Patstāvīgā darba protokols

QR koda nolasītājs

Darbu izstrādāja:

Niks Šķērsts

Darba apraksts

Projekta galvenā doma ir izveidot QR koda nolasītāju, kas izgūst informāciju no koda, un ievieto to SQLITE datubāzē.

Programmai nav pieejama vairāku QR kodu vienlaicīga lasīšanas funkcionalitāte (proti, izvadīt visus atrastos QR kodus ar vienu reizi), bet tā spēj tos lasīt secīgi.

Darbības princips ir sekojošs:

1. Izpilda programmu.
2. Atvērsies logs ar reāllaika kameras video plūsmu.
3. Pavēršot kameru pret QR kodu notiek tā nolasīšana.
4. Kods tiek ievadīts datubāzē. Tiek izvadīts dialoga logs, kas parāda QR kodā saturošo informāciju.
5. Uzspiežot uz pogas “Izvadīt QR kodus”, tiek izvadīti visi QR kodi, kuri atrodas datubāzē kopā ar ievades datumu.

Projekts izmanto:

- Raspberry PI 3+ Model B,
- Raspberry Pi camera v2.

Uzsākot darbu ar Raspberry PI ir nepieciešama SD karte ar jau uzstādītu Raspian sistēmu, un pievienotu kameru. Pirms programmas izpildes ir nepieciešamas veikt izmaiņas konfigurācijā (Caur CLI, TUI vai GUI):

- Interfaces -> Enable Camera Interface

- Advanced options -> Enable Glamor[1]
    - Glamor ir nepieciešams, jo pakotnes imutils (python 3) VideoStream klase, kā arī opencv VideoCapture atsakās izpildīties norādot, ka kameras interfeiss ir atslēgts. Tas notiek neatkarīgi no tā, vai interfeiss realitātē ir ieslēgts vai atslēgts. Kļūdas kods fd-21.
- Palaist komandu *sudo modprobe bcm2835-v4l2* [2]
    - Ne vienmēr ir nepieciešams, bet dažreiz gadās, ka netiek pievienots draiveris kodolam. Šis ir tikai nepieciešams, ja izmanto Raspberry Pi kameras pieslēgvietu. Webkameras šo draiveri neizmanto!
- Ieinstalēt *libcamera-tools, libcamera-apps* pakotnes[3]
    - Ja iepriekšējie pakāpieni nedarbojas, var pārbaudīt kameras darbību izmantojot komandu *libcamera-still*. Tas arī ielādē konfigurāciju vai draiverus, pēc kā parasti arī pazūd ar kameru saistītās kļūdas. Lietotne *raspistill* nav pieejama uz jaunākajām Raspian sistēmām.

Nepieciešamās komandas projektam nepieciešamo pakotņu uzstādīšanai:

- sudo apt-get install libhdf5-dev libhdf5-serial-dev libatlas-base-dev libjasper-dev libqtgui4 libqt4-test -y
- pip install opencv-contrib-python

Programmatūra ir sadalīta divās daļās:

1. QR nolasītājs, kas aprakstīts sadaļā “Kods” tabulā *main.py*.
2. Sqlite datubāze. Sqlite integrācija ar programmas kodu ir aprakstīta sadaļā “Kods” tabulā *Database.py*.

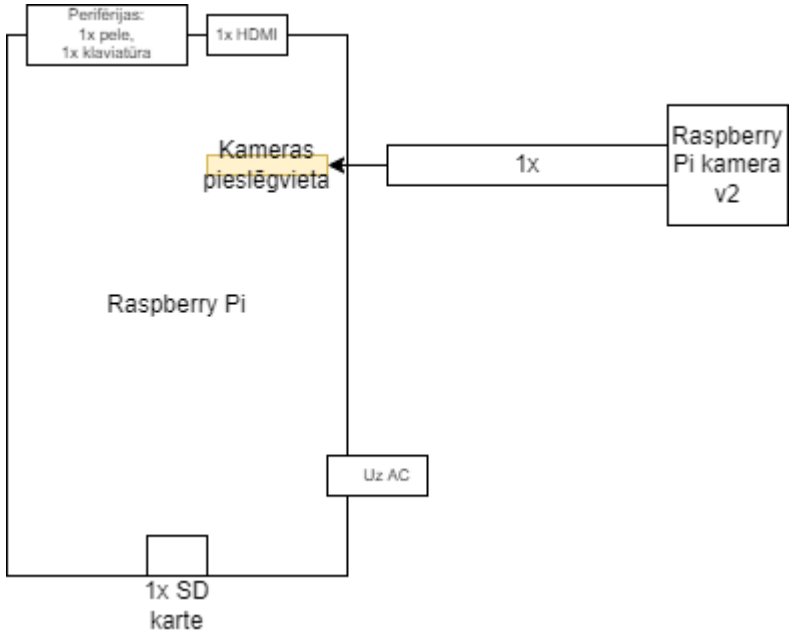
Sqlite datubāzes uzstādīšanai izmantotais kods:

```
CREATE TABLE IF NOT EXISTS barcode (
id integer PRIMARY KEY,
code text NOT NULL,
date integer);
```

Kods izveido datu tabulu *barcode* ar *id, code* un *date* kolonām.

Kolonas nosaukums	**Funkcija**
id	Norāda uz rindas identifikātoru.
code	Saglabā nolasītā QR koda vērtību.
date	Saglabā nolasītā QR koda nolasīšanas laiku un datumu kā integer tipu. Pirms uzglabāšanas, datums ir jāpārveido uz EPOCH formātu.

Slēguma shēma



Attēls 1 - Slēguma shēma

Kods

```
# Import region!
import cv2
import sys
import threading
import time
import tkinter as tk
from PIL import Image, ImageTk
from datetime import datetime

import Database


# Function region

# Popup function deals with listing all the entries from
the underlying database.
# Function goes like this:
# -> Gets data from database. If null, alerts the user
instead.
# -> Outputs header.
# -> Foreach row of data, converts POSIX to human
readable datetime and outputs it.
# -> creates a mainLoop and waits for user to close it.

def popup():
    result = Database.inquiry()
    popupRoot = tk.Tk()
    if result is not None:
        tk.Label(popupRoot, text="Datums un laiks",
font=('Mistral 18 bold')).pack()
        for i in result:
# datetime.fromtimestamp converts EPOCH date integer to
human-readable datetime.
            dt =
datetime.fromtimestamp(i[1]).strftime('%Y-%m-%d
%H:%M:%S')
            tk.Label(popupRoot, text="{0},
{1}".format(i[0], dt), font=('Mistral 18 bold')).pack()
    else:
        tk.Label(popupRoot, text="Netika atrasti
ieraksti", font=('Mistral 18 bold')).pack()
    popupRoot.mainloop()
```

```
# ok_popup blocks the video thread, and unblocks it on
quitting the window.
# It's to avoid bombarding the database with the same QR
code over and over again.
# Alerts the user of successful QR code decode.

def ok_popup(code):
    popupRoot = tk.Tk()
    tk.Label(popupRoot, text="Atrastais QR :
{0}".format(code)).pack()
    tk.Button(popupRoot, text="OK!", font=("Verdana",
18), bg="yellow", command=popupRoot.destroy).pack()
    return popupRoot

# Deals with closing of the program. Releases the used
resources.

def onClose():
    print("[INFO] closing...")
    stopEvent.set()
    cap.release()
    root.quit()
    sys.exit(0)


# Secondary thread. Updates the camera panel and detects
QR codes.
# As far as I know, it's impossible to create a GUI with
tkinter without using multithreading.
# To launch the GUI, you have to run main threads
mainLoop - an infinite loop that blocks the thread.
# Thus, to update anything, it's required to create a
new thread that deals with updates and so forth.
# This method can generate a few errors that aren't
"bad" enough to kill the program.
# It's mandatory to catch them before the app gets
destroyed

def video_loop():
    global panel
    try:
        while not stopEvent.is_set():
            # Gets camera data in form of matrix.
            _, frame = cap.read()
            frame = read_barcodes(frame)
```

```
            # flips the matrix and then creates an image
from it.
            frame = cv2.flip(frame, 1)
            cv2image = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGBA)
            img = Image.fromarray(cv2image)
            imgtk = ImageTk.PhotoImage(image=img)

            # Updates the panel to reflect the changes
in frame. If panel doesn't exist, it creates a new one,
            # else - just reconfigures it.
            if panel is None:
                panel = tk.Label(image=imgtk)
                panel.image = imgtk
                panel.pack(side="left", padx=10,
pady=10)
            else:
                panel.configure(image=imgtk)
                panel.image = imgtk
    except RuntimeError as e:
        print("[INFO] caught a RuntimeError")


# Function that deals with QR code recognition.
# It does have some faults. For example, it does create
false positives from time to time.

def read_barcodes(frame):
    data, points, _ = decoder.detectAndDecode(frame)
    if points is not None:
        points = points[0]
        # Draw a line around the QR code.
        # This for cycle draws a rectangle around the QR
code.
        # It should probably be removed, or I should try
to reduce the amount of false detections.
        # A lot of false detections lead to strange line
drawing and empty data.
        for i in range(len(points)):
            pt1 = [int(val) for val in points[i]]
            pt2 = [int(val) for val in points[(i + 1) %
4]]
            cv2.line(frame, pt1, pt2, color=(255, 0, 0),
thickness=3)
        # False positives are barcodes that have an
empty string for data.
        # This if filters those out before inserting
into database and blocking the thread with ok_popup.
```

```
        if data != '' or "":
            # It is possible that the library doesn't
support reading multiple QRs.
            # As a workaround I have implemented a set()
of QR codes, that gets updated each time I find a new QR
code.
            # I have also implemented an if function
that checks if the code that was presented is already in
the set().
            if not Previous.__contains__(data):
                Database.insert(data)
                ok = ok_popup(data)
                Previous.add(data)
                ok.mainloop()
    # Return the frame, as it will be used to create an
image, that will later be displayed in panel.
    return frame


# On setting the stop event, it should be enough to
terminate the video_loop thread.
# Event is working, but the thread fails to join with
the main thread.
# It would most likely take a class to fix that.
# Something along the lines of:
#
https://github.com/NiksSkersts/iot/commit/6f9c031b8b903b
cb93873b462c363727e2f0baab#diff-dc7b94e12cffc1d23c55f1f9
bf77857827d2148f12a8f010df361e3b2db05533

stopEvent = threading.Event()

# Panel that is responsible for displaying the frames.

panel = None

# Implementation that detects the QR codes from the
matrix provided by the camera.

decoder = cv2.QRCodeDetector()

Previous = set()

# Main part of the code, that deals with construction of
the GUI and initial setup.
# The code fully works on laptop, and I expect it to
fully work on Raspberry Pi.
```

```
if __name__ == '__main__':
    # Creates the GUI
    root = tk.Tk()
    root.title("SCANNER")
    root.config(background='black')
    panel = tk.Label(root)
    panel.grid(row=0, column=0)
    popupButton = tk.Button(root, text="Izvadīt esošos
QR", font=("Verdana", 12), bg="yellow", command=popup)
    popupButton.grid(row=1, column=0)

    # Set up the video capture and QR detection
    cap = cv2.VideoCapture(0)
    time.sleep(2.0)
    detector = cv2.QRCodeDetector()

    # Set up the secondary thread
    thread = threading.Thread(target=video_loop,
args=())
    thread.start()

    # Set a callback to handle when the window is
closed.
    # Breaks the root.mainLoop().
    root.wm_title("qr")
    root.wm_protocol("WM_DELETE_WINDOW", onClose)

    # Blocks the thread with an infinite loop.
    root.mainloop()
```

Tabula 1 - main.py

```
import pytz as pytz
import sqlite3
import time
from datetime import datetime, timedelta, tzinfo,
timezone
from sqlite3 import Error


# Inserts into database. Converts Human readable date to
EPOCH.
```

```
def insert(code):
    conn = sqlite3.connect("data.db")
    insert_query = ''' INSERT INTO barcode(code,date)
VALUES (?,?)'''
    cur = conn.cursor()
    try:
        current =
datetime.now(pytz.timezone("Europe/Riga"))
# time.mktime converts human-readable datetime to EPOCH.
        ti = time.mktime(current.timetuple())
        cur.execute(insert_query, (code, ti))
        conn.commit()
    except Error as e:
        return -1
    return cur.lastrowid


# Counts the entries in database

def count():
    conn = sqlite3.connect("data.db")
    data_copy = conn.execute("SELECT max(rowid) from
barcode")
    values = data_copy.fetchone()[0]
    return values


# selects the rows from database, that gets parsed and
used in listing the data.

def inquiry():
    conn = sqlite3.connect("data.db")
    codes_in_database = set(())
    cur = conn.cursor()
    cur.execute("SELECT code, date FROM barcode")
    rows = cur.fetchall()
    for row in rows:
        codes_in_database.add(row)
    return codes_in_database
```

Tabula 2 - Database.py

Piezīmes

Meklējot informāciju atklāju vairākus interesantus projektus:

1. Number Plate Detection using OpenCV & Python | by Apoorva Sinha | QuikNapp | Medium
2. Scan QR Codes in Real-Time with Raspberry Pi - Hackster.io
3. An OpenCV barcode and QR code scanner with ZBar - PyImageSearch
    - Līdzīga ideja kā man, tikai izmanto citu bibliotēku. Es izmantoju opencv, lai dekodētu QR kodus. Šī ideja izmanto Zbar.
    - Zbar var iztestēt uz Windows sistēmas, bet tā uzstādīšana ir apgrūtinoša. Vieglāk ir izmantot opencv.

Dažādi risinājumi problēmām ar kurām saskāros būvējot projektu

1. python - How to pip or easy_install tkinter on Windows - Stack Overflow
2. Convert datetime to Unix timestamp and convert it back in python - Stack Overflow
    - Datubāze uzglabā datumus kā integer tipa mainīgos. Uzglabājot tekstā, radās kļūdas ar izvadi. SQlite neatbalsta datetime tipus.
3. Epoch Converter - Unix Timestamp Converter

Avoti, ar kuru palīdzību projekts ir veidots:

1. Displaying a video feed with OpenCV and Tkinter - PyImageSearch
2. QR Code Scanner using Raspberry Pi and OpenCV (circuitdigest.com)
3. PySimpleGUI: The Simple Way to Create a GUI With Python – Real Python
4. SQLite Python (sqlitetutorial.net)
5. Detect and Decode QR Code in Image using OpenCV | Lindevs