Computer Vision CS543
Homework 2

Due Date: 27 Feburary 2017

Submit your pdf and zip files on Compass. Be sure to include the required outputs described below in your pdf, as well as any other equation or explanation indicated in the questions. Please keep your code in separate folders for each problem. In each folder, include a `‘runThis.m’` for demo. Thanks!

1 A feature tracker (50pt)

For this problem, you will track features from the image sequence `hotel.seq0.png` ... `hotel.seq50.png`. Since this is a two part problem, we have included precomputed intermediate results in the supplemental material in case you're unable to complete any portion.

Please also include pseudocode in your report. Furthermore, do not use existing keypoint detectors, trackers, or structure from motion code, such as found in OpenCV.

1.1 Keypoint Selection (15pt)

For the first frame, use the second moment matrix to locate strong corners to use as keypoints. These points will be tracked throughout the sequence.

You can either use the Harris criteria (1), *or* the Shi-Tomasi/Kanade-Tomasi criteria (2). Here M is the second moment matrix, λ_1, λ_2 are the eigenvalues of M, and τ is the threshold for selecting keypoints:

$$\det(M) - \alpha \cdot \operatorname{trace}(M)^2 \geq \tau \tag{1}$$

$$\min(\lambda_1, \lambda_2) \geq \tau \tag{2}$$

If using the Harris criteria, it is good to choose $\alpha \in [0.01, 0.06]$. Choose τ so that edges and noisy patches are ignored. Do local non-maxima suppression over a 5x5 window centered at each point. This should give several hundred good points to track.

Required output:

1. Display the first frame of the sequence overlaid with the detected keypoints. Ensure that they are clearly visible (try `plot(..., ‘g.’, ‘linewidth’,3)`).

Suggested Structure:

Write this as a function such as `[keyXs, keyYs] = getKeypoints(im, tau);` Be sure to smooth the gradients when constructing the second moment matrix.

Useful functions:

`imfilter.m`

References:

C. Harris and M. Stephens. *A Combined Corner and Edge Detector.* 1988

J. Shi and C. Tomasi. *Good Features to Track.* 1993

1.2 Tracking (35pt)

Apply the Kanade-Lucas-Tomasi tracking procedure to track the keypoints found in part 1.1. For each keypoint k, compute the expected translation from $(x, y) \rightarrow (x', y')$:

$$I(x', y', t+1) = I(x, y, t) \tag{3}$$

This can be done by iteratively applying (4): Given the i^{th} estimate (x'_i, y'_i), we want to update our estimate $(x'_{i+1}, y'_{i+1}) = (x'_i, y'_i) + (u, v)$. Here, W is a 15x15 pixel window surrounding the keypoint, which is located at (x, y) in frame t. I_x, I_y are the x, y gradients of image $I(x, y, t)$, computed at each element of W at time t. $I_t = I(x', y', t+1) - I(x, y, t)$ is the "temporal" gradient. A fixed, small number of iterations should be sufficient.

$$(x'_0, y'_0) = (x, y)$$
$$I_t \approx I(x'_i, y'_i, t+1) - I(x, y, t)$$

$$\begin{bmatrix} \sum_W I_x I_x & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_W I_x I_t \\ \sum_W I_y I_t \end{bmatrix} \tag{4}$$

$$(x'_{i+1}, y'_{i+1}) = (x'_i, y'_i) + (u, v)$$

This should be applied iteratively, that is, begin with $(x'_0, y'_0)^T = (x, y)^T$, which is needed to compute I_t. Use this I_t to estimate $(u, v)^T$, which can in turn be used to compute $(x'_1, y'_1) = (x'_0, y'_0) + (u, v)$, and so on. Note that $(x', y')^T$ (and $(x, y)^T$) need not be integer, so you will need to interpolate $I(x', y', t+1)$ ($I_x, I_y, ...,$ etc.) at these non-integer values.

Some keypoints will move out of the image frame over the course of the sequence. Discard any track if the predicted translation falls outside the image frame.

Required Output:

1. For 20 random keypoints, draw the 2D path over the sequence of frames. That is, plot the progression of image coordinates for each of the 20 keypoints. Plot each of the paths on the same figure, overlaid on the first frame of the sequence.

2. On top of the first frame, plot the points which have moved out of frame at some point along the sequence.

Useful functions:

`interp2` - For computing I_x, I_y and $I(x', y', t+1)$ when x, y, u, v are not integers.

`meshgrid` - For computing the indices for `interp2`

Suggested Structure:

`[newXs newYs] = predictTranslationAll(startXs, startYs, im0, im1);` - Compute new X,Y locations for all starting locations. Precompute gradients Ix,Iy here, then compute translation for each keypoint independently:

`[newX newY] = predictTranslation(startX, startY, Ix, Iy, im0, im1);` - For a single X,Y location, use the gradients `Ix, Iy`, and images `im0, im1` to compute the new location. Here it may be necessary to interpolate `Ix,Iy,im0,im1` if the corresponding locations are not integer.

References:

Carlo Tomasi and Takeo Kanade. *Detection and Tracking of Point Features.* 1992

2 Shape alignment (30pt)

Write a function that aligns two sets of points:

```
T = align_shape(im1, im2)
```

where T is a transformation that maps non-zero points in $im1$ to non-zero points in $im2$ (see the zip for the images). You may choose the alignment algorithm and the type of (global) transformation (e.g., rigid Euclidean, affine, perspective).

Test your function by mapping: object2 to object2t, object1, object3. For example,

```
T_2t = align_shape(imread('object2.png')>0, imread('object2t.png')>0);
```

should align the points in 'object2' to the points in 'object2t', display all three sets of points (original and aligned) and compute the alignment error. We've included functions `evalAlignment` and `displayAlignment` to help with evaluation and display.

Required output:

1. A brief explanation of your algorithm, initialization, and model of the transformation
2. For each result:

   The alignment display;

   The final error;

   The runtime (e.g., use 'tic', 'toc').

Grading:

Your algorithm can align object2 to object2t. (10pt)

Your algorithm can align object2 to object1 and object3. (10pt)

Your writeup. (10pt)

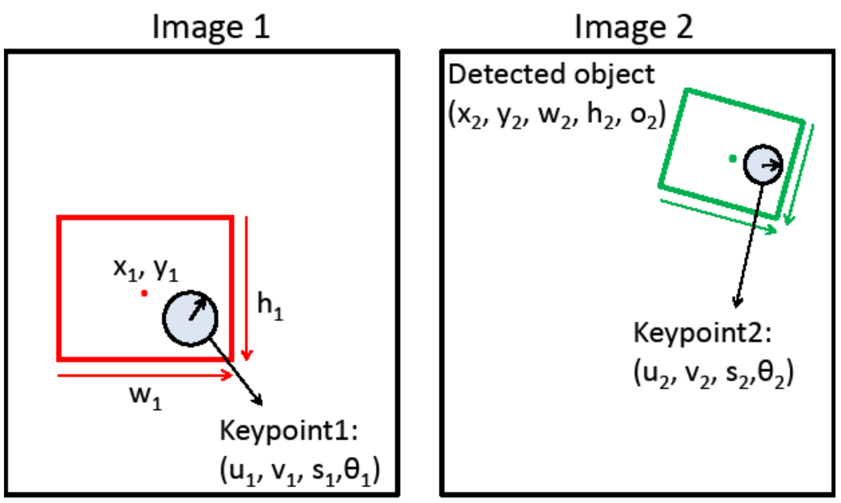

Figure 1: Object instance detection from keypoint matching

3 Object instance recognition (20pt)

For this problem, you will explore Lowe-style object instance recognition problem. No code is required.

3.1 Keypoint matching (5 pt)

Given a keypoint descriptor g from one image and a set of keypoint descriptors $f_1 \dots f_n$ from a second image, write the algorithm and equations to determine which keypoint in $f_1 \dots f_n$ (if any) matches g.

3.2 Object alignment (15 pt)

Suppose that you have matched a keypoint in the object region to a keypoint in a second image (see Figure 1 above). Given the object bounding box center (x, y), width, and height (x_1, y_1, w_1, h_1) and the position, scale, and orientation of each keypoint $(u_1, v_1, s_1, \theta_1; u_2, v_2, s_2, \theta_2)$, show how to compute the predicted center position, width, height, and relative orientation of the object in the second image.