Image and Vision Computing : Lab 2

This lab includes three parts. We will use Matlab environment throughout. In part one we will implement a version of the Harris keypoint detector. In part two, implement a simple descriptor matching method and use it for object matching. In part three, implement a smiple RANSAC routine for robust matching.

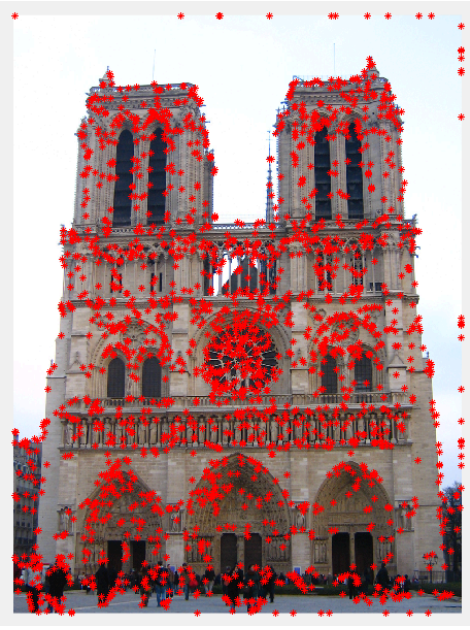
Part 1: Harris Detector implementation

Try to program a simple implementation of the Harris corner detector. You can use the provided **`lab2part1.m`** as a template if you wish. Load an image, compute the Harris keypoints and plot them on the image. You will first get the requisite gradients using convolution as studied last week. Then compute the Harris matrix at each point, the Harris score at each point, and threshold this score to obtain the keypoint detections. A summary of required steps is given by Algorithm 4.1 of the Szeliski book. This is reprinted below, but see the lecture notes and Szeliski book Ch 4.1.1 for details. For simplicity you can compute the Harris score at each point pointwise rather than summing over a local window.

1. Compute the horizontal and vertical derivatives of the image I_x and I_y by convolving the original image with derivatives of Gaussians (Section 3.2.3).
2. Compute the three images corresponding to the outer products of these gradients. (The matrix $\boldsymbol{A}$ is symmetric, so only three entries are needed.)
3. Convolve each of these images with a larger Gaussian.
4. Compute a scalar interest measure using one of the formulas discussed above.
5. Find local maxima above a certain threshold and report them as detected feature point locations.

Algorithm 4.1 Outline of a basic feature detection algorithm.

The result applied to the provided image should look like something this:



You can compare your results to the output of matlab commands for local feature detection such as `detectHarrisFeatures()` and `detectSURFFeatures()`.

- Bonus [Moderate]: The naïve way to implement the Harris score computation involves looping over all pixels which is slow. Figure out how to vectorise the algorithm and eliminate the for loop entirely.

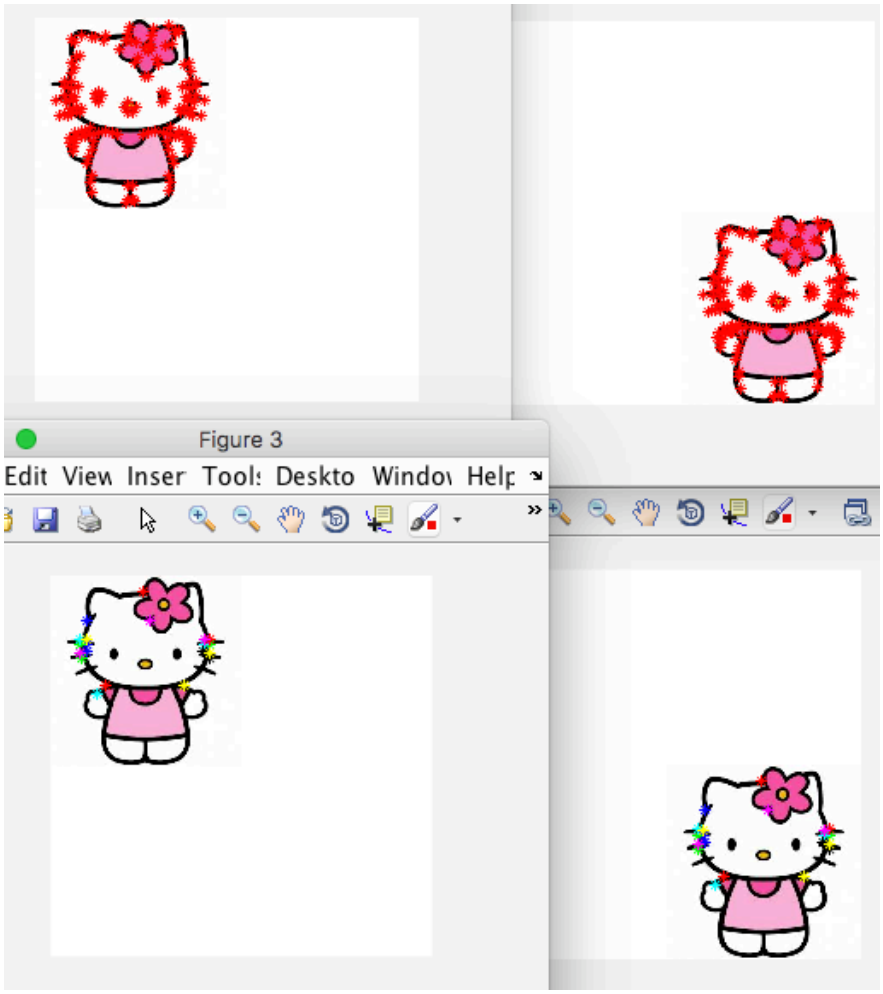
Part 2: Descriptor Matching

In this section, we'll implement simple descriptor matching. For interest point detection, now use matlab functions `corner()` or `detectHarrisFeatures()` as they are likely to be more robust than your own implementation. Examine the provided template **lab2part2.m**. The initial script creates two images to match by taking an input image and translating it. Your goal is: (1) to write the routines `extractDescr()` for extracting descriptors and (2) routine `matchDescrs()` for matching them. Assuming N keypoints have been detected, then:

- `extractDescrAppearance()` should take the Nx2 list of detected keypoint coordinates, and return an NxPatchSize matrix, where each row encodes the vector of pixels centred at each patch.
- `extractDescrGradient()` should take the Nx2 list of detected keypoint coordinates, and return an NxHistSize matrix representing the histogram of gradients within each patch.
    - Hints Commands like hist and histc can make this easier.
- `matchDescrs()` should take one descriptor matrix for each image, and return the Nx1 vector of matches. Each element of this vector says for the corresponding keypoint in image1, which is the corresponding keypoint in image2. No matches can be encoded by a negative number.
    - Hint: Checkout pdist2 between the descriptors (and visualise it as an image).

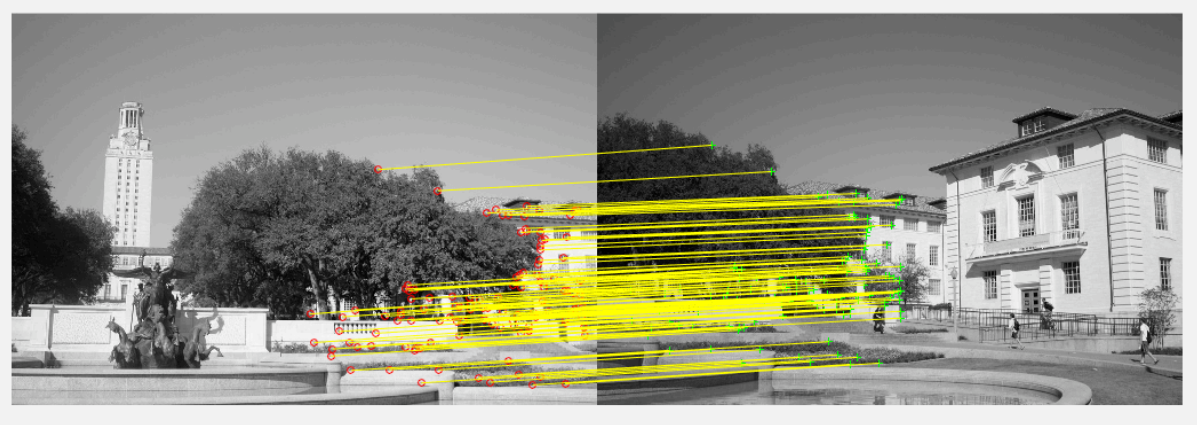
The result should look something like the below, where the first row shows the initial detected interest points and the second row shows the result of matching them. The matches are color coded, and you can see the corresponding points in each image match colors.

- Bonus [Moderate]: Rather than using the initial image ka.jpg twice, try to load up ka.jpg and kb.jpg as the first and second images. Try to implement a gradient-based descriptor and matching routine that is robust enough to match points between these two different Hello Kitty variants?



Part 3: Image Alignment with RANSAC

In this section, we'll use RANSAC to do robust estimation of an affine image transformation based on pairs of matched interest points. Examine the provided script **lab2part3.m**. You will see it loads two images and matches features between them.



The code also includes an example of computing the image alignment with least squares. But this may not be robust enough due to outliers. Your task is to write a RANSAC loop that **robustly** computes the affine transformation between the left and right image. As per the given example, it should return the 3x1 transformation M that takes homogeneous points X_1 in image 1 and transforms them to match their corresponding points X_2 in image 2. IE: $X_2=MX_1$. But to do this robustly with RANSAC, your routine should repeatedly sample different subsets of matched points to compute the transformation, and then score them by the number of inliers. Return the transformation with the highest inlier score.

You can then use `affine2d()` and `imwarp()` to warp one image so its aligned with the other.

image1

transformed image1

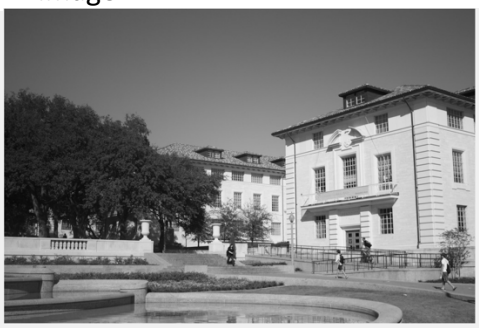


image2

- Bonus [Moderate]: Use the RANSAC loop to compute alignments, and average the two aligned images together to make a panorama.