Multiscale Template Matching

By Nishad Patil

Technology Stack Used

1] OpenCV

2] Scaling Algorithm

3] Numpy

4] imutils library

5] Triangle Similarity

Introduction to Technology Stack

OpenCV

Most used computer vision library. Highly efficient. Facilitates real-time image processing.

imutils

collection of helper functions and utilities to make working with OpenCV easier.

Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Task to be Performed

1. Performs Template Matching on static images as well as live Web cam feed.

2. Able to match Template on different Scales and Should Consume minimum time for each frame

Need to use separate Scaling Algorithm for Template Matching

We are using the standard approach to template matching using cv2.matchTemplate() but it is not very robust.

Most of the times, when we try to apply template matching using the cv2.matchTemplate function, we are left with a false match — this is because the size of the Template image on the *left* is substantially smaller than the real time image Feed on the Web Cam live Feed.

Given that the dimensions of the template does not match the dimensions of the real time image Feed on the Web Cam live Feed, we are left with a false detection.

Working of Scaling Algorithm

1. Loop over the input image at multiple scales (i.e. make the input image progressively smaller and smaller).

2. Apply template matching using cv2.matchTemplate and keep track of the match with the largest correlation coefficient (along with the *x, y*-coordinates of the region with the largest correlation coefficient).

3. After looping over all scales, take the region with the largest correlation coefficient and use that as your "matched" region.

Working Animation

https://pyimagesearch.com/wp-content/uploads/2015/01/cod_blackops_animated.gif

Challenges Solved

First challenge I come across in it takes a lot of time for Scaling each frame from web Cam resulting in lot of time Consumption

And the solution I found is that I change the Scaling algorithm parameters like initial point and gap between frame

The Second Challenge is that our model is putting rectangular box around the matched template in image when template is

matched and when template is not present in image at that time it put box around any random position where it match few pixels I solved it by checking Max Val from template Match function and put a condition to place rectangular box only if most of the pixels are match

Limitations and Drawbacks

While we can handle variations in translation and scaling, our approach will not be robust to changes in rotation or non-affine transformations.

If we are concerned about rotation on non-affine transformations we are better off taking the time to detect keypoints, extract local invariant descriptors, and apply keypoint matching.

But in the case where our templates are (1) fairly rigid and well-defined via an edge map and (2) we are only concerned with translation and scaling, then multi-scale template matching can provide us with very good results with little effort.

Hardware Limitations

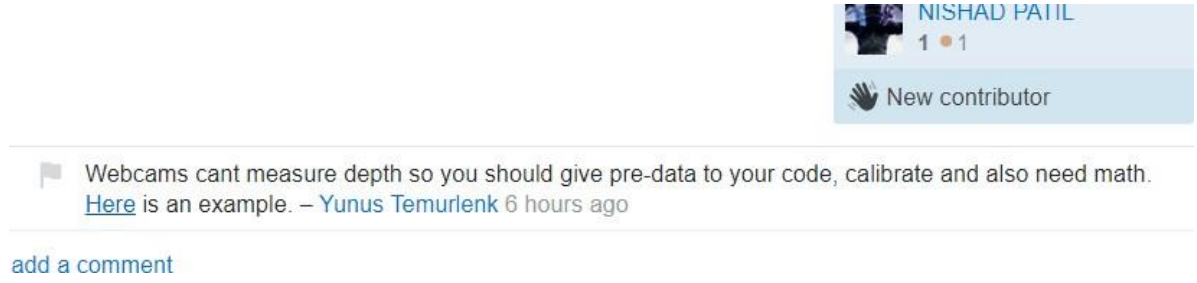
Measuring Distance between object and Camera is not possible through Web Cam it require different Hardware like Stereo Camera which has in build functionalities that we can use for this measuring distance

Another option is if we want to measure the distance through web camp then we have to provide some predefined values like

Known distance, focallength and Known width of object like that have shown in this link

https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/

But it would be only for one time purpose its not suitable for real time environment



Distance Measurement Between Matched Template and Camera

In order to determine the distance from our camera to a known object or marker, we are going to utilize *triangle similarity*.

The triangle similarity goes something like this: Let's say we have a marker or object with a known width *W*. We then place this marker some distance *D* from our camera. We take a picture of our object using our camera and then measure the apparent width in pixels *P*. This allows us to derive the perceived focal length *F* of our camera:

$F = (P \times D) / W$

For example, let's say I place a standard piece of *8.5 x 11in* piece of paper (horizontally; *W = 11*) *D = 24 inches* in front of my camera and take a photo. When I measure the width of the piece of paper in the image, I notice that the perceived width of the paper is *P = 248 pixels*.

My focal length *F* is then:

$F = (248px \times 24in) / 11in = 543.45$

As I continue to move my camera both closer and farther away from the object/marker, I can apply the triangle similarity to determine the distance of the object to the camera:

$D' = (W \times F) / P$

Again, to make this more concrete, let's say I move my camera *3 ft* (or 36 inches) away from my marker and take a photo of the same piece of paper. Through automatic image processing I am able to determine that the perceived width of the piece of paper is now *170 pixels*. Plugging this into the equation we now get:

$D' = (11in \times 543.45) / 170 = 35in$

Or roughly 36 inches, which is 3 feet.

Referred blog

https://www.pyimagesearch.com/2015/01/26/multi-scale-template-matching-using-python-opencv/

https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/

https://stackoverflow.com/questions/62758294/measuring-distance-of-matched-template-in-live-feed-from-web-cam-using-computer?noredirect=1#comment110999316_62758294