Exercise 8: BLOBs and SIFT features

02504 Computer vision

Morten R. Hannemose, mohan@dtu.dk, DTU Compute

March 22, 2024

Learning objectives

These exercises will introduce you to BLOBs and SIFT features. In this exercise you will write the code for the blob detector as well as use the SIFT feature detector and matcher.

BLOB detector

You will implement a BLOB detector using the Difference-of-Gaussians (DoG) method and apply it to the following image of sunflowers.

Start by loading the image into Python, and converting it to black and white and floating point.
`im = im.astype(float).mean(2)/255`

Exercise 8.1

Create the function `im_scales = scaleSpaced(im, sigma, n)`, where `im_scales` is a list containing the scale space pyramid of the original image `im`. The width and height of all images in the pyramid `im_scales` are exactly the same as the original image `im`. I.e. here we do the naïve implementation with increasing widths of Gaussians and no image downsampling, to make the exercise easier. In other words `im_scales` is not a pyramid in image sizes; only in scale space.

This function should apply a Gaussian kernel with standard deviation $\texttt{sigma} \cdot 2^i$, where $i = 0, 1, \ldots, \texttt{n} - 1$.

Exercise 8.2

Now, create the function `DoG = differenceOfGaussians(im, sigma, n)`, where `DoG` is a list of scale space DoGs of the original image `im`. Like the `scaleSpaced` function, the returned images are all the same size as the original.

Exercise 8.3

Finally, create the function `blobs = detectBlobs(im, sigma, n, tau)`, where `blobs` are the BLOBs (pixels) of the original image `im` with a DoG magnitude larger than a threshold `tau`. The function should use non-maximum suppression to only give a single response for each BLOB. Reuse or extend your code from week 6 to do the non-maximum suppression for the 8 neighbours in the same scale. Instead of checking against all nine neighbours in the scale above and below, you can apply a 3×3 max filter to all scales above and below, and only compare against the center pixel in the max filtered image above and below.

```
MaxDoG[i] = cv2.dilate(abs(DoG[i]), np.ones((3,3)))
```

Try the detector on the image of sunflowers. Visualize your result by drawing a circle for each BLOB, with the radius proportional to the scale of the BLOB. You can use `cv2.circle` for this.

Figure 1: Detected blobs with `sigma`=2, `n`=7, `tau`=0.1. You may have slight deviations in the detected blobs depending on how you choose

Using SIFT features

The SIFT feature detector and descriptor are quite difficult to implement, so we will use existing implementations. However, first we need a good test case scenario.

Exercise 8.4

Create the function `r_im = transformIm(im, theta, s)`, where `r_im` is a scaled and rotated version of the original image `im`. In this case, `theta` is a rotation angle and `s` is a scale factor.

Use this function to produce a transformed version of the original image.

Exercise 8.5

Use the SIFT detector to detect features in both the original and the transformed image. Plot the features on top of the images. There are quite a few parameters to play with. Try changing them and see the results.

Now match the features to each other. For this you can use `cv2.BFMatcher()`.

Plot the matches; do they look qualitatively correct?

Filter your matches with the ratio test. Does this remove incorrect matches?

Exercise 8.6

Take two photos of the same scene from different angles using e.g. your smartphone and find matching SIFT features.

Exercise 8.7

This is an optional exercise. Try downloading R2D2 and use it to match features in your images.