Šolski center Celje, Srednja šola za kemijo, elektrotehniko in računalništvo

Pametna garažna vrata

raziskovalna naloga

Avtor
Boštjan Planko, R-4.A

Mentor
Borut Slemenšek, univ. dipl. inž

Mestna občina Celje, Mladi za Celje
Celje, marec 2019

Povzetek

Raziskoval sem področje elektronike in avtomizacije. Glavno vprašanje razsikovalne naloge je bilo ali je mogoče avtomatizirati garažna vrata z Raspberry Pi računalnikom. Potrebno znanje, ki ga še nisem imel, sem pridobil z metodo analize dokumentov in metodo eksperimenta. Tekom nastajanja projekta sem prišel do zaključka, da je osnovna ideja sicer enostavna, v praksi pa stvari zelo hitro postanejo zelo kompleksne. Rezultat moje raziskovalne naloge je delujoč prototip, na katerem delujejo vse osnovne funkcije. Je dober temelj za nadaljnji razvoj in širitev projekta.

Summary

I was researching the field of electronics and automation. The main goal of this paper was finding out if it is possible to automate garage doors using Raspberry Pi. I had to analyze a lot of documentation to get the knowledge that was required to finish my project. I also had to experiment quite a lot. In the end, I came to the conclusion that the basic idea for my project is quite simple. Despite that, things can get really complicated really fast. The end result of my work is a working prototype that has a lot of possibilities to expand the project even further.

Ključne besede

raspberry pi, senzorji, python, raspbian, avtomizacija

Kratice

GPIO General-purpose input/output. 9, 11

LCD Liquid Crystal Display. 7, 8, 10, 13, 15

SSH Secure Shell. 7, 8

Kazalo

Kazalo slik

1 Uvod

1.1 Opis problema

Problem današnjih garažnih vrat je, da jih je običajno možno odpreti samo na dva načina: s priloženim daljincem ali s tipko, običajno nameščeno na notranji strani garažnih vrat. Če želimo garažna vrata odpreti moramo torej biti v garaži ali pa moramo imeti pri sebi daljinec. To pa je v vsakdanjem življenju nepraktično, sploh v primeru, ko pri hiši živi veliko ljudi, vsi pa rabijo dostop do garaže.

V tej nalogi bom predstavil svojo idejo, pametna garažna vrata, kot sem si jih zamislil in poskušal realizirati.

1.2 Hipoteze

Za raziskovalno nalogo sem si postavil naslednje hipoteze:

1. Garažna vrata bo možno upravljati s telefonom.
2. Garažna vrata bo možno upravljati preko spletne strani.
3. Raspberry Pi bo spremljal, ali je avto v garaži ali ne, in glede na to samodejno zapiral garažna vrata.
4. Raspberry Pi bo spremljal temperaturo v garaži in jih samodejno zaprl v primeru prenizke ali previsoke temperature.
5. Raspberry Pi bo samodejno zaprl garažna vrata, če ostanejo odprta po določeni uri.
6. Raspberry Pi nas bo s potisnimi obvestili obveščal o spremembi stanja garažnih vrat.
7. Raspberry Pi bo beležil, kdo in kdaj je aktiviral garažna vrata.
8. Uporabnik bo imel možnost preklicati samodejno zapiranje vrat.

2 Raziskovalne metode

Med raziskovanjem sem uporabil kombinacijo različnih metod, saj sem tako najlaže prišel do rešitve problema.

Na začetku raziskovanja sem uporabljal predvsem metodo analize dokumentov. Prebral in preučil sem veliko dokumentacije in objav na forumih. Tako sem pridobil osnovno podlago za začetek dela na svojem projektu.

Pri programiranju in sestavljanju prototipa sem uporabil predvsem metodo eksperimenta. Preizkusil sem veliko različnih konfiguracij in programskih rešitev, na koncu pa sem izbral tiste, ki so se mi zdele najbolj optimalne.

3 Izbira komponent

Ker ne potrebujem veliko precesorske moči, hkrati pa želim, da je moj projekt kar se da kompakten sem za krmilnik izberal Raspberry Pi Zero W. To je najmanjša verzija Raspberry Pi-ja z že vgrajenim WiFi-jem in Bluetoothom. Slednja bosta pri projektu najverjetneje potrebna.

Za upravljanje garažnih vrat bom uporabil 1-kanalni rele. Tega bom sprogramiral tako, da se bo obnašal kot tipka, tj. zaprl se bo za kratek časovni interval, prib. 0.5 s, nato pa se znova odprl. Nameščen bo v bližini že obstoječe tipke, ki se uporablja za upravljanje garažnih vrat. S to bo vezan vzporedno.

Za spremljanje stanja garažnih vrat bom uporabil reed stikala, in sicer dve stikali in magnet. Stikali bosta nameščeni na ogrodje vrat, medtem ko bo magnet nameščen neposredno na garažna vrata.

Za spremljanje temperature v garaži bom uporabil 1-Wire digitalni element DS18B20. Ta bo nameščen nekje v garaži, po možnosti meter od tal, na najmanj prepišnem mestu.

Ultrazvočni senzor, s katerim bom preverjal je avto v garaži ali ne, bo nameščen ali na stropu garaže, najverjetneje kar na motorju garažnih vrat.

Ker želim, da bo mogoče v garaži preveriti trenutno temperaturo in čas, bom uporabil tudi 16x2 Liquid Crystal Display (LCD) zaslon.

Poleg že naštetih komponent bo uporabil še dve LED diodi in dve tipki. Te bodo uporabljene paroma kot indikator stanja avta oziroma temperature v garaži. Če bo na primer garaža odprta in bo vanjo pripeljal avto, se bo pognal program, ki bo po določenem času samodejno zaprl vrata. Istočasno bo začela utripati ustrezna LED dioda, uporabnik pa bo imel s pritiskom tipke možnost, da prekliče samodejno zapiranje garaže. Pri temperaturi je namen LED diode in tipke enak, le da z njima spremljamo temperaturo v garaži.

4 Priprava Raspberry Pi-ja

Da bom lahko uporabljal Raspberry Pi, moram najprej naložiti usterzen opreacijski sistem. Ker za svoj projekt ne potrebujem grafičnega vmesnika, na Raspberry Pi namestim Raspbian Lite. To storim tako, da iz uradne strani Raspberry Pi [4] prenesem Raspbian Stretch Lite. Nato sledim navodilom za namestitev [3] operacijskega sistema na microSd kartico, ki jo nato vstavim v Raspberry Pi.

Ker do Raspberry Pi-ja že od samega začetka nimam dostopa preko tipkovnice, moram pred zagonom omogočiti še Secure Shell (SSH) ter vnesti podatke, ki jih Raspberry Pi potrebuje za povezavo na WiFi dostopno točko. Da omogočim SSH, na boot particijo microSD katrice dodam datoteko ssh. Da pa se bo Raspberry Pi lahko povezal na WiFi dostopno točko, moram na boot particiji ustvariti datoteko wpa_supplicant.conf, v katero vnesem naslednje:

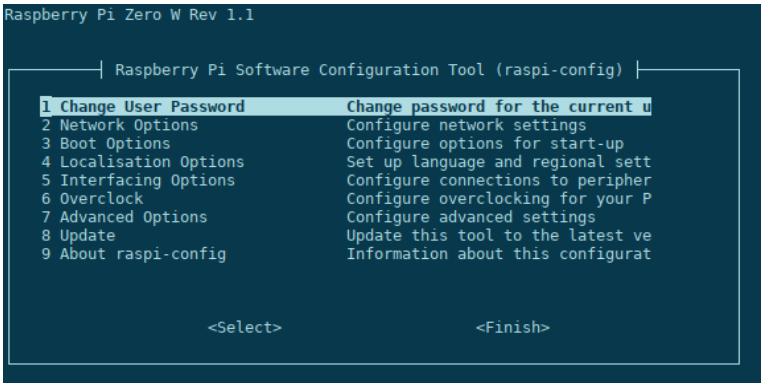
```
country=SI
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
^^Issid="imeDostopneTpočke"
^^Ipsk="geslo"
^^Ikey_mgmt=WPA-PSK
}
```

Nato microSD kartico vstavim v Raspberry Pi in priključim napajanje.

Nato se iz terminala na svojem računalniku preko SSH povežem na Raspberry Pi:

```
ssh pi@IP_RaspberryPi #uporabnik pi, privzeto geslo pa je raspberry
```

Ko je povezava vzpostavljena, uporabim ukaz *passwd pi*, da spremenim geslo uporabnika pi. Nato zaženem ukaz *sudo raspi-config*



Slika 1: Raspi-config: Glavni meni.

Pojavi se meni, po katerem se premikam s pomočjo tipkovnice.
Tukaj nastavim vse potrebno:

- hostname spremenim iz ”raspberrypi” v ”projectPi”,
- časovno območje nastavim na Europe/Ljubljana,
- omogočim SPI, I2C in 1-Wire (vse to bom potreboval za priklop senzorjev),
- razširim datotečni sistem.

Preden namestim nove pakete, je potrebno sistem posodobiti:

```
sudo apt update && sudo apt upgrade -y
```

Ker bom program za upravljanje garažnih vrat napisal v Pythonu, le tega namestim na Raspberry Pi:

```
sudo apt install python python-pip
```

Ker bom uporabil tudi LCD, moram namestiti še RPILCD knjižnico:

```
1 sudo pip install RPLCD
```

S tem je priprava Raspberry Pi-ja zaključena.

5 Priključitev komponent

5.1 Rele

Priklop releja je zelo enostaven. Modul z relejem ima tri priključke: VCC, GND in IN1. VCC priključek povežem na 5 V pin. GND priključim na GND pin. Za IN1 izberem BMC pin 16.

5.2 Reed stikali

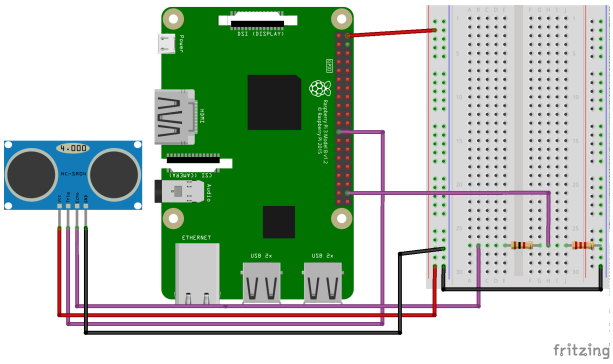
Priključitev reed stikal je v osnovi zelo enostavna. Paziti je treba, da imamo NO (Normally Open) in ne NC (Normally Closed) stikala. Nato en del stikala povežemo na 3.3 V, drugi del pa na enega izmed General-purpose input/output (GPIO) pinov. Prvo stikalo (garaža odprta) sem priklopil na BCM pin 5, drugega (garaža zaparta) pa na BCM pin 6.

Po priklopu, je treba paziti, da v programu ne pozabimo nastaviti pull-down uporov, oba pina pa morata biti nastavljena kot vhodna. Tako ima pin vrednost 1, če je stikalo zaprto in vrednost 0, če je stikalo odprto.

5.3 Ultrazvočni senzor razdalje HC-SR04

Ker sem ultrazvočni senzor razdalje na Raspberry Pi uporabljal prvič, sem si pomagal z vodičem na ModMyPi [6].

Senzor ima štiri pine: VCC, GND, Trig, Echo. VCC povežem na 5 V. GND povežem na GND pin, Trig na BCM pin 11, Echo pa najprej preko 1 kΩ upora na BCM pin 20, nato pa še preko 2 kΩ na GND.



Slika 2: Priklop HC-SR04 unltrazvočnega senzorja razdalje.

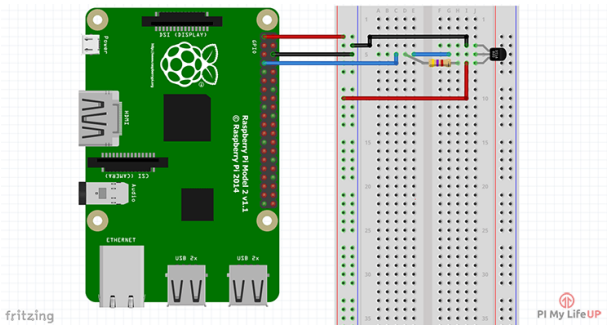
Da pridobim razdaljo, uporabim naslednjo metodo:

```
def checkCar():
    GPIO.output(GPIO_VARS_DICT['TRIG'], False)
    time.sleep(0.001)

    GPIO.output(GPIO_VARS_DICT['TRIG'], True)
    time.sleep(0.00001)
    GPIO.output(GPIO_VARS_DICT['TRIG'], False)

    while GPIO.input(GPIO_VARS_DICT['ECHO'])==0:
      pulse_start = time.time()

    while GPIO.input(GPIO_VARS_DICT['ECHO'])==1:
      pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150

    return round(distance, 2)
```

5.4 DS18B20 temperaturni senzor

Ker sem v projektu prvič uporabil DS18B20 senzor temperature, sem si pomagal z vodičem na PiMyLifeUp [5].

Priklop je povsem enostaven. Vse kar potrebujemo, je DS18B20 senzor temperature in pa 4.74.7 kΩ upor. Vcc nogico senzorja sem povezal na 3.3 V. GND nogico na GND, DATA nogico pa na BCM pin 19. Upor namestim med 3.3 V in DATA nogico senzorja.



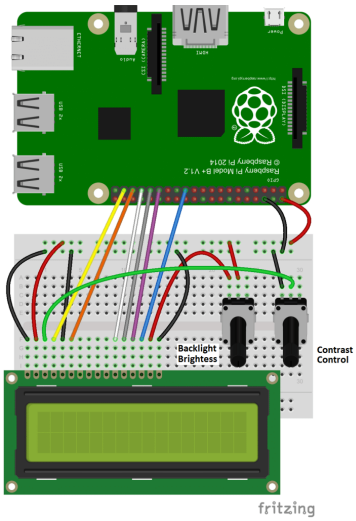
Slika 3: Priklop DS18B20.
Vir: https://cdn.pimylifeup.com/wp-content/uploads/2016/03/Raspberry-Pi-Temperature-Sensor-Diagram-v2.png

5.5 LCD zaslon

Tudi priklop LCD zaslona je bil zame nekaj novega. Zato sem si pomagal z vodičem na circuitbasics.com [1]. Za priklop sem potreboval:

- LCD zaslon,
- 2 10 kΩ potenciometra.

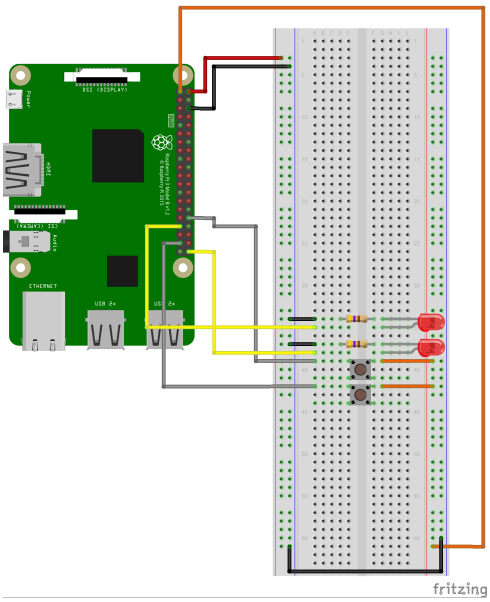
LCD zaslon sem priklopil na Raspberry Pi, kot je prikazano na spodnji sliki.



Slika 4: Priklop LCD zaslona.
Vir: http://www.circuitbasics.com/wp-content/uploads/2015/04/Raspberry-Pi-LCD-4-bit-mode-719x1024.png

5.6 Tipki in LED diodi

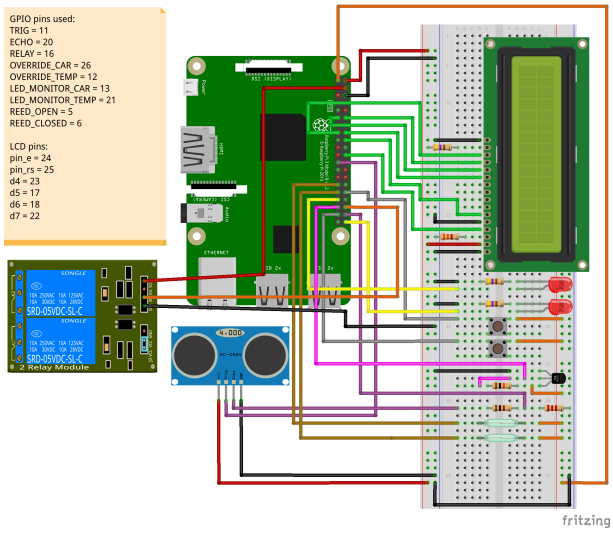
Priklop tipk in LED diod je zelo enostaven. Tipki povežem preko 3.3 V na poljuben GPIO pin, pri čemur pazim le, da je pin nastavljen kot vhodni in da je omogočen notranji pull down upor. LED diodi sem priklopil preko 270 Ω uporov iz poljubnega GPIO pina na GND. GPIO pin mora biti nastavljen kot izhodni.



Slika 5: Priklop LED diod in tipk.

5.7 Vse komponente

Ko sem priključil vsako komponento posebej in se prepričal o tem kako deluje, sem vse skupaj združil v en sistem.
Diagram se od dejanske izvedbe razlikuje le v tem da je tukaj uporabljen dvo kanalni modul relejev, ker je bil edini primeren za uporabo v diagramu. Ne glede na to je priklopljen samo en kanal.



Slika 6: Priklop vseh komponent.

6 Programske rešitve

Celotna koda vseh programov, omenjenih v raziskovalni nalogi, je v prilogi.

6.1 Premikanje garažnih vrat

Premikanje garažnih vrat je zelo enostavno. Vse, kar mora program storiti, da odpre garažna vrata, je preklop releja. Rele se tako preklopi iz odprtega stanja v zaprto, nato pa se čez pol sekunde vrne v odprto stanje. Za to poskrbi spodnja metoda.

```
1 def toggleGarage():
2     GPIO.output(GPIO_VARS_DICT['RELAY'], 0)
3     time.sleep(.5)
4     GPIO.output(GPIO_VARS_DICT['RELAY'], 1)
```

6.2 Preveranje stanja garažnih vrat

Preverjanje stanja garažnih vrat poteka z uporabo dveh reed stikal. Eno je nameščeno na spodnjem delu vodila garažnih vrat, kjer se vrata nahajajo ko so zaprta, drugo pa je nameščeno na zgornjem delu nosilca, kjer se vrata nahajajo, ko so odprta. Magnet, ki sproži stikali, je nameščen neposredno na garažna vrata.

Določanje, ali so vrata odprta, zaprta ali priprta, poteka s preverjanjem stanja stikal. Če je aktivno spodnje stikalo, pomeni, da so garažna vrata zaprta. Če je aktivno zgornje stikalo, pomeni, da so vrata odprta. Če ni aktivno nobeno izmed stikal, so garažna vrata priprta.

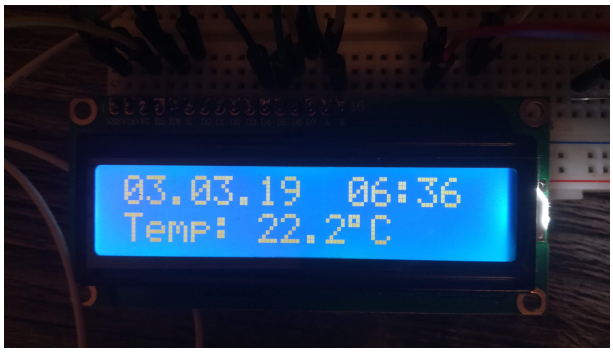
6.3 Spremljanje temperature

Spremljanje temperature poteka z uporabo DS18B20 temperaturnega senzorja. Ta je nameščen meter od tal na najmanj prepišnem delu garaže. Program začne spremljati temperaturo, ko se garažna vrata odprejo. Med spremljanjem temperature urtipa kontrolna LED dioda. Tako uporabnik ve, da program spremlja temperaturo v garaži. Če temperatura pade oziroma preseže nastavljeno mejo, program samodejno zapre vrata in o tem obvesti uporabnike. V primeru, da uporabnik ne želi, da program v primeru neprimerne temperature samodejno zapre vrata , lahko to prekliče s pritiskom tipke.

6.4 Izpisovanje temperature na LCD

LCD, ki bo nameščen v ohišju na vidnem mestu v garaži, bo uporabljen za izpisovanje trenutega datuma, časa in temperature.

Za izpisovanje se uporablja poseben program, ki v neskončni zanki meri temperaturo ter jo skupaj s trenutnim datumem in časom izpisuje na LCD zaslon.



Slika 7: Izpis na LCD.

6.5 Samodejno zapiranje glede na avto

Ultrazvočni senzor razdalje HC-SR04 se uporablja za preverjanje, ali je avto v garaži ali ne. Če na primer avta ni v garaži, nato pa se garaža odpre in vanjo pripelje avto, bo program po določenem času samodejno zaprl vrata in o tem obvestil uporabnike preko potisnih sporočil. Enako se zgodi, če avto zapusti garažo. V primeru, da uporabnik ne želi, da se ob prihodu ali odhodu avta iz garaže vrata samodejno zaprejo, lahko ukaz prekliče s pritiskom tipke.

Senzor je nameščen na stropu garaže. Razdalja (med senzorjem in avtom), ki določa, kdaj senzor zazna, da je v garaži avto, se nastavi v konfiguracijski datoteki.

6.6 Če garažnih vrat ni možno zapreti

Med testiranjem programa za garažna vrata sem kmalu odkril relativno pomembno napako. Dogajalo se je, da je program želel vrata zapreti, to pa zaradi npr. ovire pod vrati ni bilo mogoče. Ker program ni bil dobro napisan, je v nedogled poskušal brez uspeha zapreti vrata.

Težavo sem rešil tako, da sem napisal metodo v katero program vstopi, če se vrata po poskusu zapiranja ne zaprejo. Metoda nato še nekajkrat poskuša zapreti vrata. Če to ni mogoče, vrata ostanejo priprta.

```
1 def doorAjar():
2     for attempts in range(0, TIMEOUTS_VARS_DICT['AJAR_CLOSE_ATTEMPTS']):
3         toggleGarage()
4         for x in range(0, TIMEOUTS_VARS_DICT['AJAR_TIMEOUT']):
5             if checkDoor() != 'priprta':
6                 break
7             time.sleep(1)
8         if checkDoor() == 'zaprta':
9             break
```

6.7 Konfiguracijska datoteka

Za lažje nastavljanje spremenljivk, kot je na primer termperatura, pri kateri se garažna vrata zaprejo, sem se odločil uporabiti konifugacijsko datoteko.

Ker te do sedaj še nikoli nisem uporabljal, mi je bila uradna Pythonova dokumentacija [8] v veliko pomoč.

6.7.1 Datoteka

Zgradba konfiguracijske datoteke je zelo preprosta. V oglatih oklepajih se nahaja ime odseka, sledijo imena spremenljivk, katerih vrednost je določena z enačajem, ki mu sledi vrednost.

```
1 [gpio]
2 TRIG = 11
3 ECHO = 20
4 RELAY = 16
```

6.7.2 Branje iz datoteke

Branje iz datoteke je bilo na začetku kar precejšen izziv. Po nekaj različnih poskusih implementacije sem se določil da bom vrednosti iz konfiguracijske datoteke shranjeval v slovar. Za branje iz datoteke sem napisal svojo metodo.

Ta prejme del, iz katerega mora brati, spremenljivke, ki jih je potrebo prebrati in slovar v katerega se bodo shranile vrednosti. Nato odpre datoteko za branje, nato pa v zanki prebere vrednosti spremenljivk in jih shrani v slovar.

6.8 Potisna obvestila

Ker je potrebno uporabnike obveščati o spremebi stanja garažnih vrat, sem moral poiskati način, kako to storiti. S pomočjo raziskovanja sem se odločil, da bom uporabil storitev za pošiljanje potisnih obvestil Pushover [9].

Storitev je zelo enostavna in se je do sedaj odlično obnesla. Edina slabost je, da je plačljiva. Treba je namreč kupiti licenco za vsako napravo, ki storitev uporablja. Cena licence je 4.99 $.

6.9 Možne razširitve projekta

Čeprav sem s projektom do sedaj zelo zadovoljen, sem prepričan, da je možnosti za izboljšave še veliko.

6.9.1 Sistem dnevniških datotek

Beleženje dogodkov programa je dobro že z vidika odpravljanja težav. Trenutno je zelo težko določiti, kje v programu se pojavlja napaka. Zato bi bilo zelo koristno, če bi v program dodal zapisovanje dnevniški datotek in pa možnost razhroščevanja programa (debug).

Druga stvar, ki bi bila po mojem mnenju koristna, pa je beleženje kdo je garažo aktiviral, kdaj in s katere naprave. Iz tega bi lahko delal statistiko uporabe garažnih vrat, hkrati pa bi lahko zelo hitro ugotovil, če do garaže dostopa kdo, ki nima pooblastil.

6.9.2 Android aplikacija

Trenutno za upravljanje garaže preko telefona uporabljam kar Shell skripto, ki jo poženem z aplikacijo Termux. Zaradi boljše uporabniške izkušnje in uporabnosti bi bilo zato zelo dobro, če bi naredil še aplikacijo za Android. Aplikacija bi se lahko uporabljala za upravljanje garažnih vrat in spremljanje temperature. Lahko bi dodal tudi možnost, da uporabniki preko aplikacije nastavijo določene spremenljivke, kot je na primer ura, ob kateri se garaža samodejno zapre.

6.9.3 Brezžični moduli

Ko sem razmišljal, kako bi lahko svoj projekt še dodatno izboljšal, sem prišel na idejo, da bi bili moduli samostojni. Tako bi bil na primer en modul rele, drugi bi bil LCD zaslon, tretji senzor temperature itd. To bi bilo verjetno izvedljivo, če bi za vsak modul uporabil na primer NodeMcu [7]. Tako bi bil vsak modul zase povezan v WiFi omrežje, preko katerega bi komuniciral s centralnim računalnikom.

6.9.4 Odpiranje s prstnim odtisom ali številčnico

Kaj storiti v primeru, če želimo od zunaj odpreti garažo a pri sebi nimamo ne daljinca ne pametnega telefona? To težavo bi lahko enostavno rešili tako, da bi na zunanji strani vrat namestili čitalec prstnih odtisov ali številčnico. Tako bi lahko uporabniki odprli garažna vrata s prstnim odtisom ali z ustrezno kodo.

6.9.5 Spletna stran

Za spremljanje stanja garažnih vrat in njihovo upravljanje bi bilo dobro imeti tudi spletno stran. Poleg tega bi se lahko slednja uporabila (če bi se prijavil ustrezen uporabnik) za spremljanje dnevniških datotek in nastavljenje določenih parametrov programa.

7 Razprava

1. Prvo hipotezo: garažna vrata bo možno upravljati preko telefona, lahko delno potrdim, saj je uporavljanje vrat s telefonom možno, vendar le, če so na njem nameščene ustrezne aplikacije (Termux ali PiRelay).

2. Drugo hipotezo: garažna vrata bo možno upravljati preko spletne strani, ovržem, saj mi spletne strani ni uspelo narediti.

3. Tretjo hipotezo: Raspberry Pi bo spremljal ali je avto v garaži ali ne in glede na to samodejno zapiral garažna vrata, lahko potrdim, saj se garažna vrata samodejno zaprejo, če avto pripelje v garažo ali jo zapusti.

4. Tudi četrto hipotezo: Raspberry Pi bo spremljal temperaturo v garaži in jih samodejno zaprl v primeru prenizke ali previsoke temperature, lahko potrdim, saj se garaža ob neprimerni temperaturi samodejno zapre.

5. Peto hipotezo: Raspberry Pi bo samodejno zaprl garažna vrata, če ostanejo odprta po določeni uri, lahko potrdim, saj se garažna vrata ob določeni uri zaprejo samodejno.

6. Šesto hipotezo: Raspberry Pi nas bo preko potisnih obvestil obveščal o spremembi stanja garažnih vrat, lahko potrdim, saj Raspberry Pi pošilja potisna obvestila preko soritve Pushover.

7. Sedmo hipotezo: Raspberry Pi bo beležil kdo in kdaj je aktiviral garažna vrata, ovržem, saj Raspberry Pi ne beleži dnevnika uporabe garaže.

8. Osmo hipotezo: uporabnik bo imel možnost preklicati samodejno zapiranje vrat, lahko delno potrdim. Uporabnik ima možnost preklicati samodejno zapiranje vrat v primeru neustrezne temperature ali spremembe stanja avta. Ne more pa preklicati samodejnega zapiranja ob določeni uri.

7.1 Težave med delom

7.1.1 Rele predolgo ostane zaprt

Prva težava, ki se je pojavila pri izdelavi mojega projekta, je bilo, da je rele ostal predolgo zaprt. To je pomenilo, da se je rele zaprl, garažna vrataso se začela premikati, nato pa se ni več odprl. Ko se je to zgodilo, garažnih vrat ni bilo mogoče uporavljati s tipko.

Težavo sem rešil zelo preprosto. Napisal sem program (checkRelay.py), ki spremlja stanje releja.V priemru, da ta ostane zaprt predolgo, ga odpre, dogodek ap zabeleži v dnevniško datoteko (checkRelay.log).

7.1.2 Multithreading

Med izdelavo projekta sem se kmalu zavedel, da imam še eno težavo. Program mora hkrati spremljati temperaturo v garaži in stanje avta. Najprej sem razmišljal, da bom problem rešil z uporabo Multithreadinga. Tako sem prebral vodič [10]. Ker po nekaj neuspešnih poskusih še vedno nisem vedel kaj natanko moram storiti, sem začel iskati alternative.

Na koncu mi je težavo uspelo rešiti z uporabo multiprocessinga. Najprej sem prebral dokumentacijo [2], nato pa sem z nekaj eksperimentiranja težavo kmalu odpravil.

Izdelava raziskovalne naloge je zahtevala veliko truda in časa. Moral sem prebrati veliko dokumentacije, brskati po forumih, eksperimentirati. :e bi imel več časa, denarja in še kakšen dodaten par rok, pa bi lahko svoj projekt še precej razširil.

8 Zaključek

Moj cilj je bil narediti funkcionalen izdelek, ki se bo namestil v garaži in se uporabljal za avtomatizacijo garažnih vrat ter upravljanje le teh preko telefona. Čeprav nimam estetsko dovršenega in praktično uporabnega izdelka, pa imam delujoč prototip.

Med izdelavo raziskovalne naloge sem pridobil veliko novih znanj. Dobil sem tudi kar precej izkušenj pri delu z elektroniko, ki jih do sedaj nisem imel.

Projekt zame še zdaleč ni končan. Razvoj sistema bom nadaljeval kolikor bo čas dopuščal, seveda pa bom moral upoštevati tudi finančne omejitve. Odlično bi bilo tudi, če bi našel še koga, ki bi bil pripravljen sodelovati pri projektu. Kot vedno pravijo, več glav več ve.

Literatura

[1] Circuit Basics. How to Setup an LCD on the Raspberry Pi and Program It With Python. http://www.circuitbasics.com/raspberry-pi-lcd-set-up-and-programming-in-python/. Zadnji dostop: 26. 2. 2019.

[2] Python Software Foundatio. multiprocessing — Process-based "threading" interface. https://docs.python.org/2/library/multiprocessing.html. Zadnji dostop: 9. 3. 2019.

[3] Raspberry Pi Foundation. Installing operating system images. https://www.raspberrypi.org/documentation/installation/installing-images/README.md. Zadnji dostop: 27. 2. 2019.

[4] Raspberry Pi Foundation. Raspbian. https://www.raspberrypi.org/downloads/raspbian/. Zadnji dostop: 27. 2. 2019.

[5] Gus. Raspberry Pi Temperature Sensor: Build a DS18B20 Circuit. https://pimylifeup.com/raspberry-pi-temperature-sensor/. Zadnji dostop: 26. 2. 2019.

[6] ModMyPi LTD. HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi. https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi. Zadnji dostop: 15. 2. 2019.

[7] NodeMcu. NodeMcu. https://www.nodemcu.com/index_en.html. Zadnji dostop: 1. 3. 2019.

[8] Python. ConfigParser — Configuration file parser. https://docs.python.org/2/library/configparser.html. Zadnji dostop: 1. 3. 2019.

[9] Superblock. Pushover. https://pushover.net/. Zadnji dostop: 1. 3. 2019.

[10] tutorialspoint. Python - Multithreaded Programming. https://www.tutorialspoint.com/python/python_multithreading.htm. Zadnji dostop: 9. 3. 2019.

9 Zahvale

Zahvaljujem se svojemu mentorju, profesorju Borotu Slemenšku za nasvete ter pomoč pri izdelavi raziskovalne naloge.

10 Priloge

10.1 Izjava

IZJAVA*

Mentor Borut Slemenšek v skladu z 2. in 17. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, zagotavljam, da je v raziskovalni nalogi z naslovom Pametna garažna vrata, katere avtorica je Boštjan Planko:

- besedilo v tiskani in elektronski obliki istovetno,
- pri raziskovanju uporabljeno gradivo navedeno v seznamu uporabljene literature,
- da je za objavo fotografij v nalogi pridobljeno avtorjevo dovoljenje in je hranjeno v šolskem arhivu,
- da sme Osrednja knjižnica Celje objaviti raziskovalno nalogo v polnem besedilu na knjižničnih portalih z navedbo, da je raziskovalna naloga nastala v okviru projekta Mladi za Celje,
- da je raziskovalno nalogo dovoljeno uporabiti za izobraževalne in raziskovalne namene s povzemanjem misli, idej, konceptov oziroma besedil iz naloge ob upoštevanju avtorstva in korektnem citiranju,
- da smo seznanjeni z razpisni pogoji projekta Mladi za Celje.

Celje, 8.3.2019

ŠOLSKI CENTER CELJE SREDNJA ŠOLA ZA KEMIJO, ELEKTROTEHNIKO IN RAČUNALNIŠTVO CELJE, Pot na Lavo 22

Žig šole

Podpis mentorja

Podpis odgovorne osebe

*

POJASNILO

V skladu z 2. in 17. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje je potrebno podpisano izjavo mentorja (-ice) in odgovorne osebe šole vključiti v izvod za knjižnico, dovoljenje za objavo avtorja (-ice) fotografskega gradiva, katerega ni avtor (-ica) raziskovalne naloge, pa hrani šola v svojem arhivu.

10.2 garage.py - glavni program

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import RPi.GPIO as GPIO
import time
import argparse
import os
import time
import glob
from RPLCD.gpio import CharLCD
from multiprocessing import Process
from pushover import Client
import ConfigParser
import io
import logging
from logging.config import fileConfig

#fileConfig('logging_config.ini')
#logger = logging.getLogger()

GPIO.setwarnings(False)

homeFolder=os.environ['HOME'] #pridobim domačo mapo uporabnika, ki je pognal program
logPath=homeFolder+'/.garage/logs' #nastavim mapo v katero se bo shranjevala dnevniška datoteka
logFile=homeFolder+'/.garage/logs/toggleRelay.log'

def checkLogFilePath(): #metoda, ki preveri ali obstajajo vse ptrebne mape in jih po potrebi ustvari
    if(not os.path.exists(homeFolder+'/.garage')):
        os.mkdir(homeFolder+'/.garage')
    if(not os.path.exists(logPath)):
        os.mkdir(logPath)

def readConf(section, vars, val_dict):
    configParser = ConfigParser.RawConfigParser(allow_no_value=True)
    configParser.read(os.environ['HOME']+'/.garage/garage.conf')
    for value in vars:
        val_dict[value] = int(configParser.get(section, value))

def lcd_write(line1, line2):
    lcd.clear()
    lcd.cursor_pos = (0, 0)
    lcd.write_string(line1)
    lcd.cursor_pos = (1, 0)
    lcd.write_string(line2)

def read_temp_raw():
```

```
47      f = open(device_file, 'r')
48      lines = f.readlines()
49      f.close()
50      return lines
51
52  def read_temp():
53      lines = read_temp_raw()
54      while lines[0].strip()[-3:] != 'YES':
55          time.sleep(0.2)
56          lines = read_temp_raw()
57      equals_pos = lines[1].find('t=')
58      if equals_pos != -1:
59          temp_string = lines[1][equals_pos+2:]
60          temp_c = int(temp_string) / 1000.0 # TEMP_STRING IS THE SENSOR OUTPUT,
            ↪  MAKE SURE IT'S AN INTEGER TO DO THE MATH
61          temp_c = round(temp_c, 1) # ROUND THE RESULT TO 1 PLACE AFTER THE
            ↪  DECIMAL
62          return temp_c
63
64  def blink(LED):
65      GPIO.output(LED, GPIO.HIGH)  # led on
66      time.sleep(.5)
67      GPIO.output(LED, GPIO.LOW) # led off
68      time.sleep(.5)
69
70  def toggleGarage():
71      GPIO.output(GPIO_VARS_DICT['RELAY'], 0)
72      time.sleep(.5)
73      GPIO.output(GPIO_VARS_DICT['RELAY'], 1)
74
75  def checkDoor():
76      if GPIO.input(GPIO_VARS_DICT['REED_OPEN']) == True:
77          return "odprta"
78      elif GPIO.input(GPIO_VARS_DICT['REED_CLOSED']) == True:
79          return "zaprta"
80      else:
81          return "priprta"
82
83  def checkCar():
84      GPIO.output(GPIO_VARS_DICT['TRIG'], False)
85      time.sleep(0.001)
86
87      GPIO.output(GPIO_VARS_DICT['TRIG'], True)
88      time.sleep(0.00001)
89      GPIO.output(GPIO_VARS_DICT['TRIG'], False)
90
91      while GPIO.input(GPIO_VARS_DICT['ECHO'])==0:
92        pulse_start = time.time()
93
94      while GPIO.input(GPIO_VARS_DICT['ECHO'])==1:
95        pulse_end = time.time()
```

```
96
97      pulse_duration = pulse_end - pulse_start
98      distance = pulse_duration * 17150
99
100     return round(distance, 2)
101
102 def monitorCar():
103
        ↪  GPIO.add_event_detect(GPIO_VARS_DICT['OVERRIDE_CAR'],GPIO.RISING,bouncetime=300)
104     distance = checkCar()
105     for x in range(0,TIMEOUTS_VARS_DICT['CAR_STATUS_TIMEOUT']):
106         if GPIO.event_detected(GPIO_VARS_DICT['OVERRIDE_CAR']):
107             break
108         elif checkDoor() == 'zaprta':
109             break
110         blink(GPIO_VARS_DICT['LED_MONITOR_CAR'])
111         if (distance >=25 and checkCar() <= 20) or (distance <=20 and
            ↪  checkCar() >= 25):
112             for i in range(0,5):
113                 blink(GPIO_VARS_DICT['LED_MONITOR_CAR'])
114             if GPIO.event_detected(GPIO_VARS_DICT['OVERRIDE_CAR']):
115                 break
116             elif checkDoor() != 'odprta':
117                 break
118             if distance >=25 and checkCar() <= 20:
119                 toggleGarage()
120                 while checkDoor() != "zaprta":
121                     time.sleep(5)
122                 time.sleep(2)
123                 break
124             elif distance <=20 and checkCar() >= 25:
125                 toggleGarage()
126                 pushover.send_message("Avto odpeljal! Zapiram garažo!",
                    ↪  title="Garaža")
127                 while checkDoor() != "zaprta":
128                     time.sleep(5)
129                     pushover.send_message("Avto odpeljal! Garaža zaprta!",
                        ↪  title="Garaža")
130                 time.sleep(2)
131                 break
132
133 def monitorTemp():
134     time.sleep(TIMEOUTS_VARS_DICT['BEGIN_TEMP_WATCH'])
135
        ↪  GPIO.add_event_detect(GPIO_VARS_DICT['OVERRIDE_TEMP'],GPIO.RISING,bouncetime=300)
136     count = 0
137     while 1:
138         blink(GPIO_VARS_DICT['LED_MONITOR_TEMP'])
139         if GPIO.event_detected(GPIO_VARS_DICT['OVERRIDE_TEMP']):
140             break
141         elif checkDoor() == 'zaprta':
```

```
142             break
143         temp = read_temp()
144         if temp < TEMP_VARS_DICT['MIN_TEMP']:
145             toggleGarage()
146             pushover.send_message("Temperatura v garaži prenizka! Zapiram
                ↪  garažo!", title="Garaža prehladna!")
147             while checkDoor() != "zaprta":
148                 time.sleep(1)
149             pushover.send_message("Garaža zaprta zaradi prenizke temperature!",
                ↪  title="Garaža zarta!")
150             time.sleep(2)
151             break;
152         elif temp > TEMP_VARS_DICT['MAX_TEMP']:
153             toggleGarage()
154             pushover.send_message("Temperatura v garaži previsoka! Zapiram
                ↪  garažo!", title="Garaža pretopla!")
155             while checkDoor() != "zaprta":
156                 time.sleep(1)
157             pushover.send_message("Garaža zaprta zaradi previsoke
                ↪  temperature!", title="Garaža zarta!")
158             time.sleep(2)
159             break;
160         count += 1
161
162
163 def arguments():
164     parser = argparse.ArgumentParser()
165     parser.add_argument("-t", "--toggle", action="store_true", help="Trigger
        ↪  garage doors relay.")
166     parser.add_argument('-C', '--car-status', action="store_true", help="Check
        ↪  wether or not the car is in the garage.")
167     parser.add_argument('-S', '--door-status', action="store_true",
        ↪  help="Preveri v kaksnem stanju so vrata.")
168     args = parser.parse_args()
169
170     if args.toggle == True:
171         logging.info("Toggle garage activated.")
172         if checkDoor() == 'zaprta':
173             logging.info("Garage closed.")
174             toggleGarage()
175             logging.info("Opening garage...")
176             logging.info("Waiting for garage to open.")
177             for x in range(0,60):
178                 if checkDoor() == 'odprta':
179                     logging.info("Garage opened.")
180                     logging.debug("Sending push notifiaction over Pushover...")
181                     pushover.send_message("Garaža odprta!", title="Garaža")
182                     logging.debug("Pushover notification send.")
183                     break;
184                 elif x == 60:
185                     logging.warning("Couldn't open garage.")
```

```
186                     time.sleep(1)
187                 time.sleep(1)
188             try:
189                 logging.debug("Starting car starus monitoring...")
190                 c = Process(target=monitorCar,args=())
191                 c.start()
192                 logging.debug("Car monitoring running.")
193                 logging.debug("Starting temperature monitoring...")
194                 t = Process(target=monitorTemp,args=())
195                 t.start()
196                 logging.debug("Temperature monitoring running.")
197                 c.join()
198                 t.join()
199                 logging.debug("Waiting for doors to close...")
200                 while checkDoor() != 'zaprta':
201                     time.sleep(1)
202                 logging.info("Doors closed.")
203                 logging.debug("Sending Pushover notification...")
204                 pushover.send_message('Garaža zaprta!',title="Garaža")
205                 logging.debug("Pushover notification sent.")
206             except:
207                 print "Couldn't start thread"
208         elif checkDoor() == 'odprta':
209             logging.debug("Garage is open.")
210             cd = Process(target=closeDoor(),args=())
211             cd.start()
212             cd.join(60)
213         else:
214             doorAjar()
215 
216     elif args.car_status == True:
217         if checkCar() < 15:
218             print "Avto je v garaži!"
219         else:
220             print "Avta ni v garaži!"
221     elif args.door_status == True:
222         lcd.clear()
223     else:
224         print 'nic'
225         destroy()
226 
227 def closeDoor():
228     toggleGarage()
229     pushover.send_message("Zapiram garažo!", title="Garaža")
230     for x in range(0, TIMEOUTS_VARS_DICT['AJAR_TIMEOUT']):
231         if checkDoor() == 'zaprta':
232             pushover.send_message("Garaža zaprta!", title="Garaža")
233             time.sleep(2)
234         time.sleep(1)
235 
236 def doorAjar():
```

```
237     for attempts in range(0, TIMEOUTS_VARS_DICT['AJAR_CLOSE_ATTEMPTS']):
238         toggleGarage()
239         for x in range(0, TIMEOUTS_VARS_DICT['AJAR_TIMEOUT']):
240             if checkDoor() != 'priprta':
241                 break
242             time.sleep(1)
243         if checkDoor() == 'zaprta':
244             break
245         #elif checkDoor() == 'odprta':
246         #    closeDoor()
247 
248 def destroy():
249     GPIO.output(GPIO_VARS_DICT['LED_MONITOR_CAR'], GPIO.LOW)   # led off
250     GPIO.output(GPIO_VARS_DICT['LED_MONITOR_TEMP'], GPIO.LOW)   # led off
251     GPIO.cleanup()
252 
253 def setup():
254     #variables setup
255     global lcd,base_dir,device_folder,device_file
256     #read from config
257     logging.debug("Setting variables for reading from config file...")
258     GPIO_VARS =
        ↪  ['TRIG','ECHO','RELAY','OVERRIDE_CAR','OVERRIDE_TEMP','LED_MONITOR_CAR','LED_MONITOR
259     TEMP_VARS = ['MAX_TEMP','MIN_TEMP']
260     TIMEOUTS_VARS =
        ↪  ['AJAR_TIMEOUT','CAR_STATUS_TIMEOUT','BEGIN_TEMP_WATCH','AJAR_CLOSE_ATTEMPTS']
261     LCD_VARS = ['cols','rows','pin_rs','pin_e','d4','d5','d6','d7']
262     global GPIO_VARS_DICT, TEMP_VARS_DICT, TIMEOUTS_VARS_DICT, LCD_VARS_DICT
263     TEMP_VARS_DICT = dict()
264     TIMEOUTS_VARS_DICT = dict()
265     GPIO_VARS_DICT = dict()
266     LCD_VARS_DICT = dict()
267     logging.debug("Variables for reading from config file set.")
268     logging.debug("Reading GPIO configuration..."")
269     readConf('gpio',GPIO_VARS,GPIO_VARS_DICT)
270     logging.debug("Finished reading GPIO configuration")
271     logging.debug("Reading temperature configuration..."")
272     readConf('temperature',TEMP_VARS,TEMP_VARS_DICT)
273     logging.debug("Finished reading temperature configuration")
274     logging.debug("Reading timeouts configuration..."")
275     readConf('timeouts',TIMEOUTS_VARS,TIMEOUTS_VARS_DICT)
276     logging.debug("Finished reading timeouts configuration")
277     logging.debug("Reading LCD configuration..."")
278     readConf('lcd',LCD_VARS,LCD_VARS_DICT)
279     logging.debug("Finished reading LCD configuration")
280     #GPIO setup
281     logging.debug("Setting up GPIO...")
282     GPIO.setmode(GPIO.BCM)
283     GPIO.setup(GPIO_VARS_DICT['TRIG'],GPIO.OUT)
284     GPIO.setup(GPIO_VARS_DICT['ECHO'],GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
```

```
285
        ↪ GPIO.setup(GPIO_VARS_DICT['OVERRIDE_CAR'],GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
286
        ↪ GPIO.setup(GPIO_VARS_DICT['OVERRIDE_TEMP'],GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
287     GPIO.setup(GPIO_VARS_DICT['LED_MONITOR_CAR'], GPIO.OUT)   # Set LedPin's
        ↪ mode is output
288     GPIO.setup(GPIO_VARS_DICT['LED_MONITOR_TEMP'], GPIO.OUT)   # Set LedPin's
        ↪ mode is output
289     GPIO.setup(GPIO_VARS_DICT['RELAY'], GPIO.OUT)
290     GPIO.setup(GPIO_VARS_DICT['REED_OPEN'], GPIO.IN,
        ↪ pull_up_down=GPIO.PUD_DOWN)
291     GPIO.setup(GPIO_VARS_DICT['REED_CLOSED'], GPIO.IN,
        ↪ pull_up_down=GPIO.PUD_DOWN)
292     GPIO.setwarnings(False)
293     logging.debug("GPIO setup finished.")
294     #LCD setup
295     logging.debug("Setting up LCD...")
296     lcd = CharLCD(cols=LCD_VARS_DICT['cols'], rows=LCD_VARS_DICT['rows'],
        ↪ pin_rs=LCD_VARS_DICT['pin_rs'], pin_e=LCD_VARS_DICT['pin_e'],
        ↪ pins_data=[LCD_VARS_DICT['d4'],LCD_VARS_DICT['d5'],LCD_VARS_DICT['d6'],LCD_VARS_DICT
297     logging.debug("LCD setup finished.")
298     #temp sensor setup
299     logging.debug("Setting up temperature sensor...")
300     os.system('modprobe w1-gpio')
301     os.system('modprobe w1-therm')
302     base_dir = '/sys/bus/w1/devices/'
303     device_folder = glob.glob(base_dir + '28*')[0]
304     device_file = device_folder + '/w1_slave'
305     logging.debug("Temperature sensor setup finished.")
306     #pushover setup
307     logging.debug("Setting up Pushover...")
308     configParser = ConfigParser.RawConfigParser(allow_no_value=True)
309     configParser.read(os.environ['HOME']+'/.garage/garage.conf')
310     global pushover
311     pushover = Client(configParser.get('pushover', 'user_key'),
        ↪ api_token=configParser.get('pushover', 'api_token'))
312     logging.debug("Pushover sertup finished.")
313
314 if __name__=="__main__":
315     try:
316         logging.debug("Checking if /tmp/LCD_temp.pid exists...")
317         if os.path.isfile("/tmp/LCD_temp.pid"):
318             logging.debug("/tmp/LCD_temp.pid exists.")
319             logging.debug("Killing LCD_temperature.py...")
320             os.system("kill $(cat /tmp/LCD_temp.pid)")
321             logging.debug("LCD_temperature.py killed.")
322             logging.debug("Removing /tmp/LCD_temp.pid...")
323             os.unlink("/tmp/LCD_temp.pid")
324             logging.debug("/tmp/LCD_temp.pid removed.")
325         #m = Process(target=mainScreen,args=())
326         setup()
```

```
327         lcd.clear()
328         #m.start()
329         logging.debug("Starting temperature_LCD.py...")
330         os.system('python temperature_LCD.py &')
331         logging.debug("LCD_temperature running.")
332         arguments()
333         #m.join()
334     except KeyboardInterrupt:
335         destroy()
336     finally:
337         destroy()
338         if os.path.isfile("/tmp/LCD_temp.pid"):
339             os.system("kill $(cat /tmp/LCD_temp.pid)")
340             os.unlink("/tmp/LCD_temp.pid")
341         os.system('python temperature_LCD.py &')
```

10.3 temperature_LCD.py - izpisovanje na LCD

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from datetime import datetime
import ConfigParser
from RPLCD.gpio import CharLCD
import RPi.GPIO as GPIO
import argparse
import os
import time
import glob
import sys

GPIO.setwarnings(False)

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def readConf(section, vars, val_dict):
    configParser = ConfigParser.RawConfigParser(allow_no_value=True)
    configParser.read(os.environ['HOME']+'/.garage/garage.conf')
    for value in vars:
        val_dict[value] = int(configParser.get(section, value))

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = int(temp_string) / 1000.0 # TEMP_STRING IS THE SENSOR OUTPUT,
        ↪  MAKE SURE IT'S AN INTEGER TO DO THE MATH
        temp_c = round(temp_c, 1) # ROUND THE RESULT TO 1 PLACE AFTER THE
        ↪  DECIMAL
        return temp_c

def lcd_write(line1, line2):
    lcd.cursor_pos = (0, 0)
```

```
48      lcd.write_string(line1)
49      lcd.cursor_pos = (1, 0)
50      lcd.write_string(line2)
51
52  def init():
53      #variables setup
54      global lcd,base_dir,device_folder,device_file
55      #read from config
56      GPIO_VARS =
        ↪  ['TRIG','ECHO','RELAY','OVERRIDE_CAR','OVERRIDE_TEMP','LED_MONITOR_CAR','LED_MONITOR
57      TEMP_VARS = ['MAX_TEMP','MIN_TEMP']
58      TIMEOUTS_VARS =
        ↪  ['AJAR_TIMEOUT','CAR_STATUS_TIMEOUT','BEGIN_TEMP_WATCH','AJAR_CLOSE_ATTEMPTS']
59      LCD_VARS = ['cols','rows','pin_rs','pin_e','d4','d5','d6','d7']
60      global GPIO_VARS_DICT, TEMP_VARS_DICT, TIMEOUTS_VARS_DICT, LCD_VARS_DICT
61      TEMP_VARS_DICT = dict()
62      TIMEOUTS_VARS_DICT = dict()
63      GPIO_VARS_DICT = dict()
64      LCD_VARS_DICT = dict()
65      readConf('gpio',GPIO_VARS,GPIO_VARS_DICT)
66      readConf('temperature',TEMP_VARS,TEMP_VARS_DICT)
67      readConf('timeouts',TIMEOUTS_VARS,TIMEOUTS_VARS_DICT)
68      readConf('lcd',LCD_VARS,LCD_VARS_DICT)
69      #GPIO setup
70      GPIO.setmode(GPIO.BCM)
71
        ↪  GPIO.setup(GPIO_VARS_DICT['OVERRIDE_TEMP'],GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
72      GPIO.setup(GPIO_VARS_DICT['LED_MONITOR_TEMP'], GPIO.OUT)   # Set LedPin's
        ↪  mode is output
73      #LCD setup
74      lcd = CharLCD(cols=LCD_VARS_DICT['cols'], rows=LCD_VARS_DICT['rows'],
        ↪  pin_rs=LCD_VARS_DICT['pin_rs'], pin_e=LCD_VARS_DICT['pin_e'],
        ↪  pins_data=[LCD_VARS_DICT['d4'],LCD_VARS_DICT['d5'],LCD_VARS_DICT['d6'],LCD_VARS_DICT
75      #temp sensor setup
76      os.system('modprobe w1-gpio')
77      os.system('modprobe w1-therm')
78      base_dir = '/sys/bus/w1/devices/'
79      device_folder = glob.glob(base_dir + '28*')[0]
80      device_file = device_folder + '/w1_slave'
81      #pushover setup
82      configParser = ConfigParser.RawConfigParser(allow_no_value=True)
83      configParser.read(os.environ['HOME']+'/.garage/garage.conf')
84
85  def destroy():
86      GPIO.cleanup()
87
88  if __name__=="__main__":
89      pid = str(os.getpid())
90      pidfile = "/tmp/LCD_temp.pid"
91
92      if os.path.isfile(pidfile):
```

```
93          print "%s already exists, exiting" % pidfile
94          sys.exit()
95      file(pidfile, 'w').write(pid)
96      try:
97          init()
98          lcd.clear()
99          while True:
100         ^^Ilcd_write(datetime.now().strftime("%d.%m.%y  %H:%M"), "Temp: " +
            ↪  str(read_temp())+unichr(223)+"C")
101         ^^Itime.sleep(1)
102     except:
103         os.unlink(pidfile)
104         destroy()
```

10.4 checkRelay.py

```
1               #!/usr/bin/python
2
3               #Program namenjen preklopu releja iz zaprtega v odprto stanje,
                ↪  ce je le ta predolgo zaprt
4               #Uporabljen kot resitev problema #1 (glej BUGS.md)
5
6               import RPi.GPIO as GPIO #import the GPIO library
7               import time
8               import datetime
9               from datetime import datetime
10              import os
11              import logging
12              logging.basicConfig(filename='logs/checkRelay.log')
13
14              homeFolder=os.environ['HOME']
15              path=homeFolder+'/.garage/logs'
16
17              def checkLogFilePath():
18                  if(not os.path.exists(homeFolder+'/.garage')):
19                      os.mkdir(homeFolder+'/.garage')
20                  if(not os.path.exists(path)):
21                      os.mkdir(path)
22
23              logger=logging.getLogger(__name__)
24
25              checkLogFilePath()
26
27              GPIO.setmode(GPIO.BOARD)
28              GPIO.setup(12, GPIO.OUT)
29              GPIO.output(12, GPIO.HIGH)
30
31              while True:
32                  if GPIO.input(12) == False:
33                      time.sleep(1.5)
```

```
34                      if GPIO.input(12) == False:
35                          logger.warning('Relay closed for too long.')
36                          GPIO.output(12, not GPIO.input(12))
37                          logger.info('Relay opened automatically.')
38                  time.sleep(.5)
39              GPIO.cleanup()
```