



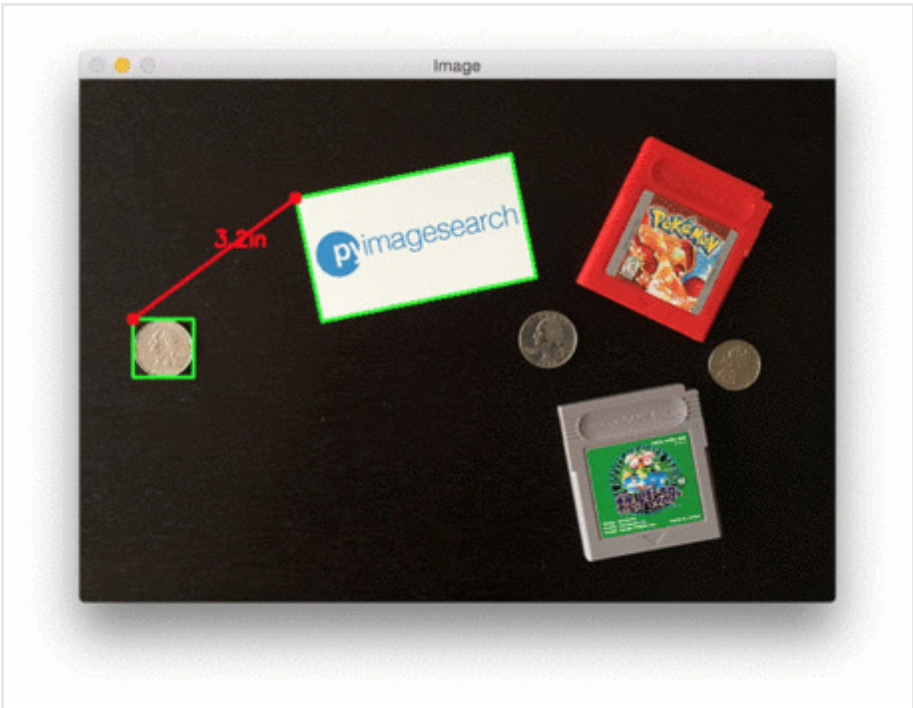
Measuring distance between objects in an image with OpenCV

by **Adrian Rosebrock** on April 4, 2016 in **Image Processing, Tutorials**

Like 12

G+1

19



We have now reached the final installment in our three part series on *measuring the size of objects in an image* and *computing the distance between objects*.

Two weeks ago, we started this round of tutorials by learning [how to \(correctly\) order coordinates in a clockwise manner using Python and OpenCV](#). Then, last week, we discussed [how to measure the size of objects in an image](#) using a reference object.

This reference object should have two important properties, including:

- 1. We know the *dimensions* (in terms of inches, millimeters, etc.) of the object.
- 2. It can be *easily identifiable* in our image (based on either *location* or *appearances*).

Given such a reference object, we can use it compute the size of objects in our image.

Today, we are going to combine the techniques used in the previous blog posts in this series and use these methods to compute the ***distance between objects***.

Keep reading to find out how...

Looking for the source code to this post?
[Jump right to the downloads section.](#)

Measuring distance between objects in an image with OpenCV

Computing the distance *between objects* is very similar to computing the *size* of objects in an image — it all starts with the reference object.

As detailed in our [previous blog post](#), our reference object should have two important properties:

- **Property #1:** We know the dimensions of the object in some measurable unit (such as inches, millimeters, etc.).
- **Property #2:** We can easily find and identify the reference object in our image.

Just as we did last week, we'll be using a US quarter as our reference object which has a width of 0.955 inches (satisfying Property #1).

We'll also ensure that our quarter is always the *left-most* object in our image, thereby satisfying Property #2:



Our goal in this image is to (1) find the quarter and then (2) use the dimensions of the quarter to measure the distance between the quarter and *all other objects*.

Defining our reference object and computing distances

Let's go ahead and get this example started. Open up a new file, name it `distance_between.py` , and insert the following code:

Measuring distance between objects in an image with OpenCV	Python
<pre>1 # import the necessary packages 2 from scipy.spatial import distance as dist 3 from imutils import perspective 4 from imutils import contours 5 import numpy as np 6 import argparse 7 import imutils 8 import cv2 9 10 def midpoint(ptA, ptB): 11 return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5) 12 13 # construct the argument parse and parse the arguments 14 ap = argparse.ArgumentParser() 15 ap.add_argument("-i", "--image", required=True, 16 help="path to the input image") 17 ap.add_argument("-w", "--width", type=float, required=True, 18 help="width of the left-most object in the image (in inches)") 19 args = vars(ap.parse_args())</pre>	

Our code here is near identical to last week. We start by importing our required Python packages on **Lines 2-8**. If you don't already have the `imutils` package installed, stop now to install it:

Measuring distance between objects in an image with OpenCV	Shell
<pre>1 \$ pip install imutils</pre>	

Otherwise, you should upgrade to the latest version (`0.3.6` at the time of this writing) so you have the updated `order_points` function:

Measuring distance between objects in an image with OpenCV	Shell
<pre>1 \$ pip install --upgrade imutils</pre>	

Lines 14-19 parse our command line arguments. We need two switches here: `--image` , which is the path to the input image containing the objects we want to measure, and `--width` , the width (in inches) of our reference object.

Next, we need to preprocess our image:

Measuring distance between objects in an image with OpenCV	Python
<pre>21 # load the image, convert it to grayscale, and blur it slightly 22 image = cv2.imread(args["image"]) 23 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) 24 gray = cv2.GaussianBlur(gray, (7, 7), 0) 25 26 # perform edge detection, then perform a dilation + erosion to 27 # close gaps in between object edges 28 edged = cv2.Canny(gray, 50, 100) 29 edged = cv2.dilate(edged, None, iterations=1) 30 edged = cv2.erode(edged, None, iterations=1) 31</pre>	

Free 21-day crash course on computer vision & image search engines

```
32 # find contours in the edge map
33 cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
34     cv2.CHAIN_APPROX_SIMPLE)
35 cnts = cnts[0] if imutils.is_cv2() else cnts[1]
36
37 # sort the contours from left-to-right and, then initialize the
38 # distance colors and reference object
39 (cnts, _) = contours.sort_contours(cnts)
40 colors = ((0, 0, 255), (240, 0, 159), (0, 165, 255), (255, 255, 0),
41     (255, 0, 255))
42 refObj = None
```

Lines 22-24 load our image from disk, convert it to grayscale, and then blur it using a Gaussian filter with a 7 x 7 kernel.

Once our image has been blurred, we apply the Canny edge detector to detect edges in the image — a dilation + erosion is then performed to close any gaps in the edge map (**Lines 28-30**).

A call to `cv2.findContours` detects the outlines of the objects in the edge map (**Lines 33-35**) while **Line 39** sorts our contours from left-to-right. Since we know that our US quarter (i.e., the *reference object*) will always be the left-most object in the image, sorting the contours from left-to-right ensures that the contour corresponding to the reference object will always be the *first* entry in the `cnts` list.

We then initialize a list of `colors` used to draw the distances along with the `refObj` variable, which will store our *bounding box*, *centroid*, and *pixels-per-metric* value of the reference object.

Measuring distance between objects in an image with OpenCV	Python
<pre>44 # loop over the contours individually 45 for c in cnts: 46 # if the contour is not sufficiently large, ignore it 47 if cv2.contourArea(c) < 100: 48 continue 49 50 # compute the rotated bounding box of the contour 51 box = cv2.minAreaRect(c) 52 box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box) 53 box = np.array(box, dtype="int") 54 55 # order the points in the contour such that they appear 56 # in top-left, top-right, bottom-right, and bottom-left 57 # order, then draw the outline of the rotated bounding 58 # box 59 box = perspective.order_points(box) 60 61 # compute the center of the bounding box 62 cX = np.average(box[:, 0]) 63 cY = np.average(box[:, 1])</pre>	

On **Line 45** we start looping over each of the contours in the `cnts` list. If the contour is not sufficiently large (**Lines 47 and 48**), we ignore it.

Otherwise, **Lines 51-53** compute the rotated bounding box of the current object (using `cv2.cv.BoxPoints` for OpenCV 2.4 and `cv2.boxPoints` for OpenCV 3).

A call to `order_points` on **Line 59** rearranges the bounding box (x, y)-coordinates in top-left, top-right, bottom-right, and bottom-left order, which as we'll see, is important when we go to compute the distance between object corners.

Lines 62 and 63 compute the center (x, y)-coordinates of the rotated bounding box by taking the average of the bounding box in both the x and y direction.

The next step is to calibrate our `refObj` :

Measuring distance between objects in an image with OpenCV	Python
<pre>65 # if this is the first contour we are examining (i.e., 66 # the left-most contour), we presume this is the 67 # reference object 68 if refObj is None: 69 # unpack the ordered bounding box, then compute the 70 # midpoint between the top-left and top-right points, 71 # followed by the midpoint between the top-right and 72 # bottom-right 73 (tl, tr, br, bl) = box 74 (tlblX, tlblY) = midpoint(tl, bl) 75 (trbrX, trbrY) = midpoint(tr, br) 76 77 # compute the Euclidean distance between the midpoints, 78 # then construct the reference object 79 D = dist.euclidean((tlblX, tlblY), (trbrX, trbrY)) 80 refObj = (box, (cX, cY), D / args["width"]) 81 continue</pre>	

If our `refObj` is `None` (**Line 68**), then we need to initialize it.

We start by unpacking the (ordered) rotated bounding box coordinates and computing the midpoint between the top-left and bottom-left along with top-right and bottom-right points, respectively (**Lines 73-75**).

Free 21-day crash course on computer vision & image search engines

From there, we compute the Euclidean distance between the points, giving us our “pixels-per-metric”, allowing us to determine how many pixels fit into `--width` inches.

Note: See [last week’s post](#) for a more detailed discussion of the “pixels-per-metric” variable.

Finally, we instantiate our `refObj` as a 3-tuple consisting of:

1. The sorted coordinates corresponding to the rotated bounding box reference object.
2. The centroid of the reference object.
3. The pixels-per-metric ratio that we’ll be using to determine the distance between objects.

Our next code block handles drawing the contours around our *reference object* and the *object we are currently examining*, followed by constructing `refCoords` and `objCoords` such that (1) the bounding box coordinates and (2) the (x, y)-coordinates of the of the centroid are included in the same arrays:

Measuring distance between objects in an image with OpenCV	Python
<pre>83 # draw the contours on the image 84 orig = image.copy() 85 cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2) 86 cv2.drawContours(orig, [refObj[0].astype("int")], -1, (0, 255, 0), 2) 87 88 # stack the reference coordinates and the object coordinates 89 # to include the object center 90 refCoords = np.vstack([refObj[0], refObj[1]]) 91 objCoords = np.vstack([box, (cX, cY)])</pre>	

We are now ready to compute the distance between the respective corners and centroids of objects in our image:

Measuring distance between objects in an image with OpenCV	Python
<pre>93 # loop over the original points 94 for ((xA, yA), (xB, yB), color) in zip(refCoords, objCoords, colors): 95 # draw circles corresponding to the current points and 96 # connect them with a line 97 cv2.circle(orig, (int(xA), int(yA)), 5, color, -1) 98 cv2.circle(orig, (int(xB), int(yB)), 5, color, -1) 99 cv2.line(orig, (int(xA), int(yA)), (int(xB), int(yB)), 100 color, 2) 101 102 # compute the Euclidean distance between the coordinates, 103 # and then convert the distance in pixels to distance in 104 # units 105 D = dist.euclidean((xA, yA), (xB, yB)) / refObj[2] 106 (mX, mY) = midpoint((xA, yA), (xB, yB)) 107 cv2.putText(orig, "{:.1f}in".format(D), (int(mX), int(mY - 10)), 108 cv2.FONT_HERSHEY_SIMPLEX, 0.55, color, 2) 109 110 # show the output image 111 cv2.imshow("Image", orig) 112 cv2.waitKey(0)</pre>	

On **Line 94** we start looping over pairs of (x, y)-coordinates that correspond to our *reference object* and *object of interest*.

We then draw a circle representing the (x, y)-coordinates of the current points we are computing the distance between and draw a line to connect the points (**Lines 97-110**).

From there, **Line 105** computes the Euclidean distance between the reference location and the object location, followed by dividing the distance by the “pixels-per-metric”, giving us the final *distance in inches* between the two objects. The computed distance is then drawn on our image (**Lines 106-108**).

Note: This distance computation is performed for each of the top-left, top-right, bottom-right, bottom-left, and centroid coordinates for a total of five distance comparisons.

Finally, **Lines 111 and 112** display the output image to our screen.

Distance measurement results

To give our distance measurement script a try, download the source code and corresponding images to this post using the “Downloads” form at the bottom of this tutorial. Unarchive the `.zip` file, change directory to the `distance_between.py` script, and then execute the following command:

Measuring distance between objects in an image with OpenCV	Shell
<pre>1 \$ python distance_between.py --image images/example_01.png --width 0.955</pre>	

Below follows a GIF animation demonstrating the output of our script:

Free 21-day crash course on computer vision & image search engines

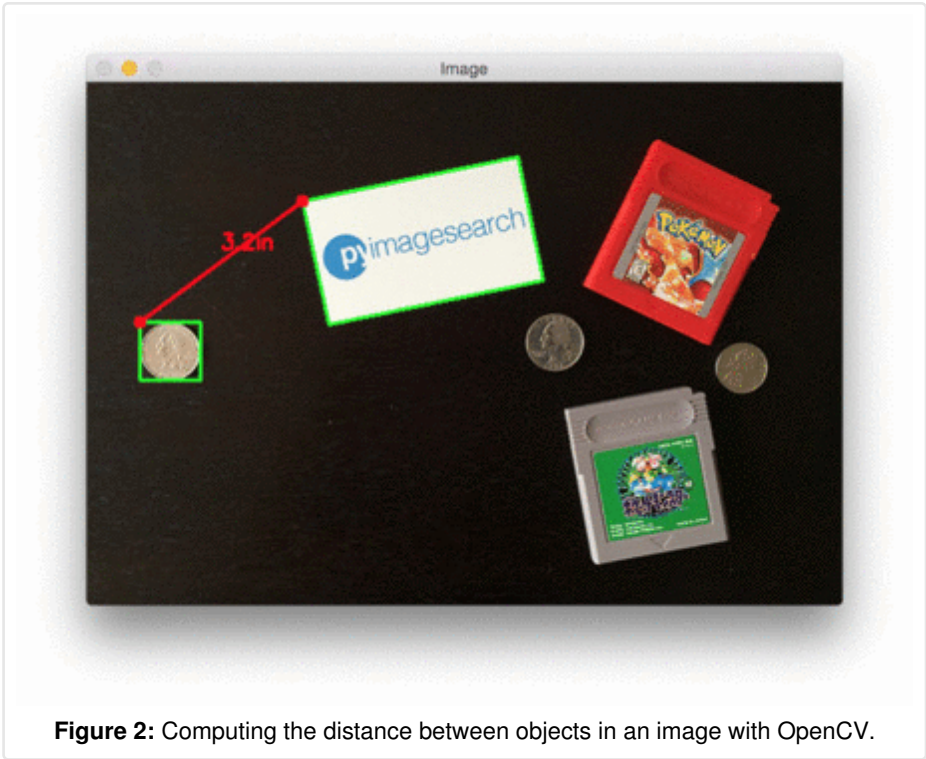


Figure 2: Computing the distance between objects in an image with OpenCV.

In each of these cases, our script matches the top-left (*red*), top-right (*purple*), bottom-right (*orange*), bottom-left (*teal*), and centroid (*pink*) coordinates, followed by computing the distance (in inches) between the reference object and the current object.

Notice how the two quarters in the image are perfectly parallel to each other, implying that the distance between all five control points is 6.1 inches.

Below follows a second example, this time computing the distance between our reference object and a set of pills:

Measuring distance between objects in an image with OpenCV	Shell
1 \$ python distance_between.py --image images/example_02.png --width 0.955	

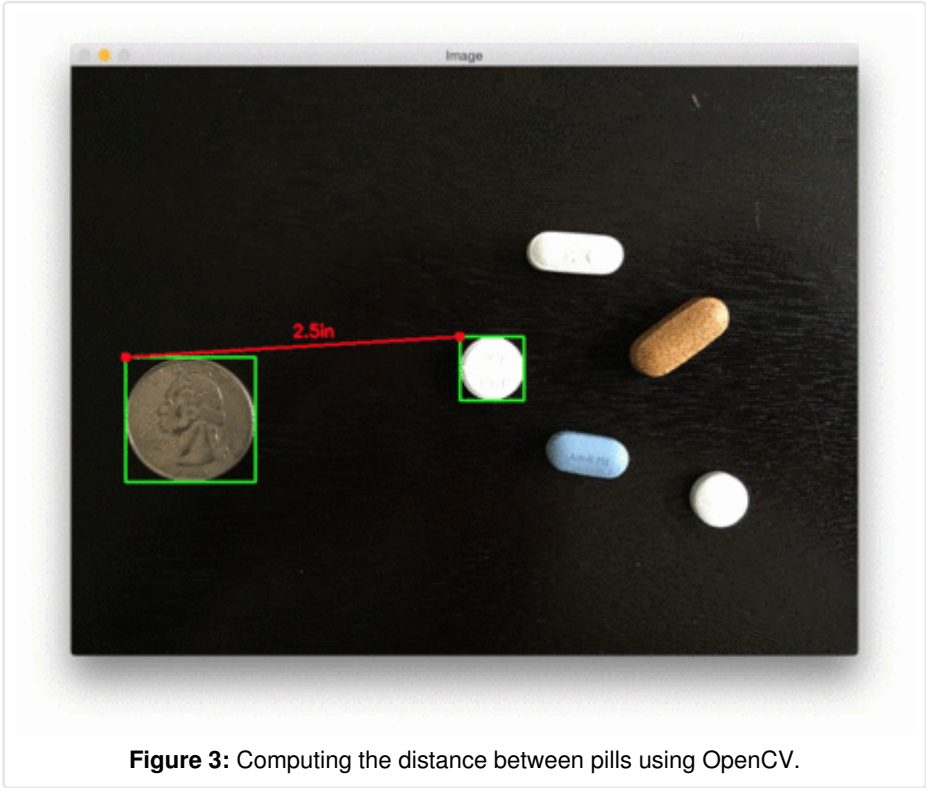


Figure 3: Computing the distance between pills using OpenCV.

This example could be used as input to a pill sorting robot that automatically takes a set of pills and organizes them according to their size and distance from a pill container.

Our last example computes the distance between our reference object (a 3.5in x 2in business card) and a set of 7" vinyl records and an envelope:

Measuring distance between objects in an image with OpenCV	Shell
1 \$ python distance_between.py --image images/example_03.png --width 3.5	

Free 21-day crash course on computer vision & image search engines

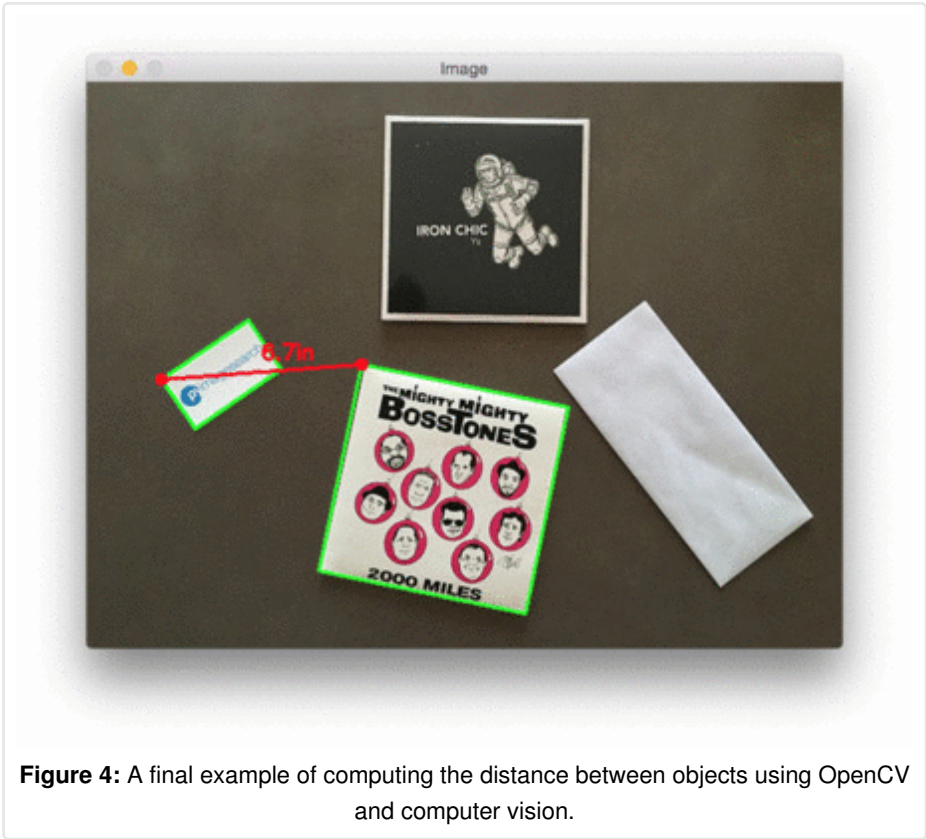


Figure 4: A final example of computing the distance between objects using OpenCV and computer vision.

As you can see, in each of these cases, we have successfully computed the distance (in actual, measurable units) between objects in an image.

Summary

In the third and final installment in our series on measuring object sizes, we learned how to take two different objects in an image and compute the distance between them in *actual measurable units* (such as inches, millimeters, etc.).

Just as we found out in [last week's post](#), before we can (1) compute the size of an object or (2) measure the distance between two objects, we first need to compute the “pixels-per-metric” ratio, used to determine how many pixels “fit” into a given unit of measurement.

Once we have this ratio, computing the distance between objects is almost trivially easy.

Anyway, I hope you enjoyed this series of blog posts! If you have any suggestions for a future series, please leave a comment on [shoot me a message](#).

And before you go, be sure to *signup for the PyImageSearch Newsletter by entering your email address in the form below!*

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 11-page Resource Guide** on Computer Vision and Image Search Engines, including **exclusive techniques** that I don't post on this blog! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

DOWNLOAD THE CODE!

Resource Guide (it's totally free).



Enter your email address below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own!

Your email address

DOWNLOAD THE GUIDE!


Free 21-day crash course on computer vision & image search engines

🔖 calibration, image processing, object size, pixels per metric, point ordering

< Measuring size of objects in an image with OpenCV

Finding extreme points in contours with OpenCV >

42 Responses to *Measuring distance between objects in an image with OpenCV*



AJ

April 4, 2016 at 5:46 pm

#


REPLY

↩

Thanks for the great tutorial, I have one question.

if I don't have a reference object but I know the distance from the left edge of the image to the right edge can I find the distance from any object to the center of the image ?

Best Regards



Adrian Rosebrock


April 6, 2016 at 9:15 am

#

REPLY

↩

Yep, you absolutely can! The same ratio test applies.



ghanendra

April 5, 2016 at 6:05 am


#

REPLY

↩

Hey Adrian Its a Great Tutorial, thanks a lot I used it in my project for determining distance between the two bright objects.

What If the reference object dimensions are continuously varying?? How to determine distance between the two variable size objects in a live stream video??



Adrian Rosebrock


April 6, 2016 at 9:14 am

#

REPLY

↩

Can you elaborate on what you mean by “continuously varying”? If you have different reference objects, you'll need to identify which one you are looking at. This can be done using all sorts of methods including object detection, keypoint matching, template matching, etc.



ghanendra

April 6, 2016 at 10:28 pm


#

REPLY

↩

In your tutorial about Ball Tracking with opencv, when we move the ball back and forth in front of camera, the size of the contour varies accordingly, So taking this as reference, how to find distance between two balls??

I have to detect the distance between the two head lights of a car and distance from the camera when car is moving.



Adrian Rosebrock


April 7, 2016 at 12:54 pm

#

REPLY

↩

You'll need to combine two calibration methods. The first (this blog post) to measure the distance between objects in images. Then, you can use [this blog post](#) to measure the distance from the camera to the object.




ghanendra

April 7, 2016 at 11:32 pm

#

Really thanks a lot Adrian!!!!!!!




Adrian Rosebrock

April 8, 2016 at 12:54 pm

#

No problem ☐



Mahed

April 5, 2016 at 10:58 am

#

REPLY

↩

Well ... you know what they, there really is no off-switch to genius !!!

Free 21-day crash course on computer vision & image search engines

Cant wait to see what Adrian has got brilliant plans for PyImage Gurus bit ☐



Adrian Rosebrock April 6, 2016 at 9:10 am #

REPLY ↩

Thanks Mahed! I have some pretty special announcements related to PyImageSearch Gurus coming in the next 1-2 months, so be sure to keep an eye on the blog ☐



Jim April 9, 2016 at 7:05 pm #

REPLY ↩

Hello Adrian,

can you explain what this line does:
cnts = cnts[0] if imutils.is_cv2() else cnts[1]

Thanks
Jim



Adrian Rosebrock April 13, 2016 at 7:11 pm #

REPLY ↩

That line of code handles if you are using OpenCV 2.4 or OpenCV 3. There is a difference in the tuple returned by OpenCV between OpenCV 2.4 and OpenCV 3. You can read more about the change to cv2.findContours [in this blog post](#).



Marcus W. April 29, 2016 at 4:54 am #

REPLY ↩

Hey, nice post.
I guess its a stupid question, but how can i loop only over the center of the objects.
Thanks for any help ^^



Adrian Rosebrock April 30, 2016 at 4:01 pm #

REPLY ↩

The center of the coordinates are stored in cX and cY. You can just loop over those and ignore the other ☐



suresh June 7, 2016 at 10:44 am #

REPLY ↩

great work bro . u r really awesome ...
can u plzz explain how to find distance between two tracked objects in a live video



Adrian Rosebrock June 7, 2016 at 3:11 pm #

REPLY ↩

The same general algorithm as discussed in this blog post needs to be applied. You first need to “calibrate” the camera. From there, determining the distance between two objects in a video stream is the same as determining the distance between two objects in an image — you just need to [access the video stream](#) and then process every frame of the stream.



vinaya June 16, 2016 at 5:00 am #

REPLY ↩

how to measure eye to eye distance?? can this be done similar way?




Adrian Rosebrock June 18, 2016 at 8:22 am #

REPLY ↩

If you can detect both eyes in an image, then yes, you can use this method to compute the distance between them.


Free 21-day crash course on computer vision & image search engines



priyanka July 1, 2016 at 6:09 am <#>

REPLY ↩


what values are stored in (xA, yA), (xB, yB) ..please explain



Adrian Rosebrock July 1, 2016 at 3:01 pm <#>

REPLY ↩


(xA, yA) stores the (x, y)-coordinates of the reference image, meanwhile (xB, yB) store the (x, y)-coordinates of the object we are computing the distance to.



Raycuz July 21, 2016 at 8:08 am <#>

REPLY ↩


Is that possible if I want to draw all the lines of distances between objects and saved it in a single image..??



Raycuz July 21, 2016 at 8:40 am <#>

REPLY ↩


Found it
Thank you very much
You gave an awesome guide for beginners



Jacki June 6, 2017 at 7:40 pm <#>

REPLY ↩


Plz if you can explain for me
How i can draw all rectangles in a single image?
Thanks in advance



Adrian Rosebrock June 9, 2017 at 1:57 pm <#>

REPLY ↩


Draw the lines on the original image loaded on Line 22 (rather than orig). Then save the image to disk once all lines are drawn via cv2.imwrite.



Jacki June 10, 2017 at 10:08 pm <#>

REPLY ↩

Dear Dr. Adrian;
Thanks in advance for your reply
I do it. Now i can save all result in a single image, but i have a problem:
When i delet line 111 [cv2.imshow("Image", orig)] when i want to stop show result on the screen, for loop not work perfectly!!
Just work for 3 object ((it save result just for 2 or 3 object)) not for all object
Please if you can help me ?
How i can cancel line 111 and run program with out error ??




Jacki June 11, 2017 at 5:38 pm <#>


REPLY ↩

Thanks a lot dear Dr. Adrian i do it
I removed line 112
cv2.waitKey(0)
And now works fine without problem
Your blog post very awesome
Thanks again


Free 21-day crash course on computer vision & image search engines




Agnel Vishal November 20, 2016 at 8:09 am <#>

REPLY 

I think some modifications are required if the distance between reference object and camera are different from other objects and camera. What should be done then?




Abeer December 9, 2016 at 2:15 am <#>


REPLY 

Hi,


I see that the calculated distance is between the reference object and all other objects, what if I want to measure the distance between two objects that both are non reference objects?




Adrian Rosebrock December 10, 2016 at 7:13 am <#>

REPLY 


You can certainly do that as well. The important part is that you find your reference object *first*. Your reference object allows you to compute the number of pixels per metric. As long as you have this, you can compute the distance between two non-reference objects.




Sam March 9, 2017 at 3:48 am <#>

REPLY 

Hi Adrian. Is the distance between two objects affected by the viewpoint of camera or the camera lens angle?



Adrian Rosebrock March 10, 2017 at 3:52 pm <#>

REPLY 

Yes, it absolutely is. In order to correct for this it's common to calibrate our camera by computing the intrinsic properties of the camera. This allows us to help us correct for distortion. For these measurements to be correct you should have an (approximately) 90 degree, birds-eye-view of the objects.




shahid March 28, 2017 at 7:57 am <#>


REPLY 

hi adrian!


Is this code suitable for android app development?




Adrian Rosebrock March 28, 2017 at 12:49 pm <#>

REPLY 

You would need to convert the code from Python to Java + OpenCV for Android, but the same computer vision algorithms can be used.



Ahmed May 7, 2017 at 3:44 am <#>


REPLY 

Hi Adrian,


Thanks a lot for your tutorials .. I an still a beginner in Opencv and I want to use the same approach but with camera .. how to determine the distance between two Pink boxes by Camera ?

What is the simplest code for that ?


Thx



Adrian Rosebrock May 8, 2017 at 12:27 pm <#>


REPLY 

I would suggest starting with [this blog post](#) to learn how to measure the distance between a camera and an object. Once you've detected both objects you can apply the same triangle similarity technique to measure the distance between the objects.



David L. May 28, 2017 at 5:30 am <#>

Free 21-day crash course on computer vision & image search engines

REPLY 

Do you have code to convert a perspective image into a birds-eye-view?



Adrian Rosebrock May 31, 2017 at 1:31 pm #

REPLY ↩

Yes, please see [this blog post](#).



Faisal June 4, 2017 at 7:00 am #

REPLY ↩

Hi,

The tutorial is awesome, but i need to adapt it for a video. So it can detect distances in real time. Could you please help me with it?

Regards



Adrian Rosebrock June 6, 2017 at 12:11 pm #

REPLY ↩

The same techniques applied to single images can also be applied to video. I would suggest reading [this blog post](#) on accessing video streams. If you are new to OpenCV and Python, be sure to read through [Practical Python and OpenCV](#) as this book will give you the fundamentals you need to be successful porting the algorithm from single images to video.



Anil June 19, 2017 at 10:44 am #

REPLY ↩

Hi Adrian!!!!

The tutorial was great. Thank you for that.

I wanted to know if i have a set of parallel lines, then can i get the distance between two consecutive lines instead of with reference to just the first one???



Anil June 20, 2017 at 2:10 am #

REPLY ↩

What would be the execution time for this code???

If given an image, in how much time can it provide the answer on a raspberry pi based camera?



Adrian Rosebrock June 20, 2017 at 11:21 am #

REPLY ↩

Regardless if your lines or parallel you would still need to be able to detect them both, otherwise how would you be able to know the distance between two arbitrary points on either line?

As for the execution time, yes, you could certainly run this script on a Raspberry Pi.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Free 21-day crash course on computer vision & image search engines

Resource Guide (it's totally free).



Click the button below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own.

Download for Free!

Deep Learning for Computer Vision with Python Book



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 PRE-ORDER MY NEW BOOK

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!

Free 21-day crash course on computer vision & image search engines


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 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
Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3 APRIL 18, 2016
How to install OpenCV 3 on Raspbian Jessie OCTOBER 26, 2015
Basic motion detection and tracking with Python and OpenCV MAY 25, 2015
Accessing the Raspberry Pi Camera with OpenCV and Python MARCH 30, 2015
Install OpenCV 3.0 and Python 2.7+ on Ubuntu JUNE 22, 2015

Free 21-day crash course on computer vision & image search engines

Search

Search...

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.
© 2017 PyImageSearch. All Rights Reserved.

Free 21-day crash course on computer vision & image search engines