**An OpenCV barcode and QR code scanner with ZBar or PyZbar**

## An OpenCV barcode and QR code scanner with ZBar

First we will show how to install the [ZBar library](http://zbar.sourceforge.net/" \t "_blank) (with Python bindings).

The ZBar library will be used together with OpenCV to scan and decode barcodes and QR codes.

Once ZBar and OpenCV are properly configured, then we can do how to scan barcodes and QR codes in a single image.

Starting with a single image will give us the practice we need to prepare for the next step: Reading barcodes and QR codes in real-time with OpenCV and ZBar,

Finally, we will do how to deploy our real-time barcode scanner to the Raspberry Pi.

### Installing ZBar (with Python bindings) for barcode decoding

The instructions to install ZBar and zbar  bindings themselves, ensuring we can:

1. **Use Python 3** (the official zbar  Python bindings only support Python 2.7)
2. **Detect and localize** exactly where in the image the barcode is.

Installing the necessary software is an easy process.

**Step 1:** Install zbar  from the apt  or brew  repository

**Installing ZBar for Ubuntu or Raspbian**

Installing ZBar for Ubuntu can be accomplished with the following command:

An OpenCV barcode and QR code scanner with ZBar

Shell

|  |  |
| --- | --- |
| 1 | $ sudo apt-get install libzbar0 |
|  |  |

**Step 2 (Optional):** Create a virtual environment and install OpenCV

You have two options here:

1. Use an existing virtual environment that has OpenCV ready to go (skip this step and head to **Step 3**).
2. Or create a new, isolated virtual environment which involves installing OpenCV.

Virtual environments are a best practice for Python development and I highly encourage you to make use of them.

I suggest to create a new, isolated Python 3 virtual environment and followed the Ubuntu OpenCV installation instructions. The only change I made while following those instructions was to name my environment barcode :

An OpenCV barcode and QR code scanner with ZBar

|  |  |
| --- | --- |
| 1 | $ mkvirtualenv barcode -p python3 |

***Note:*** If you already have OpenCV installed on your system you can skip the OpenCV compile process and simply sym-link your *cv2.so*  bindings into the *site-packages*  directory of your new Python virtual environment.

**Step 3:** Install pyzbar

Now that I have a Python 3 virtual environment named barcode  on my machine, I activated the barcode  environment (yours might have a different name) and installed pyzbar :

|  |  |
| --- | --- |
| 1  2 | $ workon barcode  $ pip install pyzbar |

If you are not using a Python virtual environment you can just do:

|  |  |
| --- | --- |
| 1 | $ pip install pyzbar |

If you’re trying to install pyzbar  into the system version of Python make sure you use the sudo  command as well.

Decoding barcodes and QR codes with OpenCV in single images

**Figure 1:** Both QR and 1D barcodes can be read with our Python app using ZBar + OpenCV.

Before we implement real-time barcode and QR code reading, let’s first start with a single image scanner to get our feet wet.

Open up a new file, name it barcode\_scanner\_image.py  and insert the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # import the necessary packages  from pyzbar import pyzbar  import argparse  import cv2    # construct the argument parser and parse the arguments  ap = argparse.ArgumentParser()  ap.add\_argument("-i", "--image", required=True,  help="path to input image")  args = vars(ap.parse\_args()) |

On **Lines 2-4** we import our required packages.

Both pyzbar  and cv2  (OpenCV) need to be installed following the instructions in the previous section.

In contrast, argparse  is included with the Python install and it is responsible for parsing command line arguments.

We have one required command line argument for this script ( --image ) and it is parsed on **Lines 7-10**.

You’ll see at the end of this section how to run the script while passing a [command line argument](https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/) containing the input image path.

Now, let’s take the input image and put pyzbar  to work:

|  |  |
| --- | --- |
| 12  13  14  15  16 | # load the input image  image = cv2.imread(args["image"])    # find the barcodes in the image and decode each of the barcodes  barcodes = pyzbar.decode(image) |

On **Line 13**, we load the input image  via its path (contained in our convenient args  dictionary).

From there, we call pyzbar.decode  to find and decode the barcodes  in the image  (**Line 16**). This is where all the magic of ZBar happens.

We aren’t finished yet — now we need to parse the information contained within the barcodes  variable:

|  |  |
| --- | --- |
| 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | # loop over the detected barcodes  for barcode in barcodes:  # extract the bounding box location of the barcode and draw the  # bounding box surrounding the barcode on the image  (x, y, w, h) = barcode.rect  cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)    # the barcode data is a bytes object so if we want to draw it on  # our output image we need to convert it to a string first  barcodeData = barcode.data.decode("utf-8")  barcodeType = barcode.type    # draw the barcode data and barcode type on the image  text = "{} ({})".format(barcodeData, barcodeType)  cv2.putText(image, text, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX,  0.5, (0, 0, 255), 2)    # print the barcode type and data to the terminal  print("[INFO] Found {} barcode: {}".format(barcodeType, barcodeData))    # show the output image  cv2.imshow("Image", image)  cv2.waitKey(0) |

Beginning on **Line 19**, we loop over the detected barcodes .

In this loop, we proceed to:

1. Extracting the bounding box (x, y)-coordinates from the barcode.rect  object (**Line 22**), enabling us to localize and determine where in the input image the current barcode is.
2. Draw a bounding box rectangle on the image  around the detected barcode  (**Line 23**).
3. Decode the barcode  into a "utf-8"  string and extract the type of barcode (**Lines 27 and 28**). It is critical to call the .decode("utf-8")  function on the object to convert from a byte array to a string. Format and draw the barcodeData  and barcodeType  on the image (**Lines 31-33**).
4. And finally, output the same data and type information to the terminal for debugging purposes (**Line 36**).

Let’s test our OpenCV barcode scanner. As you can see in the terminal, all four of the barcodes were found and properly decoded!

### Real-time barcode and QR code reading with OpenCV

In the previous section, we learned how to create a Python + OpenCV barcode scanner for single images.

Our barcode and QR code scanner worked well — but it raises the question, can we detect and decode barcode + QR codes in real-time?

To find out, open up a new file, name it barcode\_scanner\_video.py , and insert the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # import the necessary packages  from imutils.video import VideoStream  from pyzbar import pyzbar  import argparse  import datetime  import imutils  import time  import cv2    # construct the argument parser and parse the arguments  ap = argparse.ArgumentParser()  ap.add\_argument("-o", "--output", type=str, default="barcodes.csv",  help="path to output CSV file containing barcodes")  args = vars(ap.parse\_args()) |

On **Lines 2-8** we import our required packages.

At this point, recalling the above explanation, you should recognize pyzbar , argparse , and cv2 .

We’ll also use VideoStream  to handle capturing video frames in an efficient, threaded approach and if you do not have imutils  installed on your system, just use the following command:

|  |  |
| --- | --- |
| 1 | $ pip install imutils |

We’re going to parse one optional command line argument, --output , which contains the path to the output Comma Separated Values (CSV) file. This file will contain the timestamp and payload of each detected and decoded barcode from our video stream. If this argument isn’t specified, the CSV file will be placed in our current working directory with a name of "barcodes.csv"  (**Lines 11-14**).

From there, let’s initialize our video stream and open our CSV file:

|  |  |
| --- | --- |
| 16  17  18  19  20  21  22  23  24  25 | # initialize the video stream and allow the camera sensor to warm up  print("[INFO] starting video stream...")  # vs = VideoStream(src=0).start()  vs = VideoStream(usePiCamera=True).start()  time.sleep(2.0)    # open the output CSV file for writing and initialize the set of  # barcodes found thus far  csv = open(args["output"], "w")  found = set() |

On **Lines 18 and 19** we initialize and start our VideoStream . You may either:

* Use your USB webcam (uncomment **Line 18** and comment **Line 19**)
* Or if you’re using a Raspberry Pi (like me) you can use the PiCamera (uncomment **Line 19** and comment **Line 18**).

I chose to use my Raspberry Pi PiCamera, as is shown in the next section.

We then pause for two seconds to allow the camera can warm up (**Line 20**).

We’ll be writing all barcodes we find to disk in a CSV file (but ensuring duplicates are not written). **We can** do whatever we want once a barcode is detected and read such as:

* Save it in a SQL database
* Send it to a server
* Upload it to the cloud
* Send an email or text message

**The actual action is arbitrary** — we’re simply using the CSV file as an example.

We open the csv  file for writing on **Line 24**. If you are modifying the code to append to the file, you can simply change the 2nd parameter from "w"  to "a"  (but you’ll have to search the file for duplicates in a different way).

We also initialize a set  for found  barcodes. This set will contain unique barcodes while preventing duplicates.

Let’s begin capturing + processing frames:

|  |  |
| --- | --- |
| 27  28  29  30  31  32  33  34  35 | # loop over the frames from the video stream  while True:  # grab the frame from the threaded video stream and resize it to  # have a maximum width of 400 pixels  frame = vs.read()  frame = imutils.resize(frame, width=400)    # find the barcodes in the frame and decode each of the barcodes  barcodes = pyzbar.decode(frame) |

On **Line 28** we start our loop and proceed to grab and and resize a  frame  from our video stream (**Lines 31 and 32**).

From there, we call pyzbar.decode  to detect and decode any QR + barcodes in the frame .

Let’s proceed to loop over the detected barcodes :

|  |  |
| --- | --- |
| 37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | # loop over the detected barcodes  for barcode in barcodes:  # extract the bounding box location of the barcode and draw  # the bounding box surrounding the barcode on the image  (x, y, w, h) = barcode.rect  cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)    # the barcode data is a bytes object so if we want to draw it  # on our output image we need to convert it to a string first  barcodeData = barcode.data.decode("utf-8")  barcodeType = barcode.type    # draw the barcode data and barcode type on the image  text = "{} ({})".format(barcodeData, barcodeType)  cv2.putText(frame, text, (x, y - 10),  cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)    # if the barcode text is currently not in our CSV file, write  # the timestamp + barcode to disk and update the set  if barcodeData not in found:  csv.write("{},{}\n".format(datetime.datetime.now(),  barcodeData))  csv.flush()  found.add(barcodeData) |

This loop should look very familiar if you read the previous section.

In fact, **Lines 38-52**are identical to those the single image script. Please refer to the single image barcode detection and scanning section for a detailed review of this code block.

**Lines 56-60** are new. On these lines, we check if we’ve found a unique (not previously found) barcode (**Line 56**).

If that’s the case, we write the timestamp and data to the csv  file (**Lines 57-59**). We also append the barcodeData  to a found  set as a simple way to handle duplicates.

In the remaining lines of the real-time barcode scanner script, we display the frame, check if the quit key is pressed, and perform cleanup:

|  |  |
| --- | --- |
| 62  63  64  65  66  67  68  69  70  71  72  73  74 | # show the output frame  cv2.imshow("Barcode Scanner", frame)  key = cv2.waitKey(1) & 0xFF    # if the `q` key was pressed, break from the loop  if key == ord("q"):  break    # close the output CSV file do a bit of cleanup  print("[INFO] cleaning up...")  csv.close()  cv2.destroyAllWindows()  vs.stop() |

On **Line 63** we display the output frame .

Then on **Lines 64-68**, we check for keys and if "q"  is pressed, we break  out of the main execution loop.

Finally, we perform cleanup on **Lines 72-74**.

Building a barcode and QR code scanner on the Raspberry Pi

If you’re looking to build your own with the exact peripherals shown, I’ve listed the products and links:

* Respberry Pi 3(Latest One)
* Respberry Pi Camera
* Touch Screen LCD Display

Building the system is really easy and I’ve made step-by-step instructions

From there, open up a terminal on your Pi and launch the app with the following command (you’ll need a keyboard/mouse for this step but then you can disconnect and let the app run):

|  |  |
| --- | --- |
| 1  2 | $ python barcode\_scanner\_video.py  [INFO] starting video stream... |

Now you can present barcodes to the camera and when you’re done, you can open the barcodes.csv  file (or if you’re so inclined you can execute tail -f barcodes.csv  in a separate terminal to view the data as it enters the CSV file live).

I tried different angle and got success! It even works at many angles.

Now let’s try a QR code that is contains a JSON-blob of data:

No match for my OpenCV + ZBar + Python barcode scanner project!

And finally, I tried a traditional 1-D barcode:

1-D barcodes are slightly more challenging for the system especially with a PiCamera which doesn’t support autofocus. That being said, I achieved a successful detection and decoding of this barcode as well.

We can bring a USB webcam such as the Logitech C920 which has great autofocus. Alternatively, you can actually change the factory focus on your PiCamera using the method that Jeff Geerling. Try to google it.