CHAPTER 11: Make a Barcode Reader in Python

Learn how to make a barcode scanner that decodes barcodes and draw them in the image using pyzbar and OpenCV libraries in Python

A barcode is a method of representing data in a visual and machine-readable form, it consists of bars and spaces. Today, we see barcodes everywhere, especially in products in supermarkets.

Barcodes can be read by an optical barcode scanner, but in this tutorial, we will make a script in Python that is able to read and decode barcodes, as well as a drawing where they're located in a given image.

Related: *How to Extract Frames from Video in Python.*

To get started, we need to install few libraries:

```
pip3 install pyzbar opencv-python
```

Once you have these installed, open up a new Python file and import them:

```
from pyzbar import pyzbar
import cv2
```

I have few images to test with, you can use any image you want from the internet or your own disk, but you can get my test images in this directory.

I have wrapped every functionality into a function, the first function we gonna discuss is the following:

```
def decode(image):
    # decodes all barcodes from an image
    decoded_objects = pyzbar.decode(image)
    for obj in decoded_objects:
        # draw the barcode
        print("detected barcode:", obj)
        image = draw_barcode(obj, image)
        # print barcode type & data
        print("Type:", obj.type)
        print("Data:", obj.data)
        print()

    return image
```

`decode()` function takes an image as a numpy array, and uses `pyzbar.decode()` that is responsible for decoding all barcodes from a single image and returns a bunch of useful information about each barcode detected.

We then iterate over all detected barcodes and draw a rectangle around the barcode and prints the type and the data of the barcode.

To make things clear, the following is how each `obj` looked like if we print it:

```
Decoded(data=b'43770929851162', type='I25', rect=Rect(left=62, top=0, width=694,
height=180), polygon=[Point(x=62, y=1), Point(x=62, y=179), Point(x=756, y=180),
Point(x=756, y=0)])
```

So `pyzbar.decode()` function returns the data containing the barcode, the type of barcode, as well as the location points as a rectangle and a polygon.

This brings us to the next function that we used, `draw_barcode()`:

```
def draw_barcode(decoded, image):
    # n_points = len(decoded.polygon)
    # for i in range(n_points):
    #     image = cv2.line(image, decoded.polygon[i], decoded.polygon[(i+1) % n_points], color=(0, 255, 0), thickness=5)
    # uncomment above and comment below if you want to draw a polygon and not a rectangle
    image = cv2.rectangle(image, (decoded.rect.left, decoded.rect.top),
                          (decoded.rect.left + decoded.rect.width, decoded.rect.top + decoded.rect.height),
                          color=(0, 255, 0),
                          thickness=5)
    return image
```

This function takes the decoded object we just saw, and the image itself, it draws a rectangle around the barcode using `cv2.rectangle()` function, or you can uncomment the other version of the function; drawing the polygon using `cv2.line()` function, the choice is yours. I preferred the rectangle version.

Finally, it returns the image that contains the drawn barcodes. Now let's use these functions for our example images:

```
if __name__ == "__main__":
    from glob import glob

    barcodes = glob("barcode*.png")
```

```
for barcode_file in barcodes:
    # load the image to opencv
    img = cv2.imread(barcode_file)
    # decode detected barcodes & get the image
    # that is drawn
    img = decode(img)
    # show the image
    cv2.imshow("img", img)
    cv2.waitKey(0)
```

In my current directory, I have barcode1.png, barcode2.png, and barcode3.png, which are all example images of a scanned barcode, I used glob so I can get all these images as a list and iterate over them.

On each file, we load it using `cv2.imread()` function, and use the previously discussed `decode()` function to decode the barcodes and then we show the actual image.

Note that this will also detect QR codes, and that's fine, but for more accurate results, I suggest you check the dedicated tutorial for detecting and generating qr codes in Python.

When I run the script, it shows each image and prints the type and data of it, press any key and you'll get the next image, here is my output:

```
detected barcode: Decoded(data=b'0036000291452', type='EAN13', rect=Rect(left=124, top=58, width=965, height=812), polygon=[Point(x=124, y=59), Point(x=124, y=869), Point(x=621, y=870), Point(x=1089, y=870), Point(x=1089, y=58)])
Type: EAN13
Data: b'0036000291452'
```

```
detected barcode: Decoded(data=b'Wikipedia', type='CODE128', rect=Rect(left=593, top=4, width=0, height=294), polygon=[Point(x=593, y=4), Point(x=593, y=298)])

Type: CODE128

Data: b'Wikipedia'


detected barcode: Decoded(data=b'43770929851162', type='I25', rect=Rect(left=62, top=0, width=694, height=180), polygon=[Point(x=62, y=1), Point(x=62, y=179), Point(x=756, y=180), Point(x=756, y=0)])

Type: I25

Data: b'43770929851162'
```

Here is the last image that is shown:

Conclusion

That is awesome, now you have a great tool to make your own barcode scanner in Python. I know you all want to read directly from the camera, as a result, I have prepared the code that reads from the camera and detects barcodes in a live manner, check it here!

You can also add some sort of a beep when each barcode is detected, just like in supermarkets, check the tutorial for playing sounds that may help you accomplish that.

For more detailed information, I invite you to check pyzbar documentation.

Finally, if you're a beginner and want to learn Python, I suggest you take the Python For Everybody Coursera course, in which you'll learn a lot about Python. You can also check our resources and courses page to see the Python resources I recommend on various topics!

CHAPTER 12: Play and Record Audio in Python

Learn how to play and record sound files using different libraries such as playsound, Pydub and PyAudio in Python.

Many of the applications out there record your voice as well as playing sounds, if you want to do that as well, then you came into the right place, in this tutorial, we will be using different Python libraries to play and record audio in Python.

Let's install the required libraries for this tutorial:

```
pip3 install playsound pyaudio pydub ffmpeg-python
```

Audio Player

First, we gonna start with the most straightforward module here, playsound:

```
from playsound import playsound
```

```
playsound("audio_file.mp3")
```

Yes, that's it for this module. It is basically a pure Python, cross-platform, single function module. The documentation says that WAV and MP3 extensions are known to work, and they may work for other formats as well.

playsound() function plays the sound in the audio file and blocks until the file reading is completed, you can pass block=False to make the function run asynchronously.

Another alternative is to use the Pydub library:

```
from pydub import AudioSegment
from pydub.playback import play

# read MP3 file
song = AudioSegment.from_mp3("audio_file.mp3")
# song = AudioSegment.from_wav("audio_file.wav")
# you can also read from other formats such as MP4
# song = AudioSegment.from_file("audio_file.mp4", "mp4")
play(song)
```

***Note**: You need FFmpeg installed on your machine in order to use AudioSegment.from_file() function that supports all formats that are supported by FFmpeg.*

Pydub is quite a popular library, as it isn't only for playing sound, you can use it for different purposes, such as converting audio files, slicing audio, boosting or reducing volume, and much more, check their repository for more information.

If you wish to play audio using PyAudio, check this link.

Related: *How to Extract Audio from Video in Python.*

Audio Recorder

To record voice, we gonna use the PyAudio library, as it is the most convenient approach:

```
import pyaudio
import wave

# the file name output you want to record into
filename = "recorded.wav"
# set the chunk size of 1024 samples
chunk = 1024
# sample format
FORMAT = pyaudio.paInt16
# mono, change to 2 if you want stereo
channels = 1
# 44100 samples per second
sample_rate = 44100
record_seconds = 5
# initialize PyAudio object
p = pyaudio.PyAudio()
# open stream object as input & output
stream = p.open(format=FORMAT,
                channels=channels,
                rate=sample_rate,
                input=True,
                output=True,
                frames_per_buffer=chunk)
frames = []
```

```
print("Recording...")
for i in range(int(sample_rate / chunk * record_seconds)):
    data = stream.read(chunk)
    # if you want to hear your voice while recording
    # stream.write(data)
    frames.append(data)
print("Finished recording.")
# stop and close stream
stream.stop_stream()
stream.close()
# terminate pyaudio object
p.terminate()
# save audio file
# open the file in 'write bytes' mode
wf = wave.open(filename, "wb")
# set the channels
wf.setnchannels(channels)
# set the sample format
wf.setsampwidth(p.get_sample_size(FORMAT))
# set the sample rate
wf.setframerate(sample_rate)
# write the frames as bytes
wf.writeframes(b"".join(frames))
# close the file
wf.close()
```

The above code basically initializes the PyAudio object, and then we open up a stream object that allows us to record from the microphone using stream.read() method. After we finish recording, we use the built-in wave module to write that WAV audio file into the disk.

When you set input=True in the p.open() method, you will be able to use stream.read() to read from the microphone. Also, when you set output=True, you'll be able to use stream.write() to write to the speaker.

Conclusion

Alright, in this tutorial, you learned how you can play audio files using playsound, Pydub, and PyAudio libraries as well as recording voice using PyAudio.

A great challenge for you is to combine this with a screen recorder, and you'll come up with a Python tool that records your voice and screen simultaneously. You will need to use a thread that records audio and another one for the screen recorder, good luck with that!

Learn More with Courses

Finally, many of the Python and audio signal processing concepts aren't discussed in detail here, if you feel you want to dig more into Python and signal processing, I highly suggest you get these courses:

- Python for Everybody Course
- Audio Signal Processing Course

CHAPTER 13: Generate and Read QR Code in Python

Learning how you can generate and read QR Code in Python using qrcode and OpenCV libraries.

QR code is a type of matrix barcode that is a machine-readable optical label that contains information about the item to which it is attached. In practice, QR codes often contain data for a locator, identifier, or tracker that points to a website or application, etc.

In this tutorial, you will learn how to generate and read QR codes in Python using qrcode and OpenCV libraries.

Installing required dependencies:

```
pip3 install opencv-python qrcode numpy
```

Generate QR Code

First, let's start by generating QR codes, it is basically straightforward using qrcode library:

```
import qrcode
# example data
data = "https://www.thepythoncode.com"
# output file name
filename = "site.png"
# generate qr code
```

```
img = qrcode.make(data)
# save img to a file
img.save(filename)
```

This will generate a new image file in the current directory with the name of "site.png", which contains a QR code image of the data specified (in this case, this website URL), will look something like this:

You can also use this library to have full control with QR code generation using the `qrcode.QRCode()` class, in which you can instantiate and specify the size, fill color, back color, and error correction, like so:

```
import qrcode
import numpy as np
# data to encode
data = "https://www.thepythoncode.com"
# instantiate QRCode object
qr = qrcode.QRCode(version=1, box_size=10, border=4)
# add data to the QR code
qr.add_data(data)
# compile the data into a QR code array
```

```
qr.make()
# print the image shape
print("The shape of the QR image:", np.array(qr.get_matrix()).shape)
# transfer the array into an actual image
img = qr.make_image(fill_color="white", back_color="black")
# save it to a file
img.save("site_inversed.png")
```

So in the creation of `QRCode` class, we specify the `version` parameter, which is an integer from 1 to 40 that controls the size of the QR code image (1 is small, 21x21 matrix, 40 is 185x185 matrix), but this will be overwritten when the data doesn't fit the size you specify. In our case, it will scale up to version 3 automatically.

`box_size` parameter controls how many pixels each box of the QR code is, whereas the `border` controls how many boxes thick the border should be.

We then add the data using the `qr.add_data()` method, compiles it to an array using `qr.make()` method, and then make the actual image using `qr.make_image()` method. We specified white as the `fill_color` and black as the `back_color`, which is the exact opposite of the default QR code, check it out:

 And the shape of the image was indeed scaled up and wasn't 21x21:

```
The shape of the QR image: (37, 37)
```

Related: *How to Make a Barcode Reader in Python.*

Read QR Code

There are many tools that read QR codes. However, we will be using OpenCV for that, as it is popular and easy to integrate with the webcam or any video.

Alright, open up a new Python file and follow along with me, let's read the image that we just generated:

```
import cv2
# read the QRCODE image
img = cv2.imread("site.png")
```

Luckily for us, OpenCV already got QR code detector built-in:

```
# initialize the cv2 QRCode detector
detector = cv2.QRCodeDetector()
```

We have the image and the detector, let's detect and decode that data:

```
# detect and decode
data, bbox, straight_qrcode = detector.detectAndDecode(img)
```

The detectAndDecode() function takes an image as an input and decodes it to return a tuple of 3 values: the data decoded from the QR code, the output array of vertices of the

found QR code quadrangle, and the output image containing rectified and binarized QR code.

We just need data and bbox here, bbox will help us draw the quadrangle in the image and data will be printed to the console!

Let's do it:

```
# if there is a QR code
if bbox is not None:
    print(f"QRCode data:\n{data}")
    # display the image with lines
    # length of bounding box
    n_lines = len(bbox)
    for i in range(n_lines):
        # draw all lines
        point1 = tuple(bbox[i][0])
        point2 = tuple(bbox[(i+1) % n_lines][0])
        cv2.line(img, point1, point2, color=(255, 0, 0), thickness=2)
```

cv2.line() function draws a line segment connecting two points, we retrieve these points from bbox array that was decoded by detectAndDecode() previously. we specified a blue color ((255, 0, 0) is blue as OpenCV uses BGR colors) and thickness of 2.

Finally, let's show the image and quit when a key is pressed:

```
# display the result
cv2.imshow("img", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Once you run this, the decoded data is printed:

```
QRCode data:
https://www.thepythoncode.com
```

And the following image is shown:

As you can see, the blue lines are drawn in the exact QR code borders. Awesome, we are done with this script, try to run it with different data and see your own results!

Note that this is ideal for QR codes and not for barcodes, If you want to read barcodes, check this tutorial that is dedicated to that!

If you want to detect and decode QR codes live using your webcam (and I'm sure you do), here is a code for that:

```
import cv2
# initalize the cam
cap = cv2.VideoCapture(0)
# initialize the cv2 QRCode detector
detector = cv2.QRCodeDetector()
```

```
while True:
    _, img = cap.read()
    # detect and decode
    data, bbox, _ = detector.detectAndDecode(img)
    # check if there is a QRCode in the image
    if bbox is not None:
        # display the image with lines
        for i in range(len(bbox)):
            # draw all lines
            cv2.line(img, tuple(bbox[i][0]), tuple(bbox[(i+1) % len(bbox)][0]), color=(255, 0, 0), thickness=2)
        if data:
            print("[+] QR Code detected, data:", data)
    # display the result
    cv2.imshow("img", img)
    if cv2.waitKey(1) == ord("q"):
        break
cap.release()
cv2.destroyAllWindows()
```

Awesome, we are done with this tutorial, you can now integrate this into your own applications!

Check qrcode's official documentation.

Finally, if you're a beginner and want to learn Python, I suggest you take the Python For Everybody Coursera course, in which you'll learn a lot about Python, good luck!

CHAPTER 14: Record your Screen in Python

Using pyautogui and OpenCV to record display screen video and save it to a file in Python.

Screen recording enables you to create demonstration videos, record gaming achievements, and create videos that can be shared online on social media. Many industrial software exists out there that can help you do that very easily though. In this tutorial, you will learn how to make your own simple screen recorder in Python that you can further extend to your own needs.

This tutorial is about recording your whole screen. If you want to record a specific window, then check this tutorial, as it's exactly what you need.

Related: *How to Use Steganography to Hide Secret Data in Images in Python*.

Let's get started, first, install the required dependencies for this tutorial:

```
pip3 install numpy opencv-python pyautogui
```

The process of this tutorial is as follows:

- Capture a screenshot using pyautogui.
- Convert that screenshot to a numpy array.
- Write that numpy array to a file with the proper format using a video writer in OpenCV.

Importing necessary modules:

```
import cv2
import numpy as np
import pyautogui
```

Let's initialize the format we gonna use to write our video file (named "output.avi"):

```
# display screen resolution, get it using pyautogui itself
SCREEN_SIZE = tuple(pyautogui.size())
# define the codec
fourcc = cv2.VideoWriter_fourcc(*"XVID")
# frames per second
fps = 12.0
# create the video write object
out = cv2.VideoWriter("output.avi", fourcc, fps, (SCREEN_SIZE))
# the time you want to record in seconds
record_seconds = 10
```

We're getting the screen resolution from the `pyautogui.size()` function which returns the height and the width of the screen as a two-integer tuple.

fourcc is the video codec library that OpenCV will use to write the video file, we specified XVID here. We also set the FPS to `12.0` and pass these parameters to the `cv2.VideoWriter()` object.

The `record_seconds` is the number of seconds you want to record, if you want to record until pressing the **"q"** button, then simply set this to a large value.

Now we need to keep capturing screenshots and writing to the file in a loop until the seconds are passed or the user clicks the **"q"** button, here is the main loop for that:

```
for i in range(int(record_seconds * fps)):
    # make a screenshot
    img = pyautogui.screenshot()
    # convert these pixels to a proper numpy array to work with OpenCV
    frame = np.array(img)
    # convert colors from BGR to RGB
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    # write the frame
    out.write(frame)
    # show the frame
    cv2.imshow("screenshot", frame)
    # if the user clicks q, it exits
    if cv2.waitKey(1) == ord("q"):
        break

# make sure everything is closed when exited
cv2.destroyAllWindows()
out.release()
```

First, we use the screenshot() function which returns an image object, so we need to convert it to a proper numpy array. After that, we need to convert that frame into **RGB**, that's because OpenCV uses **BGR** by default.

The loop will finish once the seconds are passed or if you hit the **"q"** button. Note that if you increase the FPS, you may see the video is a bit sped up, that's because your CPU can't handle that speed.

If you don't want to see your record in real-time, simply comment out the `cv2.imshow()` line and it won't show anything until the recording is finished.

As mentioned in the pyautogui's official documentation, you can also record only regions of your screen, by passing the `region` keyword argument which is a four-integer tuple representing the **top**, **left**, **width,** and **height** of the region to capture, here is how it's done:

```
img = pyautogui.screenshot(region=(0, 0, 300, 400))
```

After you are done with recording, just click **"q"**, it will destroy the window and finish writing to the file, try it out!

Alright, there are endless ideas you can use to extend this. For example, you can combine this with an audio recorder, and you'll come up with a Python tool that records your screen and voice simultaneously, you will need to use a thread that records audio and another for the screen recorder. Or you can add an audio file to your video recording.

Or you can create keyboard shortcuts that start, pause, and stop the recording, this tutorial can help you.

Courses & Resources

Finally, if you're a beginner and want to learn Python, I suggest you take the Python For Everybody Coursera course, in which you'll learn a lot about Python, good luck!