python-quirc Documentation

Release 0.8.0

SvartalF

May 27, 2012

CONTENTS

python-quirc is a ctypes bindings for quirc, C library for QR codes recognition.

As well as a base library, it has a good features for QR code recognition:

- It is fast enough to be used with realtime video.
- It has two levels API: high-level for quick and simple decoding and low-level for those, who wants to control everything.
- It is small, and doesn't have any dependencies (except of the C library, of course).
- It has a very small memory footprint.
- It works under the CPython 2.5, 2.6, 2.7 and PyPy. Python 3k support is planned.

Contents:

CHAPTER

ONE

INSTALL

Install it with **pip**:

```
pip install quirc
```

Or with *setuptools*:

```
easy_install -U quirc
```

1.1 Requirements

This library optionally requires the image processing library for reading images.

At now, the only one supported library is a `PIL` or it's fork `Pillow`.

Ability to use any other libraries, such a `ImageMagick`, is in the future plans.

CHAPTER

TWO

USAGE

Library' usage splits into a two parts: *high-level* and *low-level* API.

In a sunny and beautiful world you will need only a *high-level* API.

For decoding images from the video streaming in the realtime use Streaming decoder (*TODO: link*).

But if you are have not so much free memory or want to control whole the process, use a *low-level* API.

2.1 High-level API

It is very simple in use:

```
import quirc
from PIL import Image

for code in quirc.decode(Image.open('images/qr-code.png')):
    print code
```

2.2 Streaming

In case if you are detect QR codes from video stream or something else, creating all required structures for each frame is very expensive. In that case you can operate Low-level API (*TODO: Link*) or use `quirc.Decoder`.

2.2.1 Usage

Create new decoder and supply to it frame' width and height: `quirc.Decoder(640, 480)`.

For each frame call decoder method `decode()` and apply to it raw pixels binary data.

Attention! Streaming decoder suppress `quirc.DecodeException`, because if current frame cannot be decoded, it must not crash a program.

```
import quirc

# Create new decoder with frame size
decoder = quirc.Decoder(640, 480)

while True:
```

```
    # Use OpenCV or something else here
    # In this example, here is a pseudo-code
    image = get_new_frame_from_camera()

    # Convert it to the grayscale, so each byte will represent one pixel
    image.convert('grayscale')

    for code in decoder.raw(image.to_string()):
        print code.text
```

2.3 Low-level API

Low-level API fully copies the C API and is contained in the `quirc.api` module.

See example for workflow.

Warning: you will need to use `ctypes` here manually.

2.3.1 Functions

Initialization

Recognition

Extraction

Miscellaneous

2.3.2 Classes

2.3.3 Example

```
from quirc import api
from PIL import Image

image = Image.open('images/qr-code.png')
width, height = image.size

# Convert image to the grayscale mode
if not image.mode in ('L', '1'):
    image = image.convert('L')
pixels = image.load()

obj = api.new()
api.resize(obj, width, height)

buffer = api.begin(obj, width, height)

# Fill buffer with a image pixels. One cell, one pixel.
idx = 0
for i in range(width):
    for j in range(height):
        buffer[idx] = ctypes.c_uint8(pixels[j, i])
```

```
        idx += 1

# Finish image recognition
api.end(obj)

code = api.structures.Code()
data = api.structures.Data()

for i in range(api.count(obj)):
    api.extract(obj, i, code)
    api.decode(code, data)

    print ctypes.string_at(data.payload, data.payload_len)

api.destroy(obj)
```

2.4 Data format

All functions of the library receives image data in a binary format. There is some rules that must be applied to your data:

- Data length must be equal to the `width * height` of the image.
- Each byte **must** represent one pixel, so you will need manually to convert it to the *grayscale* or *black&white* mode.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*