



Increasing webcam FPS with Python and OpenCV

by **Adrian Rosebrock** on December 21, 2015 in **Libraries, Tutorials**

Like 4



```
(cv)malcolm:test_fps adrianrosebrock$ python fps_demo.py
[INFO] sampling frames from webcam...
[INFO] elapsed time: 3.34
[INFO] approx. FPS: 29.97
Cleaned up camera.
[INFO] sampling THREADED frames from webcam...
[INFO] elapsed time: 0.70
[INFO] approx. FPS: 143.71
Cleaned up camera.
(cv)malcolm:test_fps adrianrosebrock$
```

The image shows a terminal window titled 'test_fps — bash — 124x32'. It displays the output of a Python script 'fps_demo.py'. The script first samples frames from a webcam in a non-threaded manner, taking 3.34 seconds and achieving approximately 29.97 FPS. After cleaning up, it samples frames using a threaded approach, which takes only 0.70 seconds and achieves approximately 143.71 FPS. The terminal window has a dark background and is framed by a light grey border with standard macOS window controls.

Over the next few weeks, I'll be doing a series of blog posts on how to **improve your frames per second (FPS) from your webcam** using Python, OpenCV, and threading.

Using threading to handle I/O-heavy tasks (such as reading frames from a camera sensor) is a programming model that has existed for decades.

For example, if we were to build a web crawler to spider a series of webpages (a task that is, by definition, I/O bound), our main program would spawn *multiple threads* to handle downloading the set of pages *in parallel* instead of relying on only a *single thread* (our “main thread”) to download the pages in *sequential order*. Doing this allows us to spider the webpages substantially faster.

The same notion applies to computer vision and reading frames from a camera — **we can improve our FPS simply by creating a new thread that does nothing but poll the camera for new frames while our main thread handles processing the current frame**

This is a simple concept, but it's one that's rarely seen in a lot of code (or sometimes *a lot* of lines, depending on your

Free 17-day crash course on
Computer Vision, OpenCV, and Deep
Learning

make your program harder to debug, but once you get it right, you can dramatically improve your FPS.

We'll start off this series of posts by writing a threaded Python class to access your webcam or USB camera using OpenCV.

Next week we'll use threads to improve the FPS of your Raspberry Pi and the [picamera](#) module.

Finally, we'll conclude this series of posts by creating a class that unifies *both* the threaded webcam/USB camera code and the threaded [picamera](#) code into a **single class**, making all webcam/video processing examples on PyImageSearch *not only run faster, but run on either your laptop/desktop or the Raspberry Pi **without changing a single line of code!***

Looking for the source code to this post?
Jump right to the downloads section.

Use threading to obtain higher FPS

The “secret” to obtaining higher FPS when processing video (moving the reading of frames from the camera sensor) to a separate thread.

You see, accessing your webcam/USB camera using the `cv2.VideoCapture()` method is a *blocking operation*. The main thread of our Python script waits until the frame is read from the camera device and returned to the main thread.

I/O tasks, as opposed to CPU bound operations, tend to be slow. Video processing applications are certainly quite CPU heavy (especially when using OpenCV), but camera I/O can be a huge bottleneck as well.

As we'll see later in this post, just by adjusting the the camera I/O, we can achieve **much as 379%!**

Of course, this isn't a *true increase* of FPS as it is a *dramatic reduction in latency* (i.e., a frame is always available for processing; we don't need to poll the camera device and wait for the I/O to complete). Throughout the rest of this post, I will refer to our metrics as an “FPS increase” for brevity, but also keep in mind that it's a *combination* of both a decrease in latency and an increase in FPS.

In order to accomplish this FPS increase/latency decrease, our goal is to move the reading of frames from a webcam or USB device to an *entirely different thread, totally separate from our main Python script*.

This will allow frames to be read *continuously* from the I/O thread, all while our root thread processes the current frame. Once the root thread has finished processing its frame, it simply needs to grab the current frame from the I/O thread. This is accomplished *without* having to wait for the I/O thread to finish.

The first step in implementing our threaded video stream is to create a class that will help us to measure our frames per second. This class will help us to

Free 17-day crash course on Computer Vision, OpenCV, and Deep Learning

Interested in computer vision, OpenCV, and deep learning, but don't know where to start? Let me help. I've created a *free*, 17-day crash course that is hand-tailored to give you the best possible introduction to computer vision and deep learning. Sound good? Enter your email below to get started.

START MY EMAIL COURSE

indeed increase FPS.

We'll then define a `WebcamVideoStream` class that will access our webcam or USB camera in a threaded fashion.

Finally, we'll define our driver script, `fps_demo.py`, that will compare single threaded FPS to multi-threaded FPS.

Note: Thanks to [Ross Milligan](#) and his blog who inspired me to do this blog post.

Increasing webcam FPS with Python and OpenCV

I've actually already implemented webcam/USB camera and `picamera` threading inside the `imutils` library. However, I think a discussion of the implementation can greatly improve our knowledge of *how* and *why* threading increases FPS.

To start, if you don't already have `imutils` installed, you can install it using `pip` :

Increasing webcam FPS with Python and OpenCV	Shell
1 \$ <code>pip install imutils</code>	

Otherwise, you can upgrade to the latest version via:

Increasing webcam FPS with Python and OpenCV	Shell
1 \$ <code>pip install --upgrade imutils</code>	

As I mentioned above, the first step is to define a `FPS` class that we can use to approximate the frames per second of a given camera + computer vision processing pipeline:

Increasing webcam FPS with Python and OpenCV	Python
<pre>1 # import the necessary packages 2 import datetime 3 4 class FPS: 5 def __init__(self): 6 # store the start time, end time, and total number of frames 7 # that were examined between the start and end intervals 8 self._start = None 9 self._end = None 10 self._numFrames = 0 11 12 def start(self): 13 # start the timer 14 self._start = datetime.datetime.now() 15 return self 16 17 def stop(self): 18 # stop the timer 19 self._end = datetime.datetime.now() 20 21 def update(self): 22 # increment the total number of frames examined during the 23 # start and end intervals 24 self._numFrames += 1 25</pre>	

```

26     def elapsed(self):
27         # return the total number of seconds between the start and
28         # end interval
29         return (self._end - self._start).total_seconds()
30
31     def fps(self):
32         # compute the (approximate) frames per second
33         return self._numFrames / self.elapsed()

```

On **Line 5-10** we define the constructor to our `FPS` class. We don't require any arguments, but we do initialize three important variables:

- `_start` : The starting timestamp of when we commenced measuring the frame read.
- `_end` : The ending timestamp of when we stopped measuring the frame read.
- `_numFrames` : The total number of frames that were read during the `_start` and `_end` interval.

Lines 12-15 define the `start` method, which as the name suggests, kicks-off the timer.

Similarly, **Lines 17-19** define the `stop` method which grabs the ending timestamp.

The `update` method on **Lines 21-24** simply increments the number of frames that have been read during the starting and ending interval.

We can grab the total number of seconds that have elapsed between the starting and ending interval on **Lines 26-29** by using the `elapsed` method.

And finally, we can approximate the FPS of our camera + computer vision pipeline by using the `fps` method on **Lines 31-33**. By taking the total number of frames read during the interval and dividing by the number of elapsed seconds, we can obtain our estimated FPS.

Now that we have our `FPS` class defined (so we can empirically compare results), let's define the `WebcamVideoStream` class which encompasses the actual threaded camera read:

Increasing webcam FPS with Python and OpenCV	Python
<pre> 1 # import the necessary packages 2 from threading import Thread 3 import cv2 4 5 class WebcamVideoStream: 6 def __init__(self, src=0): 7 # initialize the video camera stream and read the first frame 8 # from the stream 9 self.stream = cv2.VideoCapture(src) 10 (self.grabbed, self.frame) = self.stream.read() 11 12 # initialize the variable used to indicate if the thread should 13 # be stopped 14 self.stopped = False </pre>	

We define the constructor to our `WebcamVideoStream` class on **Line 6**, passing in an (optional) argument: the `src` of the stream.

If the `src` is an *integer*, then it is presumed to be the *index* of the webcam/USB camera on your system. For example, a value of `src=0` indicates the *first* camera and a value of `src=1` indicates the *second* camera

hooked up to your system (provided you have a second one, of course).

If `src` is a *string*, then it assumed to be the *path* to a video file (such as .mp4 or .avi) residing on disk.

Line 9 takes our `src` value and makes a call to `cv2.VideoCapture` which returns a pointer to the camera/video file.

Now that we have our `stream` pointer, we can call the `.read()` method to poll the stream and grab the next available frame (**Line 10**). This is done strictly for initialization purposes so that we have an *initial* frame stored in the class.

We'll also initialize `stopped`, a boolean indicating whether the threaded frame reading should be stopped or not.

Now, let's move on to actually utilizing threads to read frames from our video stream using OpenCV:

Increasing webcam FPS with Python and OpenCV	Python
<pre> 16 def start(self): 17 # start the thread to read frames from the video stream 18 Thread(target=self.update, args=()).start() 19 return self 20 21 def update(self): 22 # keep looping infinitely until the thread is stopped 23 while True: 24 # if the thread indicator variable is set, stop the thread 25 if self.stopped: 26 return 27 28 # otherwise, read the next frame from the stream 29 (self.grabbed, self.frame) = self.stream.read() 30 31 def read(self): 32 # return the frame most recently read 33 return self.frame 34 35 def stop(self): 36 # indicate that the thread should be stopped 37 self.stopped = True </pre>	

Lines 16-19 define our `start` method, which as the name suggests, starts the thread to read frames from our video stream. We accomplish this by constructing a `Thread` object using the `update` method as the callable object invoked by the `run()` method of the thread.

Once our driver script calls the `start` method of the `WebcamVideoStream` class, the `update` method (**Lines 21-29**) will be called.

As you can see from the code above, we start an infinite loop on **Line 23** that continuously reads the next available frame from the video `stream` via the `.read()` method (**Line 29**). If the `stopped` indicator variable is ever set, we break from the infinite loop (**Lines 25 and 26**).

Again, keep in mind that once the `start` method has been called, the `update` method is placed in a *separate thread from our main Python script* — **this separate thread is how we obtain our increased**

FPS performance.

In order to access the most recently polled `frame` from the `stream`, we'll use the `read` method on **Lines 31-33**.

Finally, the `stop` method (**Lines 35-37**) simply sets the `stopped` indicator variable and signifies that the thread should be terminated.

Now that we have defined both our `FPS` and `WebcamVideoStream` classes, we can put all the pieces together inside `fps_demo.py`:

Increasing webcam FPS with Python and OpenCV	Python
<pre> 1 # import the necessary packages 2 from __future__ import print_function 3 from imutils.video import WebcamVideoStream 4 from imutils.video import FPS 5 import argparse 6 import imutils 7 import cv2 8 9 # construct the argument parse and parse the arguments 10 ap = argparse.ArgumentParser() 11 ap.add_argument("-n", "--num-frames", type=int, default=100, 12 help="# of frames to loop over for FPS test") 13 ap.add_argument("-d", "--display", type=int, default=-1, 14 help="Whether or not frames should be displayed") 15 args = vars(ap.parse_args()) </pre>	

We start off by importing our necessary packages on **Lines 2-7**. Notice how we are importing the `FPS` and `WebcamVideoStream` classes from the `imutils` library. If you do not have `imutils` installed or you need to upgrade to the latest version, please see the note at the top of this section.

Lines 10-15 handle parsing our command line arguments. We'll require two switches here: `--num-frames`, which is the number of frames to loop over to obtain our FPS estimate, and `--display`, an indicator variable used to specify if we should use the `cv2.imshow` function to display the frames to our monitor or not.

The `--display` argument is actually really important when approximating the FPS of your video processing pipeline. Just like reading frames from a video stream is a form of I/O, so is *displaying the frame to your monitor!* We'll discuss this in more detail inside the **Threading results** section of this post.

Let's move on to the next code block which does *no threading* and uses *blocking I/O* when reading frames from the camera stream. This block of code will help us obtain a *baseline* for our FPS:

Increasing webcam FPS with Python and OpenCV	Python
<pre> 17 # grab a pointer to the video stream and initialize the FPS counter 18 print("[INFO] sampling frames from webcam...") 19 stream = cv2.VideoCapture(0) 20 fps = FPS().start() 21 22 # loop over some frames 23 while fps._numFrames < args["num_frames"]: 24 # grab the frame from the stream and resize it to have a maximum 25 # width of 400 pixels 26 (grabbed, frame) = stream.read() </pre>	

```

27     frame = imutils.resize(frame, width=400)
28
29     # check to see if the frame should be displayed to our screen
30     if args["display"] > 0:
31         cv2.imshow("Frame", frame)
32         key = cv2.waitKey(1) & 0xFF
33
34     # update the FPS counter
35     fps.update()
36
37 # stop the timer and display FPS information
38 fps.stop()
39 print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
40 print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
41
42 # do a bit of cleanup
43 stream.release()
44 cv2.destroyAllWindows()

```

Lines 19 and 20 grab a pointer to our video stream and then start the FPS counter.

We then loop over the number of desired frames on **Line 23**, read the frame from camera (**Line 26**), update our FPS counter (**Line 35**), and optionally display the frame to our monitor (**Lines 30-32**).

After we have read `--num-frames` from the stream, we stop the FPS counter and display the elapsed time along with approximate FPS on **Lines 38-40**.

Now, let's look at our *threaded* code to read frames from our video stream:

Increasing webcam FPS with Python and OpenCV	Python
<pre> 46 # created a *threaded* video stream, allow the camera sensor to warmup, 47 # and start the FPS counter 48 print("[INFO] sampling THREADED frames from webcam...") 49 vs = WebcamVideoStream(src=0).start() 50 fps = FPS().start() 51 52 # loop over some frames...this time using the threaded stream 53 while fps._numFrames < args["num_frames"]: 54 # grab the frame from the threaded video stream and resize it 55 # to have a maximum width of 400 pixels 56 frame = vs.read() 57 frame = imutils.resize(frame, width=400) 58 59 # check to see if the frame should be displayed to our screen 60 if args["display"] > 0: 61 cv2.imshow("Frame", frame) 62 key = cv2.waitKey(1) & 0xFF 63 64 # update the FPS counter 65 fps.update() 66 67 # stop the timer and display FPS information 68 fps.stop() 69 print("[INFO] elapsed time: {:.2f}".format(fps.elapsed())) 70 print("[INFO] approx. FPS: {:.2f}".format(fps.fps())) 71 72 # do a bit of cleanup 73 cv2.destroyAllWindows() 74 vs.stop() </pre>	

Overall, this code looks near identical to the code block above, only this time we are leveraging the `WebcamVideoStream` class.

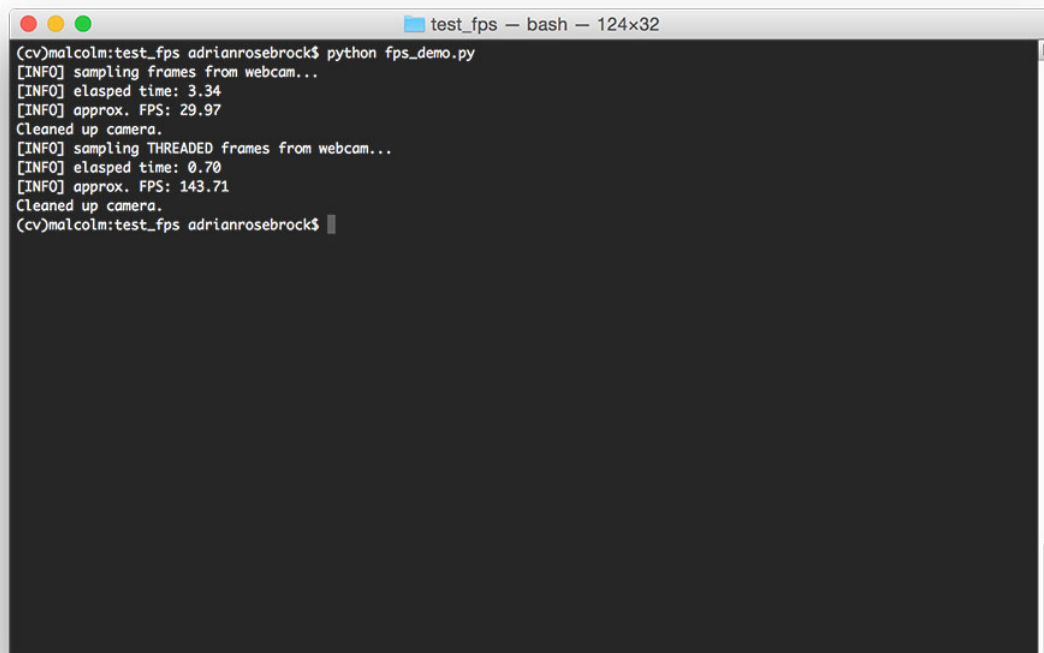
We start the threaded stream on **Line 49**, loop over the desired number of frames on **Lines 53-65** (again, keeping track of the total number of frames read), and then display our output on **Lines 69 and 70**.

Threading results

To see the affects of webcam I/O threading in action, just execute the following command:

Increasing webcam FPS with Python and OpenCV
1 \$ <code>python fps_demo.py</code>

Python



```
(cv)malcolm:test_fps adrianrosebrock$ python fps_demo.py
[INFO] sampling frames from webcam...
[INFO] elapsed time: 3.34
[INFO] approx. FPS: 29.97
Cleaned up camera.
[INFO] sampling THREADED frames from webcam...
[INFO] elapsed time: 0.70
[INFO] approx. FPS: 143.71
Cleaned up camera.
(cv)malcolm:test_fps adrianrosebrock$
```

Figure 1: By using threading with Python and OpenCV, **we are able to increase our FPS by over 379%!**

As we can see, by using no threading and sequentially reading frames from our video stream in the *main thread* of our Python script, we are able to obtain a respectable 29.97 FPS.

However, once we switch over to using *threaded camera I/O*, we reach 143.71 FPS — **an increase of over 379%!**

This is clearly a *huge decrease* in our latency and a *dramatic increase* in our FPS, obtained simply by using threading.

However, as we're about to find out, using the `cv2.imshow` can *substantially decrease* our FPS. This behavior makes sense if you think about it — the `cv2.show` function is just another form of I/O, only this time instead of reading a frame from a video stream, we're instead sending the frame to output on our display.

Note: We're also using the `cv2.waitKey(1)` function here which does add a 1ms delay to our main loop. That said, this function is necessary for keyboard interaction and to display the frame to our screen (especially once we get to the Raspberry Pi threading lessons).

To demonstrate how the `cv2.imshow` I/O can decrease FPS, just issue this command:

```
Increasing webcam FPS with Python and OpenCV
```

```
Shell
```

```
1 $ python fps_demo.py --display 1
```

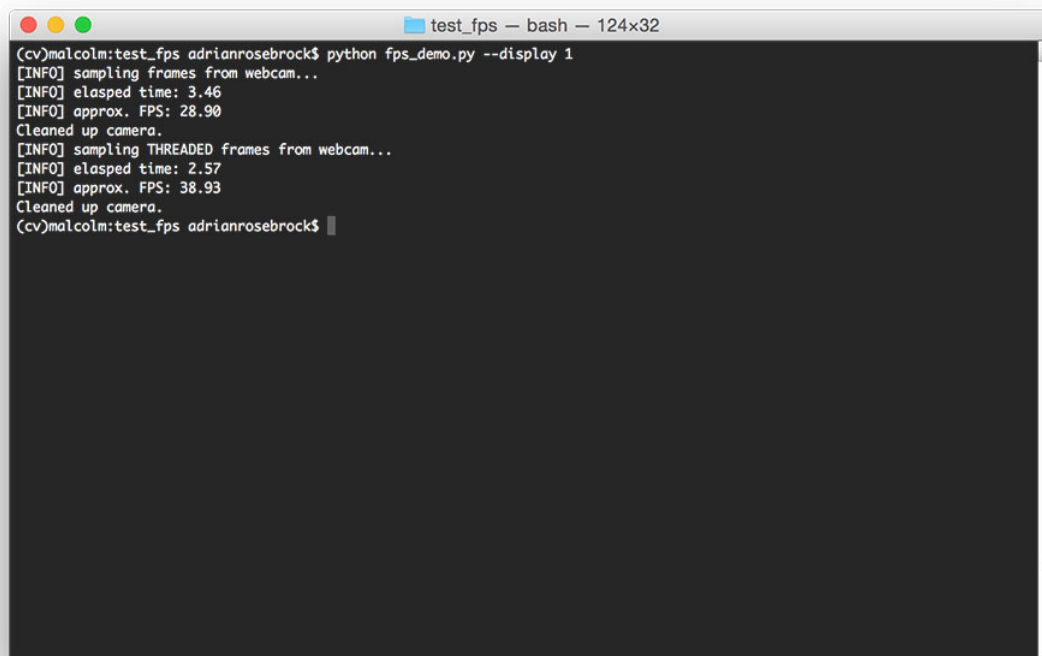


Figure 2: Using the `cv2.imshow` function can reduce our FPS — it is another form of I/O, after all!

Using *no threading*, we reach 28.90 FPS. And *with threading* we hit 39.93 FPS. This is still a **38% increase** in FPS, but nowhere near the **379% increase** from our previous example.

Overall, I recommend using the `cv2.imshow` function to help debug your program — but if your final production code doesn't need it, *there is no reason to include it* since you'll be hurting your FPS.

A great example of such a program would be developing a home surveillance motion detector that sends you a txt message containing a photo of the person who just walked in the front door of your home. Realistically, you do not need the `cv2.imshow` function for this. By removing it, you can increase the performance of your motion detector and allow it to process more frames faster.

Summary

In this blog post we learned how threading can be used to increase your webcam and USB camera FPS using Python and OpenCV.

As the examples in this post demonstrated, we were able to obtain a **379% increase in FPS** simply by using threading. While this isn't necessarily a fair comparison (since we could be processing the same frame multiple times), it does demonstrate the importance of *reducing latency* and always having a frame ready for processing.

In nearly all situations, using threaded access to your webcam can substantially improve your video processing pipeline.

Next week we'll learn how to increase the FPS of our Raspberry Pi using the [picamera](#) module.

Be sure to enter your email address in the form below to be notified when the next post goes live!

Downloads:



If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning**. Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

DOWNLOAD THE CODE!

Resource Guide (it's totally free).



Enter your email address below to get my **free 17-page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF**. Inside you'll find my hand-picked tutorials, books, courses, and Python libraries to help you master computer vision and deep learning!

DOWNLOAD THE GUIDE!

💎 **multiprocessing, threading, usb camera, video processing, video stream, webcam**

115 Responses to *Increasing webcam FPS with Python and OpenCV*



Paul Nta December 21, 2015 at 12:44 pm #

REPLY ↩

Thank you ! Great tuto !

I'm wondering if (in a production app) we should use a lock or something to synchronize access to the frame which is a shared resource, right ?



Adrian Rosebrock December 21, 2015 at 5:32 pm #

REPLY ↩

If it's a shared resource, then yes, you should absolutely use a lock on the image data, otherwise you can run into a synchronization issue.



Luke August 8, 2016 at 2:26 am #

REPLY ↩

Same here – I assumed you should have a thread acquire and release so it isn't reading the image while it is being written? Apparently assigning a value to be a numpy array is atomic – or it doesn't really matter if it was the last frame, not the very latest?

Looks like if you have ANY processing you need to have it out of that fetching image thread, and it runs pretty fast.



Jürgen December 21, 2015 at 1:00 pm #

REPLY ↩

Hi Adrian,

looks like the increase in fps is a fake; you get a frame immediately when required, but looks like it is still the former frame when the program body is executed faster than the physical frame rate of the camera. What do you think?

Jürgen



Adrian Rosebrock December 21, 2015 at 5:34 pm #

REPLY ↩

Very true — and at that point it depends on your physical camera. If your loop is faster than the physical frame rate of the camera, then the increase in FPS is not as realistic. This is further evidenced when we use the `cv2.imshow` function to simulate a more “realistic” scenario. In either case though, threading should be used since it can increase FPS.



iridos April 21, 2017 at 11:39 am #

REPLY ↩

This is 2 years later and I am using opencv 3.2. Seems to me that it is already using threading, at least all cores are in use.

I agree with Jürgen, that the higher framerate is just fake.

In the 1st (“non-threaded”) case, the code assumedly waits for the next frame of the camera to arrive, then calculates, framerate is calculated from the resulting speed of the loop. So the framerate is limited by a) camera framerate b) calculation time.

If calculation time is much smaller than camera-framerate, you are only limited by a)

In the 2nd case, you decouple the two. that means your framerate only depends on b), but you still only get another new frame according to a) and so you keep re-calculating uselessly on the same frame.

I don't see an advantage of threading the way it is done here, at least as long as your calculation time is shorter than the time between frames. (But I think that it *is* a good way to measure how long your image processing takes). You even lose some time, because you are (obviously) busy needlessly re-calculating at the time that the next frame comes in.

What happens if your calculation time is larger than the time between frames from the camera is not so clear. In your threaded case, you keep fetching images and throw away any previous ones... But I guess in the non-threaded case, the kernel does just the same? Else you'd fill kernel buffer space with frames if you cannot process fast enough (or wait long enough between fetches with `waitKey`).

You can, btw., set and query the framerate in opencv 3.2 with e.g. `cap.set(cv2.CAP_PROP_FPS, 150)` After setting, reading the value with `cap.get` only returns what you set, you have to measure fps in the read-loop, similar to what you did.

You probably also have to set smaller `cv2.CAP_PROP_FRAME_WIDTH` and `cv2.CAP_PROP_FRAME_HEIGHT` for higher fps rates or so (at least that is the case with the camera here).

Cheers,
I.



Adrian Rosebrock April 24, 2017 at 9:59 am #

REPLY ↩

Hi Iridos — thanks for the comment. I've tried to make this point clear in the blog post and in the comments section, but what you're actually measuring here is the *frame processing throughput rate of your pipeline*. There are physical limitations to the number of frames that can be read per second from a given camera sensor, so it's often helpful to instead look at how fast our process pipeline (and see if we can speed it up). By using threading, we don't have to worry about

the video I/O latency. In the *purposely simple* examples in the blog post, the **while** loop runs faster than the next frame can be read. Keep in mind that for any reasonable video processing pipeline this wouldn't be the case.



David Kadouch December 22, 2015 at 2:57 am #

REPLY ↩

Hi Adrian

This is great. I am myself experimenting with a multithreaded app that runs opencv and other libraries and I'm already using your video stream class.

Special note for OSX users: I've run into a limitation in opencv on OSX, the command `cv2.VideoCapture(0)` can only be issued in the main thread, so take this into account when designing your app. See <http://stackoverflow.com/questions/20445762/unable-to-start-camera-capture-in-python-opencv-cv2-using-a-non-main-thread> for more info



Adrian Rosebrock December 22, 2015 at 6:25 am #

REPLY ↩

Thanks for sharing David — that's a great tip regarding `cv2.VideoCapture` can only be executed from the main thread.



Hotte December 22, 2015 at 12:12 pm #

REPLY ↩

Awesome, Adrian!! Can't wait to read the tutorial about fps increase for Raspberry Pi using the picamera module!



Pär December 22, 2015 at 3:36 pm #

REPLY ↩

Great tutorial (as allways)! and good timing too...

I'm just trying to make the image gathering threaded for my raspberry pi project to improve the framerate. Without threading but with the use of a generator type of code for the image handling I improved the framerate by around 2 times but hopefully threading will do more. Another thing that is interesting is how to optimize the framerate vs the opencv computational time to reach a good balance. Jurgen mentioned that several frames could be similar and then it is no need to make calculation on that second frame (at least not in my case). On a raspberry pi 2 there is 4 cores and distributing the collection of frame data and calculations in a good way would improve the performance. Do you have any thoughts or advice about that?



Adrian Rosebrock December 23, 2015 at 6:35 am #

REPLY ↩

If you're using the Pi 2, then distributing the frame gathering to a separate thread will definitely improve performance. In fact, that's exactly what next week's blog post is going to be about 😊



Sean McLeod December 22, 2015 at 4:06 pm #

REPLY ↩

Hi Adrian

In the single threaded case you're limited to 30fps because that is the framerate of the camera in this case and you're not really achieving 143fps in the multi-threaded case since you're simply processing the same frame multiple times. The 143fps is really a measure of the amount of time the `imutils.resize()` takes, i.e. ~6.9ms. So the comparison between 30fps and 143fps isn't really a fair and accurate comparison.

I recently had a project where we ended up using the same approach, i.e. grabbing the webcam frames on a secondary thread and doing the OpenCV processing on the main python thread. However this wasn't in order to increase the fps processing rate, rather it was to minimize the latency of our image processing. We were measuring aircraft control positions by visually tracking targets on the aircraft controls to record their position during flight and needed to synchronize the recording of the control positions with another instrument we use to record the aircraft's attitude etc.

So we needed as little latency as possible in order to keep the control positions synchronized with the other aircraft data we were recording. We didn't need a high sampling rate, i.e. we were happy with 10Hz as long as the latency was as small as possible.

Our camera output at 1080@30Hz and our image processing (mainly Hough circles) took longer than the frame period of ~33ms and if we read the camera frames on the main thread the OS would buffer roughly 5 frames if we didn't read them fast enough. So going with the multithreaded approach we could always retrieve the latest frame as soon as our image processing was complete, so at a lower rate than the camera rate but with minimizing latency.

Cheers



Adrian Rosebrock December 23, 2015 at 6:34 am #

REPLY ↩

Indeed, you're quite right. The 143 FPS isn't a fair comparison. I was simply trying to drive home the point (as you suggested) of the latency. Furthermore, simply looping over a set of frames (without doing any processing besides resizing) isn't exactly fair of what a real-world video processing would look like either.



Sean McLeod December 24, 2015 at 3:36 pm #

REPLY ↩

Hi Adrian

But I think that overall you've made it more confusing mixing up fps and latency. If your main point that you were trying to drive home is the win in terms of latency then that should be in the title, in the examples you provide etc.

Sort of like mixing up a disk's transfer rate and latency.

Cheers



Adrian Rosebrock December 25, 2015 at 12:33 pm #

REPLY ↩

I'll be sure to make the point more clear in the next post on Raspberry Pi FPS and latency.



dassgo July 12, 2016 at 5:47 am #

REPLY ↩

Hi Sean,

I have the same problem as yours. I need in my project the minimum latency as possible. Due to the opencv internal buffer I have to use threads. I am working with several 8Mp cameras, each of them with its own thread. But using threads then I face the "select timeout" problem. Did you have the same problem? By the way, did you use locks to access the variable "frame"?



Ross Milligan December 23, 2015 at 4:31 am #

REPLY ↩

Nice tutorial – thanks for the mention!

I was experiencing a subtly different problem with webcam frames in my apps, which led me to use threading. I was not so concerned with the speed of reading frames, more that the frames were getting progressively stale (after running app for a minute or so on Raspberry Pi, the frame served up was a number of frames behind the real world). Perhaps my app loop was too slow interrogating the webcam and was being served up a cached image? By using a thread I was able to interrogate the webcam constantly and keep the frame fresh.



Adrian Rosebrock December 23, 2015 at 6:31 am #

REPLY ↩

Thanks for the tip Ross — you're definitely the inspiration behind where this post came from.



Sean McLeod December 24, 2015 at 3:19 pm #

REPLY ↩

Hi Ross

See my comment above, we saw the same issue as you on the ODROID we were using. On our system it looked like the OS/v4l/OpenCV stack was maintaining a buffer on the order of 5 frames if we didn't retrieve frames as fast as the camera's frame rate, which meant we ended up with an extra latency on the order of $5 \times 33\text{ms} = 165\text{ms}$.

So we ended up pulling the usb web camera images at the camera's frame rate on a secondary thread so that we were always processing the latest web camera image even though overall our main video processing only ran at 10fps.

We initially tried to see if there was a way to prevent this buffering but weren't able to find a way to disable it, so we ended up with the multi-threading approach.

Cheers



Chris Viehoff December 23, 2015 at 4:10 pm #

REPLY ↩

I installed imutils but still get this error when I run the program:

ImportError: No module named 'imutils.video'

Running python 3.4 and opencv2



Adrian Rosebrock December 23, 2015 at 6:41 pm #

REPLY ↩

Make sure you have the latest version of **imutils**:

```
$ pip install --upgrade imutils --no-cache-dir
```



Jesse June 4, 2016 at 1:35 pm #

REPLY ↩

I have the latest version installed, but I'm still getting the error. Please help if you can. All I need is something simple that can display an image on the screen from a USB webcam, and can start automatically at boot. I am running a Raspberry Pi Zero and Raspbian Jessie. The webcam is a rather cheap GE model, with YUYV 640x480 support. I have already tried multiple programs, but only luvview gave a usable picture, and it broke itself when attempting at auto-start script. Any help at all would be useful! Thank you in advance!



Adrian Rosebrock June 5, 2016 at 11:30 am #

REPLY ↩

I detail how to create an [autostart script here](#). This should also take care of the error messages you're getting since you'll be accessing the correct Python virtual environment where ideally you have installed the imutils package.



Surya February 5, 2016 at 5:17 am #

REPLY ↩

Thank you for a great tutorial. I am working with applications like SURF, Marker detection etc. but i need to increase the FPS for the above mentioned applications. Will this approach work with OpenCV C++? If yes, how?



Adrian Rosebrock February 5, 2016 at 9:18 am #

REPLY ↩

Yes, the same approach will work for C++. The exact code will vary, but you should do some research on creating and managing separate threads in C++.



Nimohunter June 8, 2017 at 4:48 am #

REPLY ↩

I completed a C ++ version, but don't know it is suitable with C++ mutex. Hope this is useful.

[mutexvideocapture.cpp](https://github.com/nimohunter/Calibration-Use-ChArucoBoard/blob/master/mutexvideocapture.cpp)



David February 6, 2016 at 2:34 am #

REPLY ↩

This is a classic producer/consumer problem.

Here we have a camera (the producer) that is delivering frames at a constant rate in real time and a frame reading program (the consumer) that is not processing the frames in real time.

In this case we must have a frame queue with a frame count. If the frame count is > 0 then the consumer consumes a frame – reducing the frame count. If the frame count is zero then the consumer must wait until the frame count rises above zero before consuming a frame.

There can be any number of consumers but the producer must serialize access to the frame queue (using locks/semaphores).

I'm wondering if Adrian has fully covered this aspect in his book and tutes...



Adrian Rosebrock February 6, 2016 at 9:55 am #

REPLY ↩

I don't cover this in detail inside Practical Python and OpenCV, but it's something that I do plan on covering in future blog posts.

Unless you want to process *every frame* that is read by the camera, I actually wouldn't use a producer-consumer relationship for this. Otherwise, as you suggested, if your buffer is > 0 , then you'll quickly build up a backlog of frames that need to be processed, making the approach unsuitable for real-time applications.



Bram June 6, 2016 at 11:09 am #

REPLY ↩

I don't see a performance increase on Windows 7. Process explorer (confirmed by cpu usage graph in Task Manager) confirms cv2 is already divided in multiple threads??



Adrian Rosebrock June 7, 2016 at 3:22 pm #

REPLY ↩

I'm not a Windows user, so I'm not entirely sure how Windows handles threading. The Python script won't be divided across multiple process, but a new thread should be spawned. Again, I haven't touched a Windows system in over 9+ years, so I'm probably not the right person to ask regarding this question.



Stijn July 3, 2016 at 5:27 am #

REPLY ↩

Hello,

Thank you very much for sharing this information.

One drawback of this method is as mentioned in the comments that you kind of lose the timestamp / counter information when a frame was shot by the camera.

Now the funny thing is a timestamp is provided if you use the `capture_continuous` function and provide a string as argument.

So diving a bit deeper in the code of this function, don't you think we could just add a timestamp / counter in the "else" condition? It might make the code that later processes these images a bit more efficient since a mechanism can be made to avoid processing the same frame twice.

Not sure if you have an opinion on this one / ever experimented with it 😊 ?

http://picamera.readthedocs.io/en/release-1.10/_modules/picamera/camera.html#PiCamera.capture_continuous



Aaron July 11, 2016 at 9:10 am #

REPLY ↩

hey is there a use for the variable "grabbed"? I can't see it being used anywhere... I might be misunderstanding a lot though!



Adrian Rosebrock July 11, 2016 at 10:12 am #

REPLY ↩

The `stream.read` function returns a 2-tuple consisting of the frame itself along with a boolean indicating if the frame was successfully read or not. You can use the `grabbed` boolean to determine if the frame was successfully read or not.



Kevin July 15, 2016 at 7:13 pm #

REPLY ↩

Hello Adrian,

Thank you for the very useful blog post. You explain everything very clearly, especially to someone very new to python and image processing. Have you ever made a blog post regarding packages/module hierarchy? In some of your other blog posts you explicitly tell us how to name a file, but I'm confused about how the FPS and WebcamVideoStream classes are defined in the directory. More specifically, what names should those files have (is there a naming convention?) Where in the project are they typically located? How are they pulled "from imutils.video"? I know these are very basic questions, but I haven't found a resource online that explains this clearly.

Thanks again for your work.



Adrian Rosebrock July 18, 2016 at 5:22 pm #

REPLY ↩

Hey Kevin — this is less of a computer vision question, but more of a Python question. I would suggest investing some time reading about Python project structure and taking a few online Python courses. I personally like the [RealPython.com](#) course. The course offered by [CodeAcademy](#) is also good.



yair levi July 24, 2016 at 12:00 pm #

REPLY ↩

hi!

Thanks for the great tutorial.

Is there any way to make the `.read()` method block until there is a new frame to return?

If not, is there any efficient way to determine if the old frame is the same as the new frame so I can ignore it?

thanks again



Adrian Rosebrock July 27, 2016 at 2:38 pm #

REPLY ↩

As far as I know, you can't block the `.read()` method until a new frame is read. However, a simple, efficient method to determine if a new frame is ready is to simply hash the frame and compare hashes. [Here is an example](#) of hashing NumPy arrays.



Peni Jitoko August 2, 2016 at 11:48 pm #

REPLY ↩

Hi,

I followed up on the tutorial, after executing "python picamera_fps_demo.py" the results were:

[INFO] sampling frames from 'picamera' module...

[INFO] elapsed time: 3.57

[INFO] approx. FPS: 28.32

[INFO] sampling THREADED frames from 'picamera' module...

[INFO] elapsed time: 0.48

[INFO] approx. FPS: 208.95

but when I ran "python picamera_fps_demo.py --display 1" the results were:

[INFO] sampling frames from 'picamera' module...

[INFO] elapsed time: 8.54

[INFO] approx.FPS: 11.83

[INFO] sampling THREADED frames from 'picamera' module...

[INFO] elapsed time: 6.54

[INFO] approx. FPS: 15.29

So it ran slower, I'm not sure what is the real issue, but I'm using a Pi 3 fresh out of the box, using a 5V 1A power source, I'm connected the Pi 3 to the laptop using Xming and putty via LAN cable.



Adrian Rosebrock August 4, 2016 at 10:18 am #

REPLY ↩

Looking at your results, it seems that in both cases the *threaded* version obtained faster FPS processing rate. 15.29 FPS is faster than 11.83 FPS and 208.95 FPS is faster than 28.32 FPS. So I'm not sure what you mean by running slower?



Peni Jitoko August 4, 2016 at 7:32 pm #

REPLY ↩

Thanks for the reply Adrian, when I see the video feed from the Pi camera on my laptop, there is a large delay, when I wave my hand over the camera it almost takes 2 to 3 seconds then I see my hand in the video feed, is this delay normal or is it an issue.



Adrian Rosebrock August 7, 2016 at 8:23 am #

REPLY ↩

So you're accessing the video feed via X11 forwarding or VNC? That's the problem then. Your Pi is reading the frames just fine, it's just the network overhead of *sending* the frames from the Pi to your VNC or X11 viewer. If you were to execute the script on your Pi with a keyboard + HDMI monitor, the frames would look much more fluid.



Peni Jitoko August 8, 2016 at 5:57 pm #

Hi Adrian,

I'm accessing the video feed via X11 forwarding, thanks for helping me identify what the problem really is. A lot of your tutorials has provided me with the basic foundation for my project,

there is no other place I would recommend a beginner like me to start off learning image and video processing.



Adrian Rosebrock August 8, 2016 at 6:35 pm #

Great job resolving the issue Peni, I'm glad it was a simple fix. And thank you for the kind words, I'm happy I can help out 😊



Samuel August 9, 2016 at 3:37 am #

REPLY ↩

Hi Adrian, thank you for the post!

I have a question that is currently bothering me a bit, that is: when we use multi threads as this post's approach, does that mean we are using multi cores? or are we just using single core with 2 threads? cuz I'm currently up to some real-time project and trying to do it using multi threads, what surprised me is that the number of frames i can process per second actually decreased a bit, cuz i thought if i'm using a different core for reading in the video i should at least save the reading time and be able to process a bit more frames per second right? Thanks again for your help!



Adrian Rosebrock August 10, 2016 at 9:31 am #

REPLY ↩

Hey Samuel — we are using a single core with 2 threads. Typically, when you work with I/O operations it's advisable to utilize threads since most of your time is spent waiting for new frames to be read/written. If you're looking to parallelize a computation, spread it across multiple cores and processors.



Shirosh August 14, 2016 at 2:35 pm #

REPLY ↩

Hi.! How to use this python code with other image processing task. should we run parallel these two codes using terminal? how to do this? please help me.
Anyway your blog is the best



Adrian Rosebrock August 16, 2016 at 1:09 pm #

REPLY ↩

Hey Shirosh — I'm not sure what you mean by "use this Python code with other image processing task". Can you please elaborate?

Cassiano Rabelo September 8, 2016 at 10:44 pm #

REPLY ↩



Hello Adrian,

I've noticed that `cv2.imshow` on OS X is muuuuch slower than its counterpart on windows.

The following benchmark runs in 15 seconds on a virtualised windows inside my mac, but it takes as long as 2 minutes to run on OS X itself!

<https://db.tt/9FklKUpJ>

Do you know what could be the reason and possible fix? Thanks a lot!

Best,

Cassiano



Adrian Rosebrock September 9, 2016 at 10:54 am #

REPLY ↩

That's quite strange, I'm not sure why that may be. I don't use Windows, but I've never noticed an issue with `cv2.imshow` between various operating systems.



Nam Taehun November 6, 2016 at 12:54 am #

REPLY ↩

Hi, Adrian.

I have a question.

Where can I save 'fps_demo.py' ?



Adrian Rosebrock November 7, 2016 at 2:50 pm #

REPLY ↩

You can save it anywhere on your system that you would like. I would suggest using the "Downloads" section of this tutorial to download the code instead though.



mazyar November 28, 2016 at 4:04 am #

REPLY ↩

hi adrian

thank you for this good project

when i run this project

after increase fps show a core dump segmentation fault error . can you help me ??..



Adrian Rosebrock November 28, 2016 at 10:17 am #

REPLY ↩

If you're getting a segmentation fault then the threading is likely causing an issue. My best guess is that the stream object is being destroyed while it's in the process of reading the next frame. What OS are you using?



Yakup Emre December 12, 2016 at 3:40 am #

REPLY ↩

Hi,

I am trying to use Ps3 EYE Camera on ubuntu for my OpenCv project. This camera support 640×480 up to 60fps and 320×240 resolution up to 187fps. I am sure you know what I mention about. I can set each one of this values on windows with CodeLaboratories driver. But on ubuntu, I use ov534.c driver and QT v4L2 software. Even though I 'm seeing all of configuration settings of this camera on v4L2, I can't set over 60fps. I can set any value under 60fps. Do you have an idea about this problem. What can I do for setting at least 120fps?



Adrian Rosebrock December 12, 2016 at 10:26 am #

REPLY ↩

Unfortunately, I don't have any experience with the PS3 Eye Camera, so I'm not sure what the proper parameter settings/values are for it. I hope another PyImageSearch reader can help you out!



Jon R February 4, 2017 at 11:41 pm #

REPLY ↩

Hey Adrian,

I'm working on an image processing pipeline with live display that runs at 75FPS without `cv2.imshow()` and 13 FPS with no screen output (OS-X FPS #s). I need a live output to the screen, and I want to maximize framerate. I already tried using Pygame's blit command as an `imshow()` replacement, but got about the same speed. Are you aware of a module/command that will get an efficient screen refresh? If I'm lucky there will be an approach that will transfer from OS X to Raspberry Pi without too many hitches.

Thanks.



Adrian Rosebrock February 7, 2017 at 9:23 am #

REPLY ↩

Hey Jon — when it comes to displaying frames to screen I've always used `cv2.imshow`. Unfortunately I don't know of any other methods to speedup the process. Best of luck with the project!



Jon R February 12, 2017 at 9:38 pm #

REPLY ↩

I found that PySDL2 as an interface to the SDL2 library fixes the problem on OS-X and may be viable on Raspberry Pi too. For any interested readers, this post has code that can be integrated into the code here to speed things up when displaying all frames to the screen.

<http://stackoverflow.com/questions/18434348/convert-cv2-images-to-pysdl2-surfaces-for-blitting-to-screen#19554202>

Thanks for a great resource Adrian!



Adrian Rosebrock February 13, 2017 at 1:37 pm #

REPLY ↩

Thanks for sharing Jon!



Hanifudin March 22, 2017 at 4:35 am #

REPLY ↩

Hello adrian.. i recently mixing this code with image filtering.. but the image had a quite delay,, the image is like quite freeze.. not like cv2.videocapture(-1). What happen..? I dont know this..Can you explain me..? and if i use only serial usart communication i/o to send x y coordinate out.. where the code part must be change.. ? Thanks



Adrian Rosebrock March 22, 2017 at 8:34 am #

REPLY ↩

Hey Hanifudin — I'm honestly not sure what you are trying to ask. Can you please try to elaborate on your question? I don't understand.



Hanifudin March 22, 2017 at 10:47 pm #

REPLY ↩

1. I send coordinate x and y target with serial pin. But when i using threading,, raspberry pi cannot send data via serial..so, how i activate serial tx rx in threading mode..?
2. In threading mode,, the image is quite freeze,, like paused and lagging (not smooth)..it doesnt like non threading mode.. so how i make it smooth like non threading mode..?

Im sorry i've bad english



Adrian Rosebrock March 23, 2017 at 9:25 am #

REPLY ↩

Unfortunately, I don't have much experience sending serial data via threading in this particular use case. I would try to debug if this is a computer vision problem by removing any computer vision code and ensure your data is being properly sent. As for the slow/freezing video stream, I'm not sure what the exact problem is. You might be overloading your hardware or there might be a bug in your code. It's impossible for me to know without seeing your machine.

Hanifudin March 23, 2017 at 7:22 pm #



Oh,, i know how why captured image is quite freeze,, im using usb hub to connect webcam,, wifi adapter and wireless mouse.. when i connect without usb hub it solved..

And sending serial data in threading mode is waiting to solved



Chris June 22, 2017 at 2:44 pm #

REPLY ↩

Hi Adrian,

Thanks for the codes!

I tried to run the code. it works perfectly without display but I get an error "Segmentation fault: 11" when sampling threaded frames with display on. what should I do?

And I'm also doing a project with python and opencv try to record 120fps video, my camera is good for the fps but the utmost I can get is 30fps, any recommendations?

Chris



Adrian Rosebrock June 27, 2017 at 6:46 am #

REPLY ↩

Regarding the segmentation fault, that sounds like a threading error. Most likely either the OpenCV window is being closed before the thread is finished or vice versa. It's hard to debug that error, especially without access to your machine.

Also, keep in mind that you cannot increase the physical FPS of your camera. The code used in this blog post shows how you can increase the frame processing rate of your *pipeline*. If your camera is limited to 30 FPS, you won't be able to go faster than that. However, you will use this technique when you add more complex steps to your video processing pipeline and need to milk every last bit of performance out of it.



Guille June 24, 2017 at 8:15 pm #

REPLY ↩

Hi Adrian, your post is impressive! I have similar problem right now but I'm working on 16 bits sensors/cameras instead. Can I apply this to 16 bits? Do you know how to use opencv to capture 16 bits frames from usb device? Thanks!



Adrian Rosebrock June 27, 2017 at 6:33 am #

REPLY ↩

Hi Guille — I haven't tried this code using a 16 bit raw sensor to OpenCV, so unfortunately I don't have any insight here.

**Ram** July 26, 2017 at 8:05 pm #

REPLY ↩

Hi,

Its a great tutorial. I am new to python and opencv and your explanation is great . Thanks for posting and explaining everything line by line. I understood the program, but if i want to store the frame using videowriter instead of viewing them, where do i do it. I am thinking wither in the webcamcapture class so that it is done in the thread itself or should i do it in the main thread where you use cv2.show.

Currently i am using a webcam to capture at 1280x720 @ 30 hz, but i only get 8Hz. I am trying to improve this and want 30 Hz. Is this possible.

**Adrian Rosebrock** July 28, 2017 at 9:54 am #

REPLY ↩

If you would like to save the frames to disk, please see [this blog post](#) where I discuss `cv2.VideoWriter`.

**Shaun** October 6, 2017 at 11:11 am #

REPLY ↩

Hi , Thank you for this awesome tutorials !

I'm using RPi3 with Raspbian Stretch.

When I run fps_demo.py, I get the error message as shown:

```
[INFO] sampling frames from webcam...
```

```
Traceback (most recent call last):
```

```
...
```

```
(h, w) = image.shape[:2]
```

```
AttributeError: 'NoneType' object has no attribute 'shape'
```

I saw your OpenCV: Resolving Nonetype errors post but I'm still confused.

Could you help me solve this error?

**Shaun** October 6, 2017 at 11:39 am #

REPLY ↩

Oh, figured out that it was for the webcam, not the picamera. I saw your increasing picamera fps and it was all good.

Is it possible to use this method if I unify picamera and cv2.VideoCapture as you posted in here ?

<https://www.pyimagesearch.com/2016/01/04/unifying-picamera-and-cv2-videocapture-into-a-single-class-with-opencv/>

Adrian Rosebrock October 6, 2017 at 4:46 pm #

REPLY ↩



I would suggest simply using the **VideoClass** discussed in the link you included. This is the best method to achieve computability with both the Raspberry Pi camera module and USB cameras.



Nipuna Cooray October 11, 2017 at 1:43 am #

REPLY ↩

Hi Adrien,

Thank you for the wonderful tutorial. I've developed an app to get 4 camera streams and show and save one video with all 4 streams. I've also used your threaded streams approach for better performance. The problem is, I cannot figure out correct FPS to use for the output video (in the VideoWriter). Any thoughts on this ?



Adrian Rosebrock October 13, 2017 at 8:59 am #

REPLY ↩

Please see [this blog post](#), in particular the comments section. The gist is that you'll likely need to tune the FPS experimentally.



Kevin Su October 14, 2017 at 1:16 am #

REPLY ↩

instead `cv2.VideoCapture(0)`

I replaced it with `WebcamVideoStream(src=0)`

then `@ ret, frame = cap.read()`

ERROR :too many value to unpack

why it happen and how to fix it ?



Adrian Rosebrock October 14, 2017 at 10:33 am #

REPLY ↩

Please refer to this blog post. Notice how the `.read` method of `WebcamVideoStream` only returns a single value:

```
frame = cap.read
```



Kevin Su October 16, 2017 at 11:43 am #

REPLY ↩

by use this code, my program running noticeable faster , but~ still have some delay, what others I can do ? or do u have any tutorial about use GPU to make it faster ?



Adrian Rosebrock October 16, 2017 at 12:13 pm #

REPLY ↩

I'm not sure what you're asking. What in particular are you trying to optimize via the GPU?



Kevin Su October 19, 2017 at 5:58 am #

REPLY ↩

basically, you code made my program running much faster , thanks for that , but it still not fast enough, I wonder if any other ways can make it even better ?



Adrian Rosebrock October 19, 2017 at 8:39 am #

REPLY ↩

What type of hardware are you using? Laptop? Desktop? Raspberry Pi? I would suggest trying to optimize the OpenCV library itself, as we do in [this post](#).



Jim G November 19, 2017 at 11:54 pm #

REPLY ↩

Hi Adrian, Awesome tutorials. I have learned a ton. I wish all online tutorials were as thoughtful and well-designed. Thank you.

I was trying to time the latency of a few of my web cams. Basically, I took your program and revised it a bit. It starts off by reading about 100 frames to warm the camera up. After 100 frames, it starts a timer and then uses serial via an Arduino to light a blue LED in front of the lens. The program then counts the number of frames read after the LED is lit and the time that elapses. Using 640×480 and 60fps, without threading, it takes about .09 secs between turning on the LED and identifying the LED. (It does this in one frame, which I gather means there's no buffer.) With threading, it takes about .06 seconds and reads about five frames. With 720p images, there's almost no difference in latency bw threaded and unthreaded approaches. So, have three questions please:

- 1) If the frames come in faster than the program can process them, will a queue build up and will that queue get longer and longer as the program runs? Is there any way to clear it out?
- 2) I am using this for robotics, so I would actually want to read the newest frame and dump the older frames, even though they haven't been read. (I actually want last in first out rather than first in first out.)
- 3) Why would threading not improve the latency of capturing video at higher resolutions?

Sorry for the long question, but I thought more detail was better.



Adrian Rosebrock November 20, 2017 at 3:55 pm #

REPLY ↩

Thanks Jim, I appreciate that.

1. That really depends on the data structure you use. With a traditional list, yes. Other data structures place limits on the number of items to store in the queue. It's really implementation dependent.
2. If that's the case, use a FIFO queue. The [Python language](#) has them built-in by default.
3. It's an issue of more data to transfer. The less data we transfer, the faster the pipeline. The more data we transfer, the slower the pipeline. Then there are overheads on packing/unpacking the data which we can't really get around.



Roman January 7, 2018 at 3:30 pm #

REPLY ↩

Hi Adrian

Thank you for the tutorial! It helps a lot (i am actually a greenhorn in programming :-))

I have a quick question: I have a C920 USB Webcam attached to a raspberry 3b. Before i start the script, i can set the cameras resolution to 1920x1080 (v1l2-ctl -set-fmt-video=width=1920,height=1080,pixelformat=1). I check the settings with: v1l2-ctl -d /dev/video0 -V. Then i can see that the resolution was changed to 1920x1080 and the codec to H264.

When i start your script, the resolution gets changed to 640x480 and the codec is changed to YUYV. I cannot set the resolution while your script is running ("Device is busy"). So i have to set it in your script. Do you have a clue where to set the resolution/codec?

Best regards,
Roman



Adrian Rosebrock January 8, 2018 at 2:41 pm #

REPLY ↩

Hey Roman — you would need to download a copy of the **WebcamVideoStream** class and then modify the **self.stream** object in the constructor after Line 9. The capture properties can be set via the **.set** function but they are a bit buggy depending on the camera and install. You can read up more on them [here](#).



Sang August 11, 2018 at 9:27 pm #

REPLY ↩

Hi Adrian,

I use Logitech 920, and I put some more lines after `self.stream = cv2.VideoCapture(src)`

```
def __init__(self, sec=0):
    self.stream = cv2.VideoCapture(src)
    self.stream.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    self.stream.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
    self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*"MJPG"))
```

but the resolution remains the same.

On the other hand, it works without threading

```
webcam = cv2.VideoCapture(0)
webcam.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
webcam.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
```

Is that a buggy thing you mentioned?



Adrian Rosebrock August 15, 2018 at 9:05 am #

REPLY ↩

IF you're modifying the actual `imutils` library itself then it *should* use the dimensions you supplied. Make sure you're not accidentally editing a different `imutils` install on your system and make sure you're importing the correct ones (a few "print" statements will help you debug that).



Javier January 25, 2018 at 11:04 am #

REPLY ↩

I was checking `imutils` code and the thing is, that using it doesn't give you a bigger framerate. When you call `read()` it will return the last read frame. You can call it 5 times, and get the same frame back over and over. So this does not improve the framerate, instead it gives you repeated frames creating the illusion of a bigger framerate



Adrian Rosebrock January 25, 2018 at 3:43 pm #

REPLY ↩

Hey Javier — you are correct that `.read()` will return whatever the newest frame is in the buffer. If your processing pipeline is so incredibly fast that you can process a frame before the next one is ready you could process duplicate frames. For most processing pipelines that will not be the case. Perhaps I didn't make it clear enough in this blog post (but I tried to in the [follow up one](#)) that we are increasing the frame rate of the processing pipeline, not necessarily the frame rate of the actual camera sensor.



SomSay January 28, 2018 at 3:03 pm #

REPLY ↩

Hello

Sir

I can't detect face using `rpi`.

Help me.



Adrian Rosebrock January 30, 2018 at 10:24 am #

REPLY ↩

Which code are you using to detect faces? For what it's worth, I demonstrate how to perform face detection on the Raspberry Pi inside [Practical Python and OpenCV](#).



Menaka February 20, 2018 at 11:04 am #

REPLY ↩

Hi Adrian,

I love your blog and all the posts in it.

I am working with Python 3.6 on a Windows 8.1 machine.

The code runs perfectly except that it doesn't exit from command-line on completion.

Is it because some threads are still active in the background?

Please help me resolve this tiny problem.

Thank you for your help!

Keep posting 😊



Adrian Rosebrock February 22, 2018 at 9:15 am #

REPLY ↩

Hey "Menaka" — can you elaborate on what you mean by "doesn't exit from command-line completion"? I'm not sure what you are referring to.



Nethra February 21, 2018 at 1:58 pm #

REPLY ↩

how to reduces frame per second in video file



Adrian Rosebrock February 22, 2018 at 8:58 am #

REPLY ↩

Just to clarify, are you referring to writing a frames to an output file to disk?



Nethra February 23, 2018 at 3:52 am #

REPLY ↩

Thanks for reply

Actually we are doing object detection in video using tensorflow. As we are using cpu the video file hardly runs. so we just tried to reduce the fps read by opencv



Adrian Rosebrock February 26, 2018 at 2:11 pm #

REPLY ↩

So the TensorFlow model is running too quickly on each of the input frames? In that case, just add a `time.sleep` call to the end of your loop that is responsible for grabbing new frames.



Phil February 26, 2018 at 11:53 pm #

REPLY ↩

This is great tutorial, but I have a couple questions: 1) How can I make use of `cv2.waitKey()` on a headless system (ie: with no windowing system, accessed over SSH)? and 2) None of this works with the PiCamera module. What would be the equivalent code for that?



Phil February 26, 2018 at 11:54 pm #

REPLY ↩

Oh wait, nevermind! I see there's a separate tutorial for the PiCamera module (<https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/>). Thanks!



Adrian Rosebrock February 27, 2018 at 11:30 am #

REPLY ↩

Yep, no problem Phil. Let me know if you have any other questions.



Martin F April 16, 2018 at 4:18 am #

REPLY ↩

Hi everybody !

I used this method to read a video not from my cam but from another stream. The problem is that if you read directly from the file, `opencv` will try to read it as fast as possible. If you want to stick with the right frame rate, add a `sleep(1/fps)` in the thread update method 😊
Thank you Adrian for this great tuto again !



sanup s babu April 19, 2018 at 12:47 pm #

REPLY ↩

I am doing a project where my robot follows a person using a color tracking method, i am using the camera https://www.amazon.in/Logitech-C270-HD-Webcam-Black/dp/B008QS9J6Y/ref=sr_1_2?ie=UTF8&qid=1506877018&sr=8-2&keywords=usb+webcams , using this camera I am getting 5-second delay to show the streaming using `cv2.VideoCapture(0)`
Can u tell me suggestion to reduce the delay?

sanup s babu April 19, 2018 at 1:17 pm #

REPLY ↩



i am using raspberry pi 3 and python code.
can u tell me to get a low latency video using cv2.videocapture()



Adrian Rosebrock April 20, 2018 at 10:02 am #

REPLY ↩

A five second delay sounds incredibly high. Can you tell us more about the stream? You mentioned using a robot, but is the robot streaming the video stream to another system where the processing takes place?



ALE April 23, 2018 at 8:53 am #

REPLY ↩

Hi Adrian,

I love your blog, it is really usefull.

I have a technical question:

I am doing an pupil diameter project and with every action I do to measure the pupil (aka: morphological transformation, median filter, among others) I lose FPS, so I start reading the video at 30 FPS and at the end it is only 4 FPS.

This is do to all the action I do in the frame video? If I use a camera with a higher default FPS, will I get at the end more FPS?



Adrian Rosebrock April 25, 2018 at 5:58 am #

REPLY ↩

The problem isn't your camera, it's your processing pipeline. Think of your pipeline as a staircase. The less steps there are (i.e., computer vision or programming operations), the less stairs on the staircase and the faster you can clime it. The more operations, the more stairs, and the longer it takes. You're in the same boat here — you need to optimize your actual processing pipeline.



Mike Glover May 8, 2018 at 9:37 am #

REPLY ↩

Adrian, wow. I can't thank you enough for all of your tutorials on OpenCV with dnn + Python. My question to you is, I am looking to consume an rtsp stream from IP Cameras on my network. I've seen some basic examples of how to capture frames in to memory but not sure I am confident in integrating the two (Your real time object detection and the rtsp frame capture). Do you have any advice or know of a tutorial that showcases this? I guess it doesn't necisarily have to be rtsp, even http (if that's all the camera supports). My initial thought was to have a different script (or thread) capturing frames and feeding them through the script we wrote in your tutorial for "Object detection with deep learning and OpenCV". Would love to know your thoughts.

**Adrian Rosebrock** May 9, 2018 at 9:40 am #

REPLY ↩

Hey Mike, thanks for the comment! I'm so happy to hear you are enjoying the PyImageSearch blog



I do not have any tutorials on RTSP but I've added it to my queue for later this year. I'll try to do a post on it sooner rather than later. As far as combining the two, is your goal to apply object detection to the frame after you grab it from an RTSP stream? Or are you trying to pull an image from your webcam, apply object detection, and then write it back out to the RTSP stream?

**Rodrigo** June 7, 2018 at 7:21 pm #

REPLY ↩

Thanks Adrian

I using the code "Real-time object detection with deep learning and OpenCV" for object detect from IPCAM hikvision throught RTSP stream.

Its great solution, but after a few minutes I have problems with the get frame: the error is:

```
-----  
frame = imutils.resize(frame, width=400)  
File "/usr/local/lib/python2.7/dist-packages/imutils/convenience.py", line 45, in resize  
(h, w) = image.shape[:2]  
AttributeError: 'NoneType' object has no attribute 'shape'  
-----
```

I think the problem is the processing pipeline and frame delay. I will hope your blog about RTSP stream.

Cheers from Loja-Ecuador

Rodrigo

**Adrian Rosebrock** June 8, 2018 at 6:53 am #

REPLY ↩

I think it may be an issue related to the RTSP stream — perhaps the stream is dropping. That said you can handle the error gracefully by checking if the frame is "None" before you continue processing it:

	Python
1	<code>if frame is None:</code>
2	<code>continue</code>

That way if a frame drops during the stream you can still handle the issue and ideally the next frame will not be dropped.

Trackbacks/Pingbacks

Increasing Raspberry Pi FPS with Python and OpenCV - PyImageSearch - December 28, 2015

[...] Last week we discussed how to: [...]

[Unifying picamera and cv2.VideoCapture into a single class with OpenCV - PyImageSearch](#) - January 4, 2016

[...] blog, we have discussed how to use threading to increase our FPS processing rate on both built-in/USB webcams, along with the Raspberry Pi camera [...]

[Real-time panorama and image stitching with OpenCV - PyImageSearch](#) - January 25, 2016

[...] the past month and a half, we've learned how to increase the FPS processing rate of builtin/USB webcams and the Raspberry Pi camera module. We also learned how to unify access to both USB webcams and [...]

[Faster video file FPS with cv2.VideoCapture and OpenCV - PyImageSearch](#) - February 6, 2017

[...] I've mentioned in previous posts, the `.read` method is a blocking operation — the main thread of your Python + OpenCV [...]

Quick Note on Comments

Please note that all comments on the PyImageSearch blog are hand-moderated by me. By moderating each comment on the blog I can ensure (1) I interact with and respond to as many readers as possible and (2) the PyImageSearch blog is kept free of spam.

Typically, I only moderate comments every 48-72 hours; however, I just got married and am currently on my honeymoon with my wife until early October. Please feel free to submit comments of course! Just keep in mind that I will be unavailable to respond until then. For faster interaction and response times, you should join the [PyImageSearch Gurus course](#) which includes private community forums.

I appreciate your patience and thank you being a PyImageSearch reader! I will see you when I get back.

Leave a Reply

Name (required)

Email (will not be published) (required)

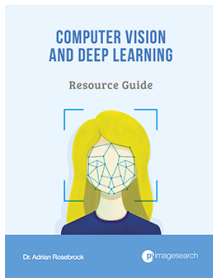
Website

SUBMIT COMMENT

Search...



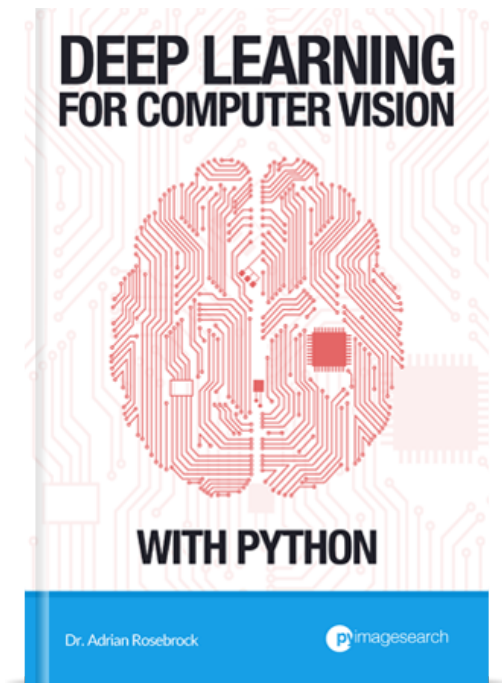
Resource Guide (it's totally free).



Get your **FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF**. Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

[Download for Free!](#)

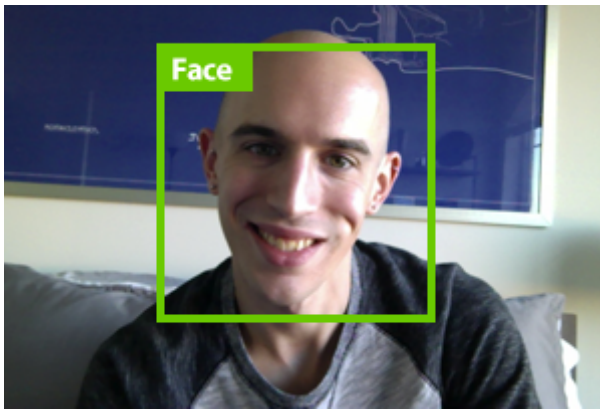
Deep Learning for Computer Vision with Python Book — OUT NOW!



You're interested in deep learning and computer vision, *but you don't know how to get started*. Let me help. **My new book will teach you all you need to know about deep learning.**

[CLICK HERE TO MASTER DEEP LEARNING](#)

You can detect faces in images & video.



Are you interested in **detecting faces in images & video**? But **tired of Googling for tutorials** that *never work*? Then let me help! I guarantee that my new book will turn you into a **face detection ninja** by the end of this weekend. [Click here](#) to give it a shot yourself.

[CLICK HERE TO MASTER FACE DETECTION](#)

PyImageSearch Gurus: NOW ENROLLING!

The PyImageSearch Gurus course is *now enrolling*! Inside the course you'll learn how to perform:

- Automatic License Plate Recognition (ANPR)
- Deep Learning
- Face Recognition
- *and much more!*

Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

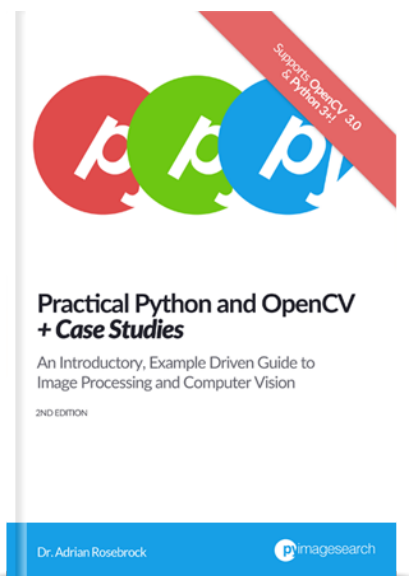
[TAKE A TOUR & GET 10 \(FREE\) LESSONS](#)

Hello! I'm Adrian Rosebrock.



I'm an entrepreneur and Ph.D who has launched two successful image search engines, [ID My Pill](#) and [Chic Engine](#). I'm here to share my tips, tricks, and hacks I've learned along the way.

Learn computer vision in a single weekend.



Want to learn computer vision & OpenCV? I can teach you in a **single weekend**. I know. It sounds crazy, but it's no joke. My new book is your **guaranteed, quick-start guide** to becoming an OpenCV Ninja. So why not give it a try? [Click here](#) to become a computer vision ninja.

[CLICK HERE TO BECOME AN OPENCV NINJA](#)

Subscribe via RSS



Never miss a post! Subscribe to the PyImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

APRIL 18, 2016

Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi

SEPTEMBER 4, 2017

Install OpenCV and Python on your Raspberry Pi 2 and B+

FEBRUARY 23, 2015

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox

JUNE 1, 2015

Ubuntu 16.04: How to install OpenCV

OCTOBER 24, 2016

How to install OpenCV 3 on Raspbian Jessie

OCTOBER 26, 2015

Basic motion detection and tracking with Python and OpenCV

MAY 25, 2015

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.

© 2018 PyImageSearch. All Rights Reserved.