

## Assignment 2: Scale-space blob detection (Python)

**Due date: Monday, March 12, 11:59:59PM**

The goal of the assignment is to implement a Laplacian blob detector as discussed in the [this lecture](#).



### Algorithm outline

1. Generate a Laplacian of Gaussian filter.
2. Build a Laplacian scale space, starting with some initial scale and going for  $n$  iterations:
  1. Filter image with scale-normalized Laplacian at current scale.
  2. Save square of Laplacian response for current level of scale space.
  3. Increase scale by a factor  $k$ .
3. Perform nonmaximum suppression in scale space.
4. Display resulting circles at their characteristic scales.

### Test images

Here are [four images](#) to test your code, and [sample output images](#) for your reference. Keep in mind, though, that your output may look different depending on your threshold, range of scales, and other implementation details. In addition to the images provided, also **run your code on at least four images of your own choosing**.

### Detailed instructions

Adapted from MATLAB to Python by Hsiao-Ching Chang

- You need the following Python libraries: `numpy`, `scipy`, `scikit-image`, `matplotlib`.
- Don't forget to convert images to grayscale (use `convert('L')` if you are using PIL) and rescale the intensities to between 0 and 1 (simply divide them by 255 should do the trick).
- For creating the Laplacian filter, use the `scipy.ndimage.filters.gaussian_laplace` function. Pay careful attention to setting the right filter mask size. **Hint:** Should the filter width be odd or even?
- It is relatively inefficient to repeatedly filter the image with a kernel of increasing size. Instead of increasing the kernel size by a factor of  $k$ , you should downsample the image by a factor  $1/k$ . In that case, you will have to upsample the result or do some interpolation in order to find maxima in scale space. **For full credit, you should turn**

**in both implementations: one that increases filter size, and one that downsamples the image.** In your report, list the running times for both versions of the algorithm and discuss differences (if any) in the detector output. For timing, use `time.time()`.

**Hint 1:** think about whether you still need scale normalization when you downsample the image instead of increasing the scale of the filter.

**Hint 2:** Use `skimage.transform.resize` to help preserve the intensity values of the array.

- You have to choose the initial scale, the factor  $k$  by which the scale is multiplied each time, and the number of levels in the scale space. I typically set the initial scale to 2, and use 10 to 15 levels in the scale pyramid. The multiplication factor should depend on the largest scale at which you want regions to be detected.
- You may want to use a three-dimensional array to represent your scale space. It would be declared as follows:

```
scale_space = numpy.empty((h,w,n)) # [h,w] - dimensions of image, n - number of levels in scale space
```

Then `scale_space[:, :, i]` would give you the  $i$ -th level of the scale space. Alternatively, if you are storing different levels of the scale pyramid at different resolutions, you may want to use an NumPy object array, where each "slot" can accommodate a different data type or a matrix of different dimensions. Here is how you would use it:

```
scale_space = numpy.empty(n, dtype=object) # creates an object array with n "slots"
scale_space[i] = my_matrix # store a matrix at level i
```

- To perform nonmaximum suppression in scale space, you should first do nonmaximum suppression in each 2D slice separately. For this, you may find functions `scipy.ndimage.filters.rank_filter` or `scipy.ndimage.filters.generic_filter` useful. Play around with these functions, and try to find the one that works the fastest. To extract the final nonzero values (corresponding to detected regions), you may want to use the `numpy.clip` function.
- You also have to set a threshold on the squared Laplacian response above which to report region detections. You should play around with different values and choose one you like best. To extract values above the threshold, you could use the `numpy.where` function.
- To display the detected regions as circles, you can use [this function](#) (or feel free to search for a suitable Python function or write your own). **Hint:** Don't forget that there is a multiplication factor that relates the scale at which a region is detected to the radius of the circle that most closely "approximates" the region.

## For extra credit

- Implement the difference-of-Gaussian pyramid as mentioned in class and described in [David Lowe's paper](#). Compare the results and the running time to the direct Laplacian implementation.
- Implement the affine adaptation step to turn circular blobs into ellipses as shown in the lecture (just one iteration is sufficient). The selection of the correct window function is essential here. You should use a Gaussian window that is a factor of 1.5 or 2 larger than the characteristic scale of the blob. Note that the lecture slides show how to find the relative shape of the second moment ellipse, but not the absolute scale (i.e., the axis lengths are defined up to some arbitrary constant multiplier). A good choice for the absolute scale is to set the sum of the major and minor axis half-lengths to the diameter of the corresponding Laplacian circle. To display the resulting ellipses, you should modify the circle-drawing function or look for a better function in the `matplotlib` documentation or on the Internet.
- The Laplacian has a strong response not only at blobs, but also along edges. However, recall from the class lecture that edge points are not "repeatable". So, implement an additional thresholding step that computes the Harris response at each detected Laplacian region and rejects the regions that have only one dominant gradient orientation (i.e., regions along edges). If you have implemented the affine adaptation step, these would be the regions whose characteristic ellipses are close to being degenerate (i.e., one of the eigenvalues is close to zero). Show both "before" and "after" detection results.

## Helpful resources

- [Sample Harris detector using scikit-image](#).
- [Blob detection](#) on Wikipedia.
- D. Lowe, "[Distinctive image features from scale-invariant keypoints](#)," International Journal of Computer Vision, 60 (2), pp. 91-110, 2004. This paper contains details about efficient implementation of a Difference-of-Gaussians scale space.
- T. Lindeberg, "[Feature detection with automatic scale selection](#)," International Journal of Computer Vision 30 (2), pp. 77-116, 1998. This is advanced reading for those of you who are *really* interested in the gory mathematical details.

## Grading Checklist

As before, you must turn in both your report and your code. Your report will be graded based on the following items:

1. The output of your circle detector on all the images (four provided and four of your own choice), together with running times for both the "efficient" and the "inefficient" implementation.
2. An explanation of any "interesting" implementation choices that you made.
3. An explanation of parameter values you have tried and which ones you found to be optimal.
4. Discussion and results of any extensions or bonus features you have implemented.

## Instructions for Submitting the Assignment

Your submission should consist of the following:

- All your Python code and output images **in a single zip file**. The filename should be **lastname\_firstname\_a2.zip**.
- A brief report **in a single PDF file** with all your results and discussion. The filename should be **lastname\_firstname\_a2.pdf**.

As before, the files will be submitted through [Compass 2g](#). Upload instructions:

1. Log into <https://compass2g.illinois.edu>, find this class and Assignment 2.
2. Upload **(1) your PDF report** and **(2) the zip file** as two attachments.
3. Hit **Submit**.

Multiple attempts will be allowed but by default, only your last submission will be graded. **We reserve the right to take off points for not following directions.**

**Late policy:** You lose 25% of the points for every day the assignment is late. If you have a compelling reason for not being able to submit the assignment on time and would like to make a special arrangement, you must send me email **at least a week before the due date** (any genuine emergency situations will be handled on an individual basis).

**Academic integrity:** Feel free to discuss the assignment with each other in general terms, and to search the Web for general guidance (not for complete solutions). Coding should be done individually. If you make substantial use of some code snippets or information from outside sources, be sure to acknowledge the sources in your report. At the first instance of cheating (copying from other students or unacknowledged sources on the Web), a grade of zero will be given for the assignment. At the second instance, you will automatically receive an F for the entire course.