CS 682

Harris Corners and Scale-space blob detection

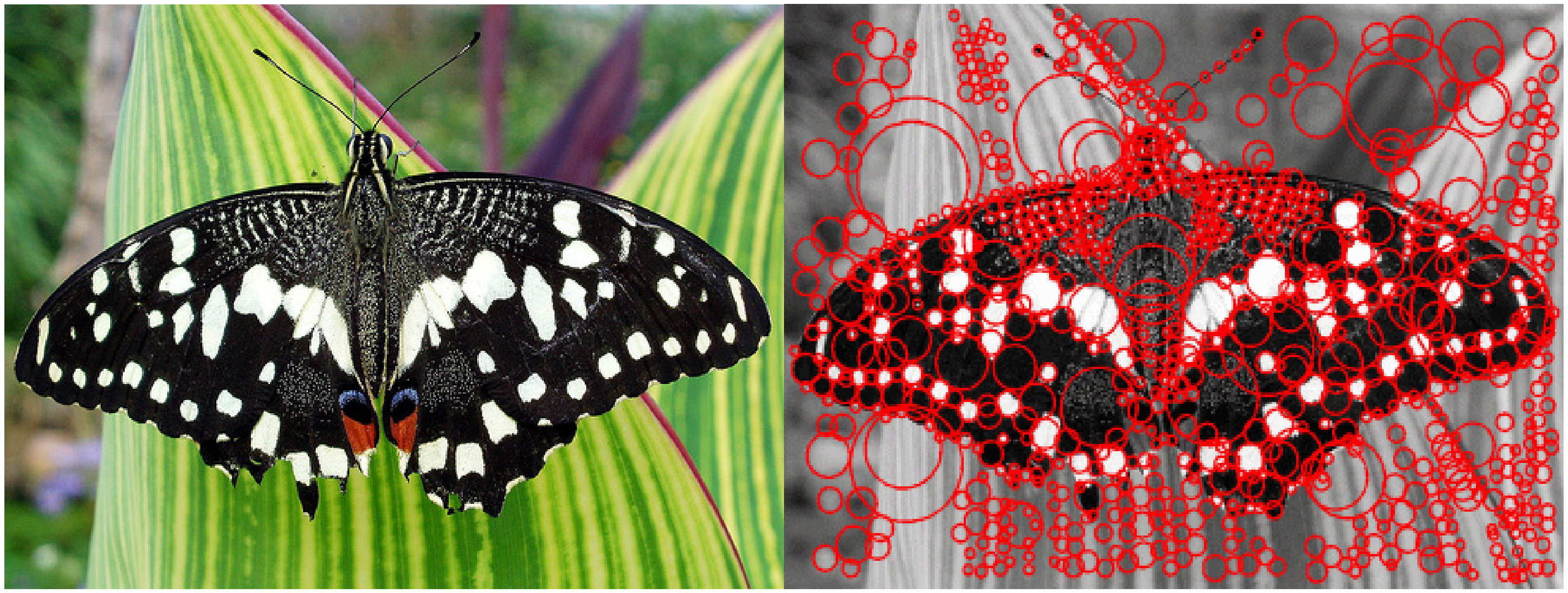
(10) Part 1: Corner Detector

In this problem, you will use the implementation of the Harris corner detector. You will be given a starter code and a set of images for testing. You will need to fill in your code for the function `compute_harris_response` to compute the Harris response. Upon completion, visualize the Harris response image, and display the final detected Harris corner points on top of the input image. You can download the starter code from here. Run the corner detector, visualize the results and answer the questions below.

- For fixed parameter values, run the detector on `house1.jpg` and `house1-rotated.jpg` . The latter image is a rotated copy of the former. If we rotate the input image, do the detected corner positions rotate by the same amount ? Justify your answer based on your observations.
- If we scale down the input image, are all the detected corner positions scaled accordingly ? You can test this experimentally by comparing the corner detection on the images `house1-2down.jpg, house1-4down.jpg` . Each image in this sequence is half the size of its predecessor. Justify your answer based on your observations.
- As part of this problem you had to compute derivatives Ix and Iy. Upon completion display the following four images Ix, Iy, gradient magnitude, and gradient orientation for the image of our choice.

(15) Part 2: Scale-space blob detection

The goal of this part of the assignment is to implement a Laplacian blob detector. .



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

- 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.

Grading Checklist

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

Part 1:

1. The house1 image and the corners detected in the image and the rotated and scaled version of the image.
2. The image and the visualization of gradient magnitude and orientation.
3. You can do it all in single notebook file and answer the questions in the written part of the report.

Part 2:

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.

Submission Instructions:

You must upload the files to GMU Blackboard.

1. Your code **in two Python notebooks**. The filenames should be **netid_hw2_part1_code.ipynb** and **netid_hw2_part2_code.ipynb**.
2. Your ipython notebooks with output cells converted to **PDF format**. The filenames should be **netid_hw2_part1_output.pdf** and **netid_hw2_part2_output.pdf**.
3. A single report for both parts in PDF format. The filename should be **netid_hw2_report.pdf**.

There is no need to turn in a zip file with all your input images, but be sure to show all images in your report.

Don't forget to hit "Submit" after uploading your files, otherwise we will not receive your submission.

Useful Python Links

- Scipy Gaussian Filter
- OpenCV Filter2D