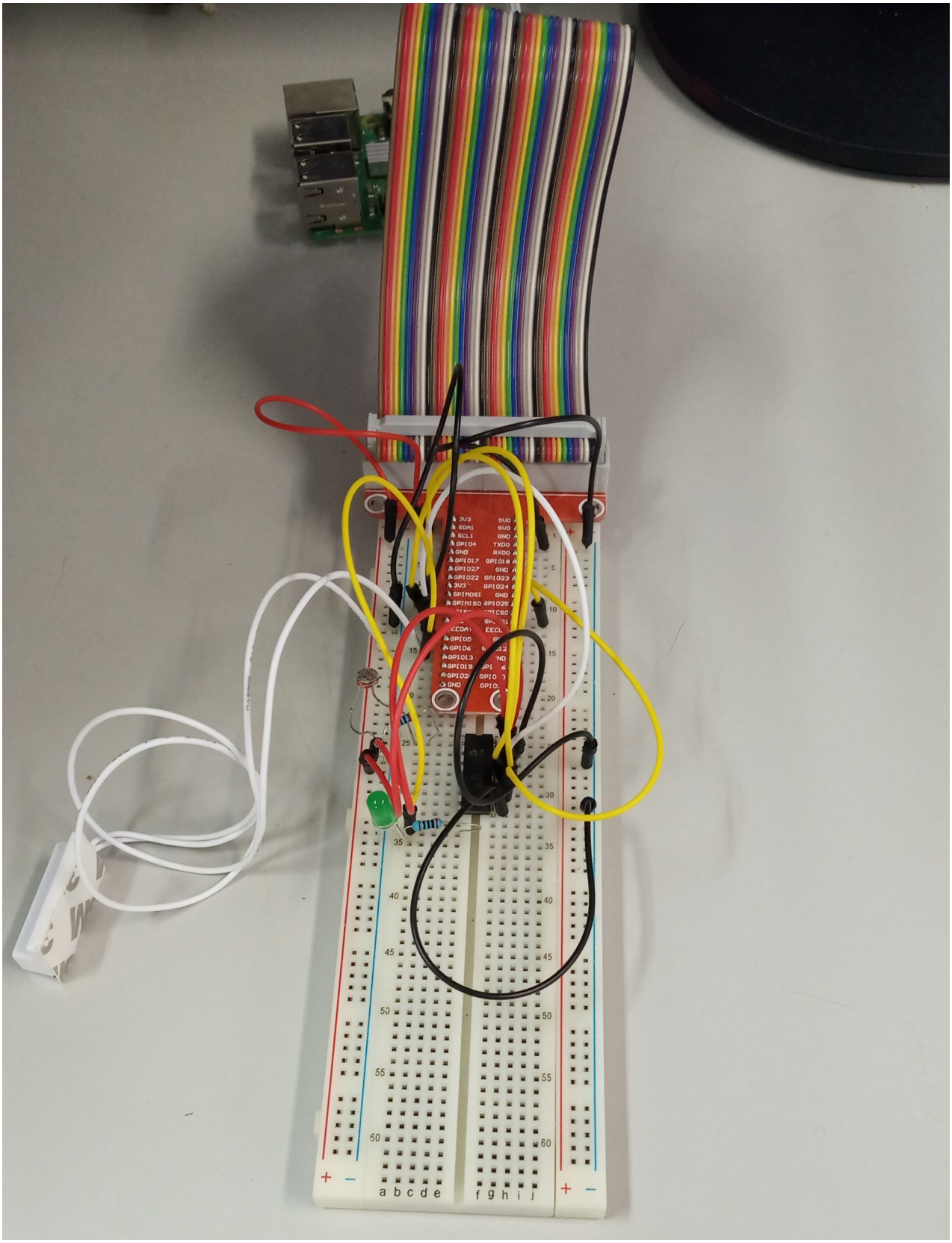
El hardware necessari per al projecte és el següent:

- Raspberry Pi 3 Model B+ o una placa de desenvolupament de característiques similars
- Sensor magnètic («reed switch»)
- «Relé» («relay») per a controlar la llum. En el laboratori hem substituït aquest element per un LED protegit amb una resistència d'1K
- Circuit per a determinar la lluminositat ambient. Aquest circuit es basarà en un fotoresistor (LDR). Al laboratori hem creat un divisor de tensió amb el fotoresistor i una resistència de 10K (fotoresistor a 3,3V i resistència a GND). Llegirem el valor de la lluminositat amb un convertidor analògic/digital (ADC) MCP3008 d'Adafruit connectat als pins SPI de la Raspberry. A la vida real seria més pràctic un circuit similar al de la il·lustració 1.



Il·lustració 1: Diagrama d'exemple d'un circuit per a encendre un LED quan la lluminositat ambient és baixa

Muntatge al laboratori



Il·lustració 2: Muntatge al laboratori

Sensor magnètic

El sensor magnètic presenta dos cables. Connectarem un a 3,3V i l'altre al GPIO5

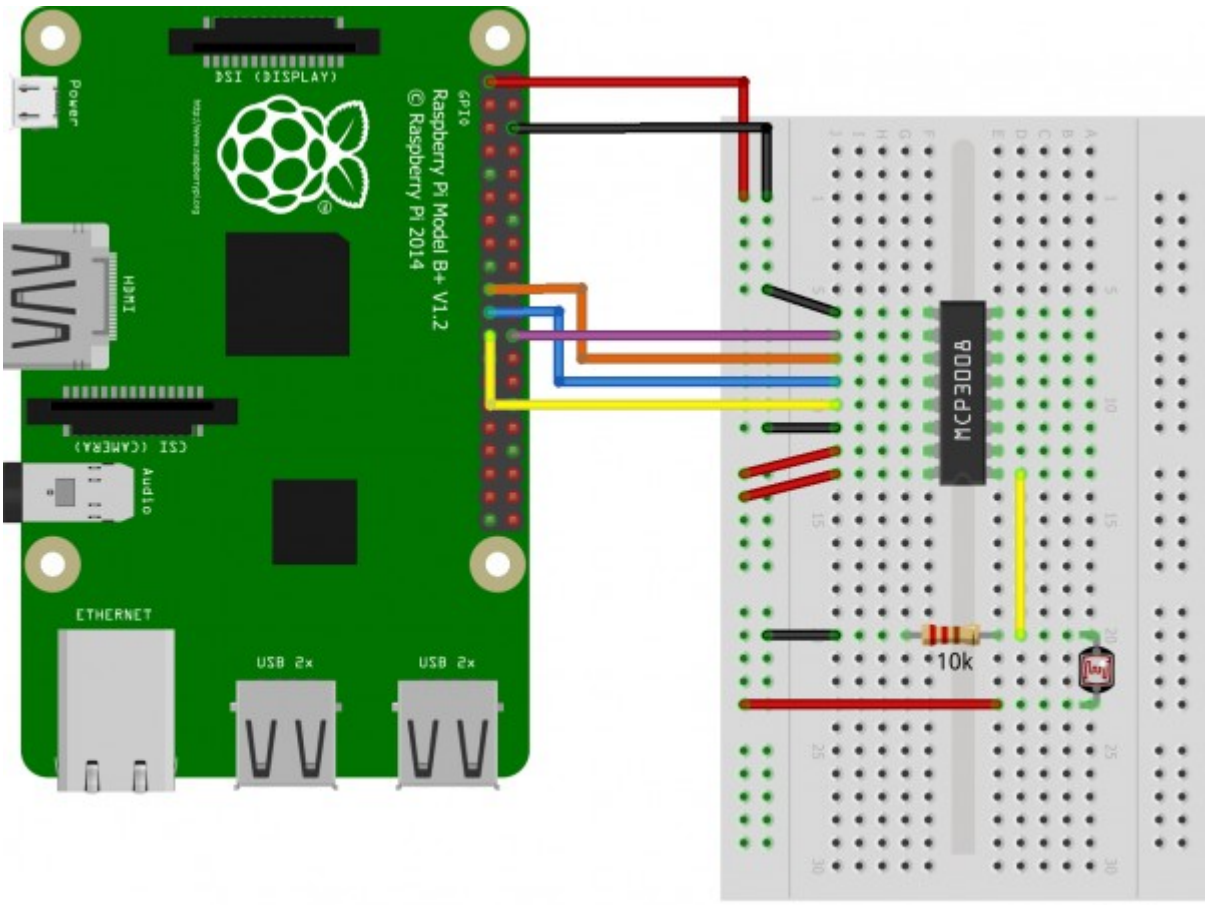
LED

Utilitzarem el GPIO6 com a sortida de control de la il·luminació. El GPIO6 anirà connectat a una pota d'una resistència de 1K. L'altra pota de la resistència anirà connectada a la pota positiva del LED (pota llarga). La pota curta (negativa) del LED anirà connectada a terra.

LDR

El fotoresistor tindrà una pota connectada a terra i l'altra connectada a la vegada a una de les potes d'una resistència de 10K i al canal 0 de l'MCP3008. L'altra pota de la resistència anirà connectada a 3,3V. Les sortides de l'MCP3008 aniran connectades així:

- Sortida 0: canal 0: connectada a una de les potes de l'LDR i de la resistència de 10K. L'LDR i la resistència formen un divisor de tensió.
- Sortida 9: Digital Ground: GND
- Sortida 10: CS/SHDN: SPICS0
- Sortida 11: Digital In: SPIMOSI
- Sortida 12: Digital Out: SPIMISO
- Sortida 13: Clock: SPISCLK
- Sortida 14: Analogue Ground: GND
- Sortida 15: V ref: 3,3V
- Sortida 16: Vdd: 3,3V



Il·lustració 3: Exemple de circuit LDR aplicat al laboratori

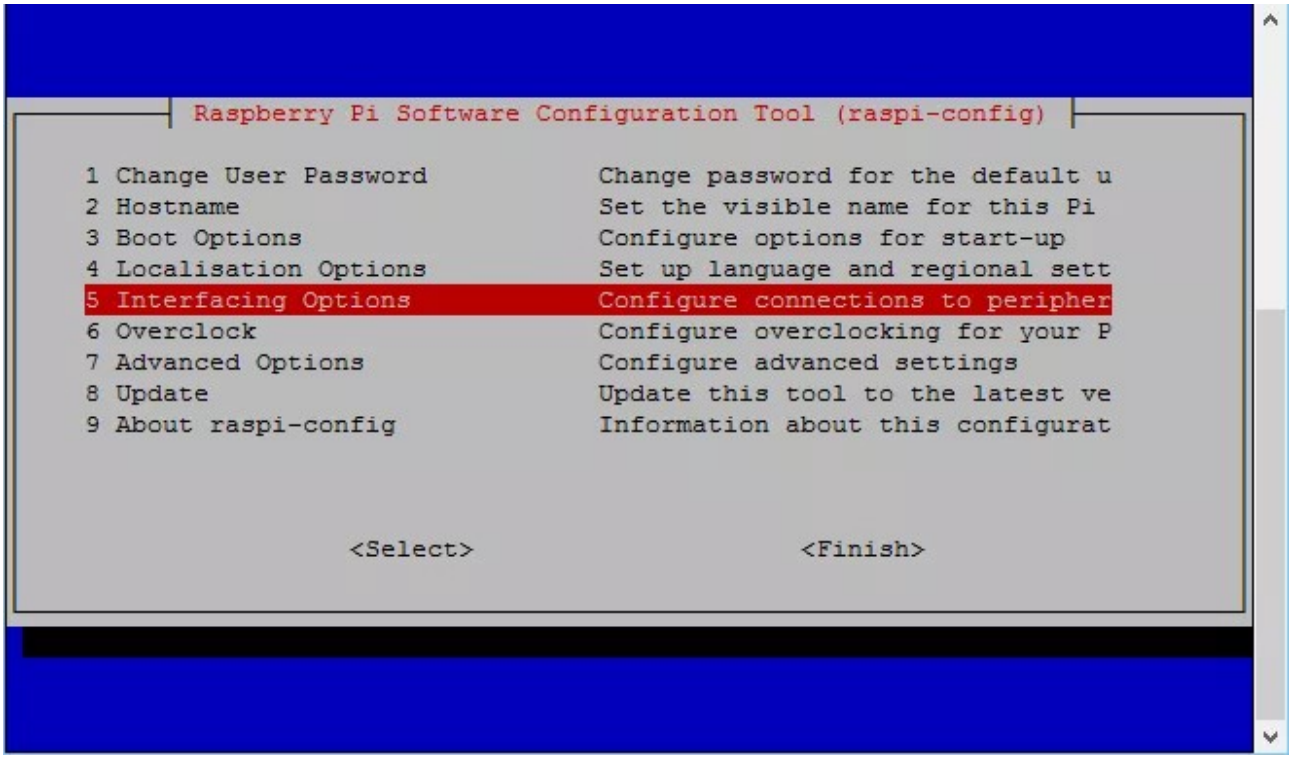
Software

Requisits

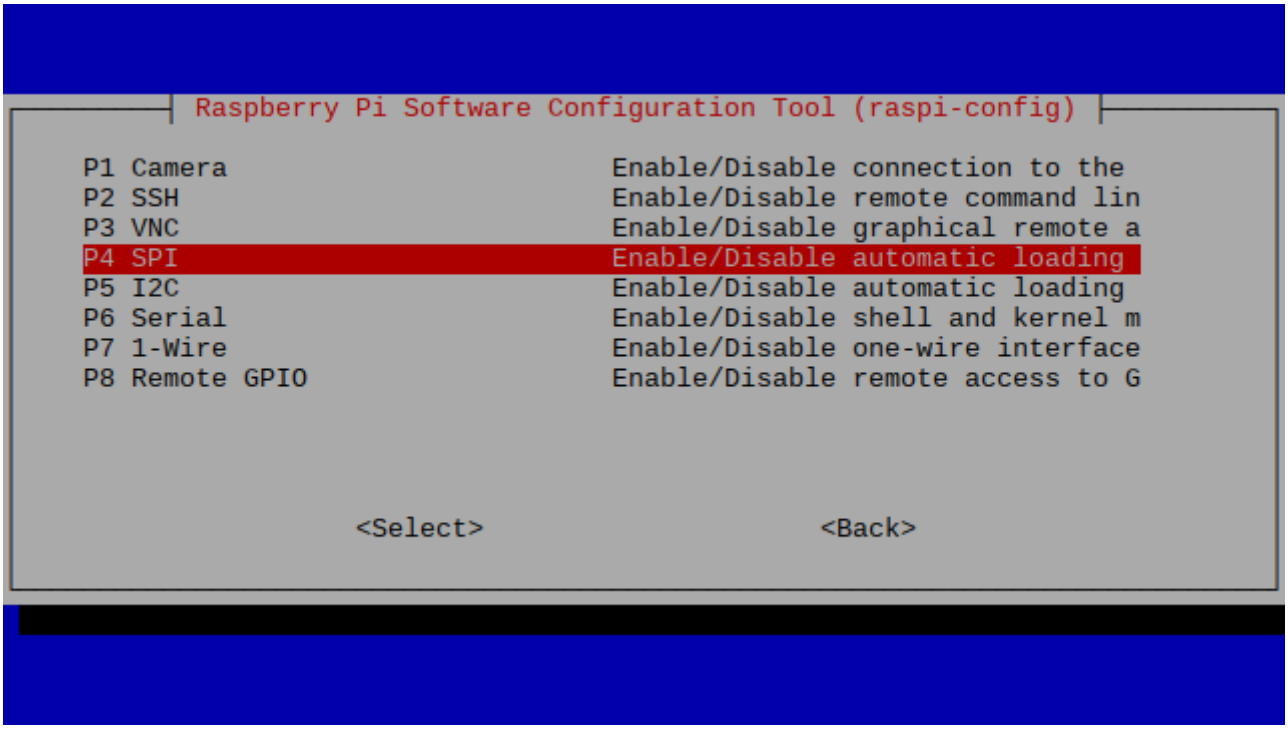
SPI

Per tal de poder utilitzar l'ADC MCP3008 connectat als ports SPI cal habilitar el suport SPI. Podem fer-ho utilitzant raspi-config:

```
sudo raspi-config
```

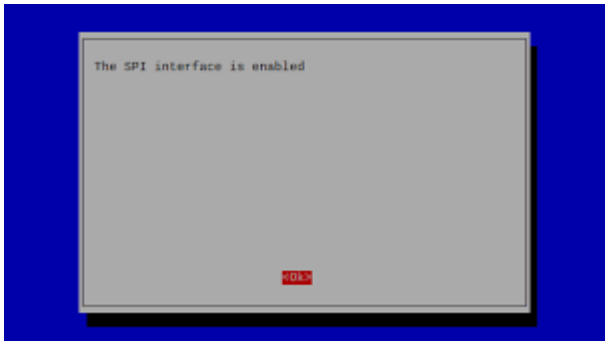


Il·lustració 4: Seleccionar Interfacing Options a raspi-config



Il·lustració 5: Seleccionar SPI a Interfacing Options de raspi-config

Il·lustració 6: Seleccionar Sí per a habilitar SPI



Il·lustració 7: Confirmació que s'ha habilitat SPI

PIP

PIP és un gestor de paquets per a Python, que és el llenguatge de programació utilitzat. Utilitzarem PIP per a gestionar els paquets de les llibreries que utilitzarem. Caldrà doncs instal·lar PIP:

```
sudo apt install python3-pip
```

RPi.GPIO

Aquesta llibreria ens permet accedir als diferents pins GPIO de la Raspberry de manera senzilla. La instal·larem amb el següent comandament:

```
pip install RPi.GPIO
```

Python SPI wrapper

```
sudo apt install python3-dev
sudo apt install python3-spidev
```

Adafruit MCP3008

Llibreria específica per a l'ADC MCP3008 d'Adafruit

```
pip install adafruit-mcp3008
```

Codi principal

```
import RPi.GPIO as GPIO
import Adafruit_GPIO.SPI as SPI
import Adafruit_MCP3008
import logging
import logging.handlers
import datetime
import time
from threading import Timer
import json
from sendemail import sendEmail

GPIO_DOOR_SENSOR = 5
GPIO_LIGHT_ACTUATOR = 6
LIGHT_VALUE_THRESHOLD = 300
LIGHT_UPTIME = 1.5 * 60 #1:30 min
SPI_PORT = 0
SPI_DEVICE = 0
SPI_ADC_CHANNEL = 0

def turn_light_off():
    if logger.isEnabledFor(logging.DEBUG):
        logger.debug('Turning light off')
    GPIO.output(GPIO_LIGHT_ACTUATOR, GPIO.LOW)

def timeFromString(string):
    values = string.split(':')
    return datetime.time(int(values[0]), int(values[1]))

waiting_threads = []

try:
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(GPIO_DOOR_SENSOR, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
    GPIO.setup(GPIO_LIGHT_ACTUATOR, GPIO.OUT)
```

```
    GPIO.output(GPIO_LIGHT_ACTUATOR, GPIO.LOW) #start with light off
    old_value = GPIO.HIGH #start with door closed
    mcp = Adafruit_MCP3008.MCP3008(spi = SPI.SpiDev(SPI_PORT, SPI_DEVICE))
    securityLogger = logging.getLogger('security')
    securityLoggerHandler =
logging.handlers.TimedRotatingFileHandler('security.log', 'W6', backupCount=24,
atTime = datetime.time())
    securityLoggerHandler.setFormatter(logging.Formatter('%(asctime)s - %
(levelname)s - %(message)s'))
    securityLogger.addHandler(securityLoggerHandler)
    securityLogger.setLevel(logging.INFO)
    logger = logging.getLogger(__name__)
    logger.addHandler(logging.handlers.RotatingFileHandler('iot2019.log',
maxBytes = 1024 * 1024, backupCount = 10))
    with open('schedule.json', 'r') as file:
        schedule_json = json.load(file)
    while True:
        value = GPIO.input(GPIO_DOOR_SENSOR)
        if value != old_value:
            if logger.isEnabledFor(logging.DEBUG):
                logger.debug('Value changed to ' + str(value))
            if value:
                securityLogger.info('Door closed')
                thread = Timer(LIGHT_UPTIME, turn_light_off)
                thread.daemon = True
                waiting_threads.append(thread)
                thread.start()
                if logger.isEnabledFor(logging.DEBUG):
                    logger.debug('Started thread ' + str(thread.ident))
            else:
                securityLogger.info('Door opened!')
                ldr_value = mcp.read_adc(SPI_ADC_CHANNEL)
                if logger.isEnabledFor(logging.DEBUG):
                    logger.debug('LDR value: ' + str(ldr_value))
                if ldr_value < LIGHT_VALUE_THRESHOLD:
                    for thread in waiting_threads:
                        thread.cancel()
                        if logger.isEnabledFor(logging.DEBUG):
                            logger.debug('killing thread ' + str(thread.ident))
                    waiting_threads = []
                    if logger.isEnabledFor(logging.DEBUG):
                        logger.debug('Turning light on')
                    GPIO.output(GPIO_LIGHT_ACTUATOR, GPIO.HIGH)
                breached = False
                for entry in schedule_json:
                    date_time = datetime.datetime
                    weekday = datetime.date.today().weekday()
                    currentTime = datetime.datetime.now().time()
                    weekdays = ["monday", "tuesday", "wednesday", "thursday",
"friday", "saturday", "sunday"]
                    if (weekday in entry['days'] or weekdays[weekday] in
entry['days']) and currentTime >= timeFromString(entry['startTime']) and
currentTime <= timeFromString(entry['endTime']):
                        breached = True
                        break
                if breached:
                    if logger.isEnabledFor(logging.DEBUG):
                        logger.debug('Alarm!!!')
                    sendEmail('Security breach: door opened!', 'Security breach:
door opened at ' + str(datetime.datetime.now()))
            old_value = value
```

```
except KeyboardInterrupt:
    logger.error('Interrupted by keyboard')
except:
    logger.exception('Undefined error!')
finally:
    GPIO.cleanup()
```

Codi per a enviar correus

L'enviament de correu es fa a través d'un relay SMTP. Hi ha diverses empreses que ofereixen aquest servei de manera gratuïta. En el laboratori s'ha utilitzat el servei de Mailjet.

```
import smtplib
from smtplib import SMTP
import json

with open('smtp.json', 'r') as file:
    smtp_json = json.load(file)

def sendEmail(subject, msg):
    # create server
    server = smtplib.SMTP(smtp_json['server'])
    server.starttls()

    # Login Credentials for sending the mail
    server.login(smtp_json['username'], smtp_json['password'])

    # send the message via the server.
    server.sendmail(smtp_json['from'], smtp_json['to'], "From: " +
smtp_json['from'] + "\r\nTo: " + smtp_json['to'] + "\r\nSubject: " + subject +
"\r\n" + msg)
    server.quit()
```

Les dades fixes (servidor, usuari, password, emissor i destinatari) es llegeixen d'un fitxer de configuració, mentre que l'assumpte i el missatge es proporcionen des del programa principal.

Configuració del servidor de correu

La configuració per al relay de correu es troba al fitxer smtp.json. Mostrem a continuació un exemple del contingut d'aquest fitxer:

```
{
      "server": "in-v3.mailjet.com:587",
      "username": "usuari proporcionat pel proveïdor",
      "password": "password proporcionat pel proveïdor",
      "from": "mailfrom@gmail.com",
      "to": "mailto@gmail.com"
}
```

Configuració de l'horari d'alarma

Anomenarem horari d'alarma l'horari durant el qual s'enviaran notificacions quan s'obri la porta. Aquest horari es configurarà al fitxer schedule.json. Presentem a continuació un exemple senzill del contingut d'aquest fitxer:

```
[
```

```
    {
        "days":
        [
            "monday",
            "tuesday",
            "wednesday",
            "thursday",
            "friday"
        ],
        "startTime": "00:00",
        "endTime": "23:00"
    }
]
```

Amb aquest exemple s'enviarien notificacions sempre que la porta s'obrís de dilluns a divendres de 0 a 23. És a dir, que les hores en les que es permetria obrir la porta (no s'enviarien notificacions) serien de 23 a 23:59 de dilluns a divendres i tot el cap de setmana.

Conclusions

Treball futur

LDR

Caldria realitzar la implementació del muntatge alternatiu proposat pel circuit de captació de llum, que és més senzill que amb l'ADC.

Relay

Caldria reemplaçar el LED per un relay i una bombeta real treballant a 220V

Millora del codi

Caldria millorar el codi amb l'ajuda d'analitzadors estàtics per a fer-lo complir amb els distints estàndards aplicables a Python

Una altra possible millora del codi podria ser fer-lo orientat a objectes.

GUI per a configuració

Es podria implementar una interfície gràfica que permetés modificar de manera senzilla els fitxers de configuració, per tal que fos més senzill per a l'usuari modificar la configuració del programa.

També caldria extreure les constants de llindar de lluminositat i temps que es manté el llum encès a un fitxer de configuració que fos fàcilment editable.

Altres opcions

El projecte ofereix nombroses opcions de personalització en molts sentits.

En el camp de les alarmes, es podria ampliar el projecte per a enviar SMSs, o fins i tot notificacions push o missatges de missatgeria instantània (WhatsApp, Telegram, LINE, ...). També es podrien

controlar diverses portes a la vegada i reaccionar en funció de cada una d'elles, de la seqüència en la que s'obren, etc.

En el camp de la il·luminació, la captació dels nivells de lluminositat i el control de l'encesa i apagada de lluminàries té molts usos. Tant en l'àmbit domèstic com industrial pot ser molt interessant ajustar la il·luminació a la necessitat de llum en funció de la falta d'aquesta i del moment del dia o fins i tot de la presència de persones (es podria afegir un sensor de distància per a detectar quan una persona entra o surt d'una estança).

Problemes trobats durant el desenvolupament

GPIO.wait_for_edge

Al principi del desenvolupament es va voler utilitzar la funció wait_for_edge de la llibreria RPi.GPIO, per tal que la CPU no estigui ocupada monitorant el GPIO corresponent a la porta. Aquesta funció posa el programa en espera i el desperta quan hi ha un canvi en el pin que se li indica. Empíricament, però, vam veure que en despertar moltes vegades el valor del pin que es llegia no era el valor nou. És per això que per tal d'evitar errors es va decidir fer una espera activa llegint tota l'estona el valor del pin i comparant amb el valor anterior. D'aquesta manera detectem quan el valor canvia i llavors actuem.

Crec que seria bo poder revisar aquesta funció per tal de poder trobar el motiu pel qual no funciona correctament en tots els casos i resoldre-ho.

Per altra banda es va provar també la funció add_event_detect de la mateixa llibreria, però aquesta no permetia definir callbacks diferents per al cas de RISING i FALLING, és a dir, que no permetia executar diferents funcions per a diferents esdeveniments per a un mateix pin. Aquesta funcionalitat seria molt interessant per a simplificar el funcionament del programa principal.

Gmail

La idea inicial era utilitzar Gmail per a enviar correus. Però per a poder enviar correus via SMTP cal habilitar la compatibilitat per a aplicacions menys segures, la qual cosa és un problema de seguretat, sobretot si fem servir aquell compte de correu per a altres usos. Per a poder utilitzar Gmail sense habilitar la compatibilitat amb aplicacions menys segures cal utilitzar la API de Gmail, que dóna accés a totes les funcionalitats del correu (la qual cosa és un problema de seguretat, ja que tenim accés a moltes més coses de les necessàries). Addicionalment, la guia que ofereix Google sobre aquesta API està desactualitzada i incompleta (almenys en Python), de manera que no és senzill aconseguir fer funcionar la API amb Python.

Per tots aquests motius es va decidir buscar una alternativa. Com a alternativa es van buscara eines de publitramesa. Es va avaluar SendGrid, que sembla que és la més coneguda, i recentment ha estat adquirida per Twilio. Malgrat això, no va funcionar bé la funcionalitat d'alta a aquesta plataforma, per la qual cosa es van buscar altres alternatives. Es van avaluar Mailgun i Mailjet i em vaig decantar per Mailjet perquè la política de preus en el pla gratuït em va semblar més clara.

Un cop realitzada l'alta a Mailjet la configuració del relay SMTP va ser molt senzilla, donant lloc al codi d'enviament de correus que s'ha mostrat abans.

SPI

Per a la correcta lectura de dades a través de SPI és necessari habilitar SPI a la configuració de la Raspberry i instal·lar les llibreries que es comenten a l'apartat de requisits. En cas contrari la lectura de l'ADC no funcionarà correctament i no obtindrem cap error que ens orienti fàcilment a la solució.

Bibliografia web

1. LDR circuit diagram (pàg. 3): https://www.build-electronic-circuits.com/ldr-circuit-diagram/
2. Raspberry photoresistor schematic (pàg. 4): https://tutorials-raspberrypi.com/photoresistor-brightness-light-sensor-with-raspberry-pi/
3. Enabling SPI on Raspberry (pàg. 7-8): https://www.raspberrypi-spy.co.uk/2014/08/enabling-the-spi-interface-on-the-raspberry-pi/
4. RPi.GPIO documentation: https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/
5. Adafruit MCP3008: https://github.com/adafruit/Adafruit_Python_MCP3008
6. Official Python documentation: docs.python.org