PiFace Control and Display 2

Installation

Followed installation tutorial from PiFaces webpage, where is very clearly demonstrated what you must do in different stages of installation. There were little problems at the step where you are installing CAD software to Raspberry, because Piface CAD is not supported by *apt-get* installation anymore. Happily, there was information that there is also *Github* guide for installation. There were still few bumps on the road, during installation but we manage to get it through with couple of extra package installation or running some python package updates during those steps.

Install on Raspbian Strech and RaspberryPi 3

The piface packages are currently not in the main repositories. You can, however, simply build them from their sources:

- Enable the spi-interface:

```
$ sudo raspi-config
Interfacing Options -> SPI -> Yes
```

- https://github.com/tompreston/python-lirc

```
$ sudo aptitude install liblircclient-dev cython gcc python{,3}-setuptools python{,3}-dev
$ git clone https://github.com/tompreston/python-lirc.git
$ cd python-lirc/
$ make py3 && sudo python3 setup.py install
$ make py2 && sudo python setup.py install
```

- https://github.com/piface/pifacecommon

```
$ git clone https://github.com/piface/pifacecommon.git
$ cd pifacecommon/
$ sudo python setup.py install
$ sudo python3 setup.py install
```

- https://github.com/piface/pifacecad

```
$ git clone https://github.com/piface/pifacecad.git
$ cd pifacecad/
$ sudo python setup.py install
$ sudo python3 setup.py install
```

- run the hello world demo:

```
>>> import pifacecad

>>> cad = pifacecad.PiFaceCAD()    # create PiFace Control and Display object
>>> cad.lcd.backlight_on()         # turns the backlight on
>>> cad.lcd.write("Hello, world!") # writes hello world on to the LCD
```

- Cleanup:

```
$ sudo rm -rf pifacecad/ pifacecommon/ python-lirc/
```

Links

http://www.piface.org.uk/guides/setting_up_pifacecad/fitting_PiFace_Control_and_Display/

https://github.com/piface/pifacecad

http://piface.github.io/pifacecad/example.html

Project: Virtual lock

Plan:

4-digit lock which have a 2-step authentication

- Program language: Python
- Waiting predefined combination from user.
- When user input is correct, a new random 4-digit combination is generated.
- Sending automatic email to user where the second combination is included.
- Waiting new combination from user. If input is correct, open lock.
- User can lock the lock again after it's open.
- If user input codes wrong 4 times, program automatically sends warning message to owner and message holds information about secure code.
- If secure code inputted wrongly, program generates previous stage again.
- Returning to normal mode only if secure code is correct.

Implementation:

In the first phase, we started to implement our plan with a very simple program structure that included several separate functions called by the main program at different stages of the program (Appendix 1). In this way, we implemented a simple and logical entity that allowed us to experiment with the practical implementation of the program.

In this implementation, we quickly noticed that the screen refresh was clearly noticeable. At this point, the program was based on the loop structure that when rotated caused this phenomenon to appear on the screen. In addition, there was a slight delay in pressing the buttons due to the program structure. After these notifications we started to think a different solution for the program. We noticed from PiFace documentation, that there is also possibility to use Interrupts -handler instead using switches to control program.

In second phase we planned to change program structure to implement class model structure and object-orientated programming (Appendix 2). After these changes to program we managed to disappear this loop effect in display and got better response in switch presses to control program. With these changes we were happy about our program's functionality and response.

In class Lock in Appendix2 we are using library named *threading* where we are importing Barrier -function. With this function we used different threads in our program, and it helped implement different interrupts from different stages. Also, nice to know learning, was to learn how program email sender with Python code, which was after all it was quite easy task and there were many tutorials available. Note, In Google account you must use 2-step authentication and "allow low secure apps", after this You can make own app password for Your project. Google will help You with this one.

For future proposal:

It could be nice continue to project where this implementation is connected or soldered to real electrical circuit and after that controlling real electrical lock.

Appendix 1

```python
import os
import pifacecad as p
import random
import smtplib
import sys
import time

cad = p.PiFaceCAD()

def authentication_numbers():
    return [random.randint(0, 9) for p in range(0, 4)]

def sending_mail(input):

    SERVER = 'smtp.gmail.com'
    FROM = 'tkdemoprojects@gmail.com'
    PASSW = [redacted]
    TO = 'teemu.kukko@gmail.com'
    PORT = 587

    with smtplib.SMTP(SERVER, PORT) as smtp:
        smtp.ehlo()
        smtp.starttls()
        smtp.ehlo()

        smtp.login(FROM, PASSW)

        SUBJECT = 'Verification code'
        BODY = f'You are trying to open virtual locker.\n\n\nAuthentication code is: {input}'
        MESSAGE = f'Subject: {SUBJECT}\n\n{BODY}'

        smtp.sendmail(FROM, TO, MESSAGE)

def error_mail(input):

    SERVER = 'smtp.gmail.com'
    FROM = 'tkdemoprojects@gmail.com'
    PASSW = [redacted]
    TO = 'teemu.kukko@gmail.com'
    PORT = 587

    with smtplib.SMTP(SERVER, PORT) as smtp:
        smtp.ehlo()
        smtp.starttls()
        smtp.ehlo()

        smtp.login(FROM, PASSW)

        SUBJECT = 'Unauthorized access try!'
        BODY = f'Somebody has tried to open Your locker, please enter safety code to activate lock again\n\nSafety code: {input}'
        MESSAGE = f'Subject: {SUBJECT}\n\n{BODY}'

        smtp.sendmail(FROM, TO, MESSAGE)

def control_lock(array):

    lock_numbers = [0, 0, 0, 0]
    error_counter = 0
    while 1:
        # switch0
        if cad.switches[0].value == 1:
            lock_numbers[0] = lock_numbers[0] + 1
            if lock_numbers[0] > 9:
                lock_numbers[0] = 0
        # switch1
        if cad.switches[1].value == 1:
            lock_numbers[1] = lock_numbers[1] + 1
            if lock_numbers[1] > 9:
                lock_numbers[1] = 0
        # switch2
        if cad.switches[2].value == 1:
            lock_numbers[2] = lock_numbers[2] + 1
            if lock_numbers[2] > 9:
                lock_numbers[2] = 0
        # switch3
        if cad.switches[3].value == 1:
            lock_numbers[3] = lock_numbers[3] + 1
            if lock_numbers[3] > 9:
                lock_numbers[3] = 0

        #set cursor to column 0 on the second row
        cad.lcd.set_cursor(0, 1)
        cad.lcd.write("Code:{}".format(lock_numbers))

        if cad.switches[4].value == 1:
            if lock_numbers == array:
                return 1
            elif error_counter == 4:
                return 2
            else:
                cad.lcd.set_cursor(0, 1)
                cad.lcd.write("Wrong Code!     ")
                error_counter += 1
                print(error_counter)
                time.sleep(2)
```

Appendix 1

```
def lock_open():
    i = 0
    while i < 5:
        cad.lcd.set_cursor(0, 1)
        cad.lcd.write("* * * * * * * * ")
        time.sleep(1)
        cad.lcd.set_cursor(0, 1)
        cad.lcd.write(" * * * * * * * *")
        time.sleep(1)
        i += 1

def lock_locked():
    counter = 0
    while 1:
        cad.lcd.set_cursor(0, 1)
        cad.lcd.write("Press: Btn4 x 4 ")
        if cad.switches[4].value == 1:
            counter += 1
            if counter == 4:
                break
    return 'Locked'

def restart_program():
    """Restarts the current program.
    Note: this function does not return. Any cleanup action (like
    saving data) must be done before calling this function."""
    python = sys.executable
    os.execl(python, python, * sys.argv)

def print_status(input):
    cad.lcd.set_cursor(0, 0)
    cad.lcd.write("Status: " + lock_status)
    cad.lcd.set_cursor(0, 1)


if __name__ == "__main__":

    cad.lcd.backlight_on()
    cad.lcd.cursor_off()
    lock_code = [9, 9, 7, 3]
    lock_status = "Locked"
    print_status(lock_status)

    status = control_lock(lock_code)
    if status == 1:
        lock_status = "Waiting"
        print_status(lock_status)
    elif status == 2:
        error_code = authentication_numbers()
        error_mail(error_code)
        while 1:
            if control_lock(error_code) == 1:
                restart_program()
            else:
                continue
    else:
        cad.lcd.clear()                 # clear the screen (also sends the cursor home)
        cad.lcd.write("Fatal Error     ")    # ends program
        exit()

    authentication = authentication_numbers()
    sending_mail(authentication)
    if control_lock(authentication) == 1:
        lock_status = "Open      "
        print_status(lock_status)
        lock_open()
    elif status == 2:
            error_code = authentication_numbers()
            error_mail(error_code)
            while 1:
                if control_lock(error_code) == 1:
                    restart_program()
                else:
                    continue
    else:
        cad.lcd.clear()                 # clear the screen (also sends the cursor home)
        cad.lcd.write("Fatal Error     ")    # ends program
        exit()

    time.sleep(10)
    lock_status = lock_locked()
    if lock_status == 'Locked':
        restart_program()

```

Appendix 2

```
import pifacecad
from time import sleep
from threading import Barrier

import random
import smtplib

class Lock(object):
    def __init__(self):
        self.cad = pifacecad.PiFaceCAD()
        self.authMode = 1 # 0 => Safe code / 1 => 1st auth / 2 => 2nd auth
        self.lockCode = [9, 9, 7, 3] # First auth code
        self.authCode = [0, 0, 0, 0]
        self.lockNumbers = [0, 0, 0, 0]
        self.end_barrier = Barrier(2) # Create two barriers to hold up the threads
        self.errCounter = 0
        self.lockingCounter = 0

        self.cad.lcd.set_cursor(0, 0)
        self.lockStatus = "Locked"
        self.write_display("Status: {}".format(self.lockStatus), cursor_row = 0)
        self.write_display(self.lockNumbers, cursor_row = 1, input_code=True)

        self.server = 'smtp.gmail.com'
        self.sender = 'tkdemoprojects@gmail.com'
        self.passwd = '[redacted]'
        self.receiver = 'matti.krusviita@edu.turkuamk.fi'

        self.port = 587

    # Control and validate lock number sequence
    def control_lock(self, event):
        if self.lockStatus == "Unlocked":
            if event.pin_num == 4: # Only switch4 will increase the counter
                self.lockingCounter += 1
                self.write_display("Locking in: {}".format(str(4 - self.lockingCounter)), cursor_row = 1)
            if self.lockingCounter >= 4: # Locking the lock, reset values
                self.lockingCounter = 0
                self.authMode = 1
                self.lockStatus = "Locked"
                self.write_display("Status: {}".format(self.lockStatus), cursor_row = 0) # Update upper row
                self.lockNumbers = [0, 0, 0, 0]
                self.write_display(self.lockNumbers, cursor_row = 1, input_code=True) # Update lower row
                self.end_barrier.wait()
        else:
            # switch0
            if event.pin_num == 0:
                self.lockNumbers[0] = self.lockNumbers[0] + 1
                if self.lockNumbers[0] > 9:
                    self.lockNumbers[0] = 0
            # switch1
            if event.pin_num == 1:
                self.lockNumbers[1] = self.lockNumbers[1] + 1
                if self.lockNumbers[1] > 9:
                    self.lockNumbers[1] = 0
            # switch2
            if event.pin_num == 2:
                self.lockNumbers[2] = self.lockNumbers[2] + 1
                if self.lockNumbers[2] > 9:
                    self.lockNumbers[2] = 0
            # switch3
            if event.pin_num == 3:
                self.lockNumbers[3] = self.lockNumbers[3] + 1
                if self.lockNumbers[3] > 9:
                    self.lockNumbers[3] = 0

            # Validate if lock number sequence is correct
            if event.pin_num == 4:
                if self.authCode == self.lockNumbers:
                    if self.authMode == 0: # Safe code input correct
                        self.authMode = 1
                        self.authCode = self.lockCode
                        self.errCounter = 0
                        self.lockNumbers = [0, 0, 0, 0]
                        self.write_display("Correct!", cursor_row = 1)
                        sleep(2)
                        self.lockStatus = "Locked"
                    elif self.authMode == 1: # 1st auth input correct
                        self.authMode += 1
                        self.lockStatus = "Waiting"
                        self.lockNumbers = [0, 0, 0, 0]
                        self.write_display("Correct!", cursor_row = 1)
                        sleep(2)
                        self.end_barrier.wait()
                    elif self.authMode == 2: # 2nd auth input correct
                        self.authMode += 1
                        self.lockStatus = "Unlocked"
                        self.write_display("Open!", cursor_row = 1)
                        self.end_barrier.wait()
                    self.write_display("Status: {}".format(self.lockStatus), cursor_row = 0)
                else:
                    self.errCounter += 1
                    self.write_display("Wrong Code!  x{}".format(str(self.errCounter)), cursor_row = 1)
                    sleep(2)
                    if self.errCounter >= 4: # Go to error mode
                        self.authMode = 0
                        self.end_barrier.wait()
                        self.write_display("Insert safe code {}".format(str(self.errCounter)), cursor_row = 1)
                        sleep(2)
            if self.authMode != 3: # When opened (authMode = 3), we won't show lockNumbers
                self.write_display(self.lockNumbers, cursor_row = 1, input_code=True)
```

Appendix 2

```
            if self.authMode != 3: # When opened (authMode = 3), we won't show lockNumbers
                self.write_display(self.lockNumbers, cursor_row = 1, input_code=True)

    # Create event listener to react button events
    def button_event_listener(self):
        if self.authMode == 1:
            self.authCode = self.lockCode
        self.listener = pifacecad.SwitchEventListener(chip=self.cad)
        for i in range(5):
            self.listener.register(i, pifacecad.IODIR_FALLING_EDGE, self.control_lock)
        self.listener.activate()

        self.end_barrier.wait()
        self.listener.deactivate()

    def write_display(self, text, cursor_row, input_code=False):
        # cursor_row "0" => upper row, "1" => lower row
        try:
            if input_code:
                output = " ".join(list(map(str,text)))
            else:
                output = str(text)
        except:
            output = "error"
        # Check and fill trailing spaces with empty spaces
        output_lenght = len(output)
        while output_lenght < 16:
            output_lenght += 1
        output = output.ljust(output_lenght)
        # Set cursor and write the row
        self.cad.lcd.set_cursor(0, cursor_row)
        self.cad.lcd.write(output)

    # Generate random auth.numbers
    def authentication_numbers(self):
        return [random.randint(0, 9) for p in range(0, 4)]

    # Create email message for second and backlock authentication
    def create_email(self):
        self.authCode = self.authentication_numbers()
        if self.authMode:
            SUBJECT = 'Verification code'
            BODY = f'You are trying to open virtual locker.\n\n\nAuthentication code is: {self.authCode}'
        else:
            SUBJECT = 'Unauthorized access try!'
            BODY = f'Somebody has tried to open Your locker, please enter safety code to activate lock again\n\nSafety code: {self.authCode}'
        message = f'Subject: {SUBJECT}\n\n{BODY}'
        print(message)
        self.send_mail(message)

    def send_mail(self, message):
        SERVER = self.server
        FROM = self.sender
        PASSW = self.passwd
        TO = self.receiver
        PORT = self.port
        with smtplib.SMTP(SERVER, PORT) as smtp:
            smtp.ehlo()
            smtp.starttls()
            smtp.ehlo()
            smtp.login(FROM, PASSW)
            smtp.sendmail(FROM, TO, message)

    def main(self):
        while True:
            self.button_event_listener()
            if self.authMode == 0 or self.authMode == 2:
                self.create_email()
        print("exit")

if __name__ == "__main__":
    job = Lock()
    job.main()
```