**TODO:**

**Rerun q13 and q14 because rectW and rectH is changed**

Change q14 to display also the old estimation

Hence,

where is the identity matrix.

where and . Note that , and .

For a unique solution of to be found, should be non-singular.

Appearance Basis

Since the basis for are orthogonal to each other, if . Therefore, to find for some , we dot both sides with :

Hence, .

We consider the new image as . If is an orthonormal basis, then . reduces to

We let . If we let , then it becomes clear that

Note by orthogonal decomposition, . Note that and . We can rewrite in terms of :

This is the form we will pass into np.linalg.lstsq. We also see that , so .

Hence,

and

where . Now .

Affine Additive

Note that in this question, the final affine transformation matrix will follow the conventions of being the horizontal axis and being the vertical axis, instead of the system used in scipy.ndimage.transform where and are the first and second axis of the image array.

Inverse Compositional

Our new objective function:

and our update rule is .

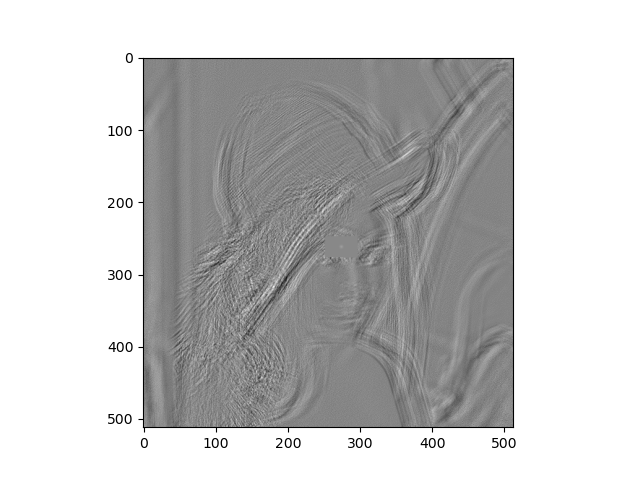
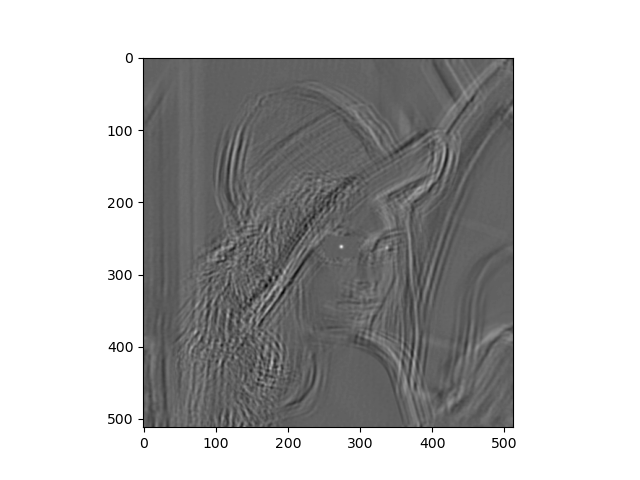
This approach is computationally more efficient since the pseudoinverse of the term in the inverse compositional method can be precomputed, while the in the classical approach has to be recalculated every time the is updated in each iteration. Calculating inverse takes a lot of time, especially when the s contains around 75k rows.

To truly utilize the efficient of precomputation, instead of selecting points common to both images’ valid area, we assume that the motion between two images are minimal and we only select points from a smaller rectangle. Both versions of the affine matrix alignment are written in SubtractDominantMotion.py. Uncomment the code to run each version.

It should be emphasized that the motivation for correlation filters is to find a filter , the provides us with the desired response , from a series of discriminators extracted from some part of the image.

Differentiate the objective function with respective to to find the minimum:

The minimum is attained when the gradient is zero:



The performance we’re trying to asses is how well the correlation filter is generated. Viewing the code, the desired response is , where is the distance away from the center of Lena’s eye; the less the distance, the stronger the response. This means that we want to fit a filter which gives a clear, strong response for an eye. Comparing the responses generated by the two correlation filters with different regularization, we find that gives a strong response for the eye they we didn’t use for fitting , while we only see random noise for .

The reason for this is gives more regularization than , limiting from straying away from the origin (which gives the default response; the response to any image patch would evaluate to 0) and effectively preventing from overfitting to specifically the left eye. In other terms, provides a trust region while doesn’t.

The response is different because convolution flips the filter horizontally and vertically before element-wise multiplication with the cropped image window, but correlation applies the given filter directly. To get a similar response, flip the filter before passing it into scipy.ndimage.convolve, like filter[::-1,::-1]. This means that the following two statements should give the same results:

scipy.ndimage.correlate(img, filter)

scipy.ndimage.convolve(img, filter[::-1,::-1])