Computer Vision **CS 543 / ECE 549**

Homework 3
Due Mar 20, 2017

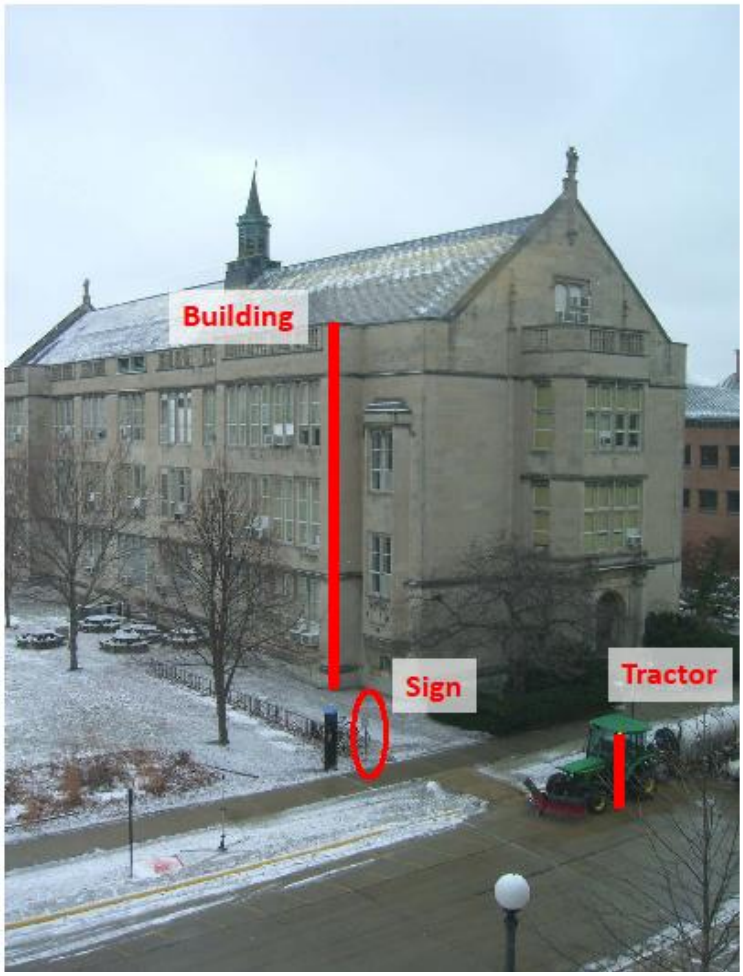
Answer the following questions and explain solutions. Numbers in parentheses give maximum credit value. You can discuss in small groups, but turn in individual solutions and indicate collaborators. Do not use code from the Internet or high-level functions from within Matlab (functions for key parts of the algorithm, such as RANSAC or structure-from-motion) unless specific permission is given.

Submit your pdf and code zip files on Compass. Please keep your code in separate folders for each problem. In each folder, include a 'runThis.m' for demo. Thanks!

1. **Single-View Metrology (40 pts)**
   A. For the Kyoto Street image, shown above, estimate the positions (in the image plane) of the three major orthogonal vanishing points (VPs), corresponding to the building orientations. Use at least three manually selected lines to solve for each vanishing point. The included code getVanishingPoint_shell provides an interface for selecting and drawing the lines, but the code for computing the vanishing point needs to be inserted.
      - Plot the VPs and the lines used to estimate them on the image plane. (3 pts)
      - Specify the VPs (u,v). (2 pts)
      - Plot the ground horizon line and specify its parameters in the form $au + bv + c = 0$. Normalize the parameters so that: $a^2 + b^2 = 1$. (5 pts)

   B. Use the fact that the vanishing points are in orthogonal directions to estimate the camera focal length (f) and optical center (u_0, v_0). Show all work. (10 pts)
   C. Show how to compute the camera's rotation matrix when provided with vanishing points in the X, Y, and Z directions. (5 pts)

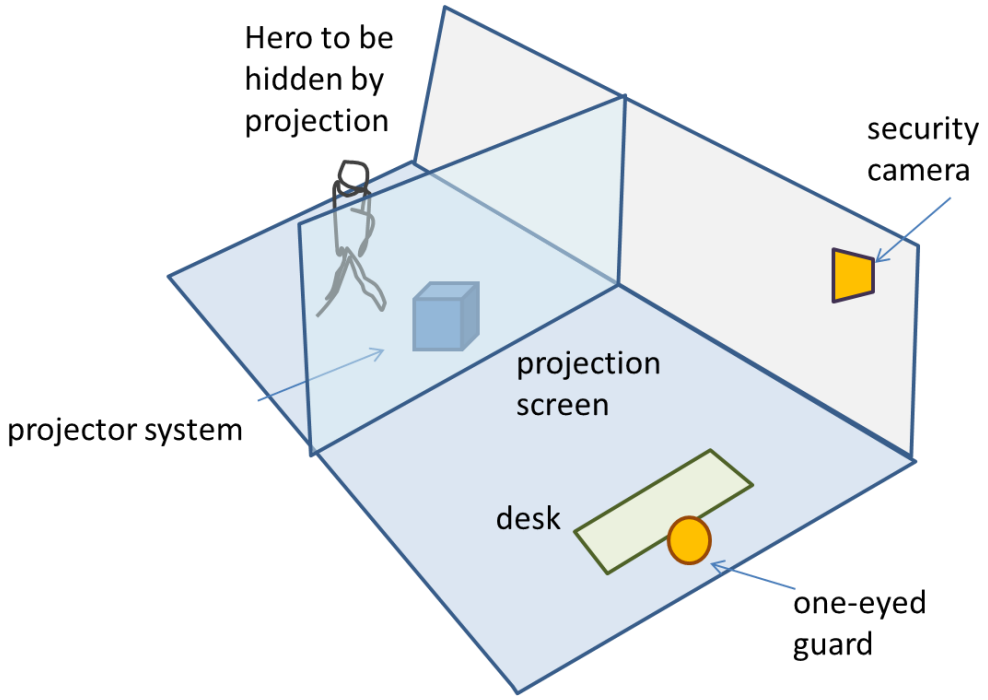
Now, compute the rotation matrix for this image, setting the vertical vanishing point as the Y-direction, the right-most vanishing point as the X-direction, and the left-most vanishing point as the Z-direction. (5 pts)



D. The above photo is of the University High building, taken from the third floor in Siebel Center facing south. Estimate the horizon and draw/plot it on the image. Assume that the sign is 1.65m. Estimate the heights of the tractor, the building, and the camera (in meters). This can be done with powerpoint, paper and a ruler, or Matlab.
   - Turn in an illustration that shows the horizon line, and the lines and measurements used to estimate the heights of the building, tractor, and camera. (5 pts)
   - Report the estimated heights of the building, tractor, and camera. (5 pts)

E. Extra credit (up to 20 pts) – see description on the last page
   - Implement an automatic vanishing point detection algorithm using RANSAC (12 pts)
   - Implement camera calibration from the three detected vanishing points (8 pts)

2. Mission Possible? (10 pts)



In "Mission Impossible: Ghost Protocol", the heroes need to work in a corridor within full view of a security guard. Their solution: erect a back-lit projection screen and project an empty hallway onto it. Assume they have ultra-super-duper projector and computer vision technology – they can update the projected image in real-time and detect/track objects. Consider the following scenarios:

a) A security camera is looking down the hallway at the screen. The camera can freely rotate but cannot otherwise move. Is it possible to put an image on the projection screen, such that the screen is undetectable for someone monitoring the camera? If not, why not? If so, what information is required? (4 pts)

b) A security guard is sitting behind his desk looking down the hallway. Recently, while inspecting a pencil, the guard poked his eye, and now he has a patch covering that eye. But he can still move around and has one good eye. Is it possible to fool the security guard? If not, why not? If so, what information is required? (3 pts)

c) Is it possible to fool both the security camera and the one-eyed security man at the same time? If not, why not? If so, what information is required? (3 pts)

Note: these should be short answers. One or two sentences is fine.

3. Epipolar Geometry (20 pts)

For the given pair of chapel images (above):

a) Use the set of matched points provided to you in prob3.mat to estimate the fundamental matrix *F* automatically using RANSAC and the normalized 8-point algorithm. (15 pts)
   - Indicate what test you used for deciding inlier vs. outlier.
   - Display F after normalizing to unit length.
   - Plot the outliers with green dots on top of the first image (plot(x, y, 'g.')).

b) Choose 7 sets of matching points that are well separated (can be randomly chosen). Plot the corresponding epipolar lines ('g') and the points (with 'r+') on each image. Show the two images (with plotted points and lines) next to each other. (5 pts)

The file prob3.mat has detected Harris corners row-column positions in variables *r1 c1* for the first image; variables *r2 c2* for the second image; and the corresponding matched pairs in the variable *matches*. For part (a), you may want to use the provided function `plotmatches.m` (e.g. `plotmatches(im1,im2,[c1 r1]',[c2 r2]',matches')`) to plot matches for debugging.

Tips:

- I strongly suggest manually selecting a set of good matches and getting it working with those matches first. Then, you can get it working using RANSAC. If you get it working with manual selection but not with RANSAC, you will get half credit for part (a).
- At the very least, your epipolar lines should pass very near (e.g., within 1 pixel) your plotted points, but the solution might vary slightly from run to run.
- Consider your outlier criterion carefully.

4. Affine Structure from Motion (30%)

This problem continues the interest point detection and tracking problem from HW2. Now, you will recover a 3D pointcloud from the image sequence `hotel.seq0.png` ... `hotel.seq50.png`. You are encouraged to use your results from HW2, but in case you were not able to complete it, we have also included pre-computed intermediate results in the supplemental material. **As before, you must submit your code. Please also include pseudocode in your report.** Furthermore, do not use existing structure from motion code, such as found in OpenCV.

Use the discovered tracks found in HW2 P1.2 as input for the affine structure from motion procedure described in *Shape and Motion from Image Streams under Orthography: a Factorization Method* 1992 by Tomasi and Kanade (see section 3.4 for an overview of the algorithm).

Note that there should be a sufficient number of tracks that persist throughout the sequence to perform the factorization on a dense matrix. There is no need to fill in missing data for this problem.

To eliminate the affine ambiguity (i.e. apply the metric constraints) by discovering QQ^T (eq. 16 of Tomasi and Kanade), let $L = QQ^T$ and solve for L using least squares. Note that $\hat{\mathbf{i}}_f, \hat{\mathbf{j}}_f$ refer to rows of $\hat{R}$. Finally, compute the Cholesky decomposition of $L = QQ^T$ to recover the appropriate Q.

Required Output:

1. Plot the predicted 3D locations of the tracked points for 3 different viewpoints. Choose the viewpoints so that the 3D structure is clearly visible.
2. Plot the predicted 3D path of the cameras. The camera position for each frame is given by the cross product $\hat{\mathbf{k}}_f = \hat{\mathbf{i}}_f \times \hat{\mathbf{j}}_f$. For consistent results, normalize all $\hat{\mathbf{k}}_f$ to be the same length (i.e. unit vectors). Give 3 plots, one for each dimension of $\hat{\mathbf{k}}_f$.

Useful functions:

`svd`, `chol`

`plot3` - for plotting 3D points

References:

Tomasi and Kanade. *Shape and Motion from Image Streams under Orthography: a Factorization Method.* 1992

Extra Problems (optional)

Any of the following extensions can be done for extra credit. Points from up to two problems (up to 30 pts) will be awarded.

Shi-Tomasi Affine Verification (15 pts) Use the Shi-Tomasi approach to infer the affine transformation between each keypoint from the first frame and frames $F = [10, 20, 30, 40, 50]$. Plot the points whose appearance drifts too far over time. Also plot the unrectified and affine-rectified patches for several points for each frame in F.

Missing Track Completion (15 pts) Some keypoints will fall out of frame, or come into frame throughout the sequence. Use the matrix factorization to fill in the missing data and visualize the predicted positions of points that aren't visible in a particular frame.

Optical Flow (15 pts) Implement the optical flow approach from lecture to estimate the translation at every point. Use convolution to compute sums over W efficiently.

Coarse to Fine Tracking (15 pts) Implement the coarse-to-fine tracking procedure. To test the accuracy on large translations, tracks the keypoints by only using frames $F = [0, 10, 20, 30, 4050]$. Compare this to if you run original tracker over frames F.

Vanishing Point Extra Credit (optional, worth 20 points total)

1) Use RANSAC to automatically detect three orthogonal vanishing points (up to 12 pts)

```
[VP, lineLabel] = vpDetectionFromLines(lines)
```

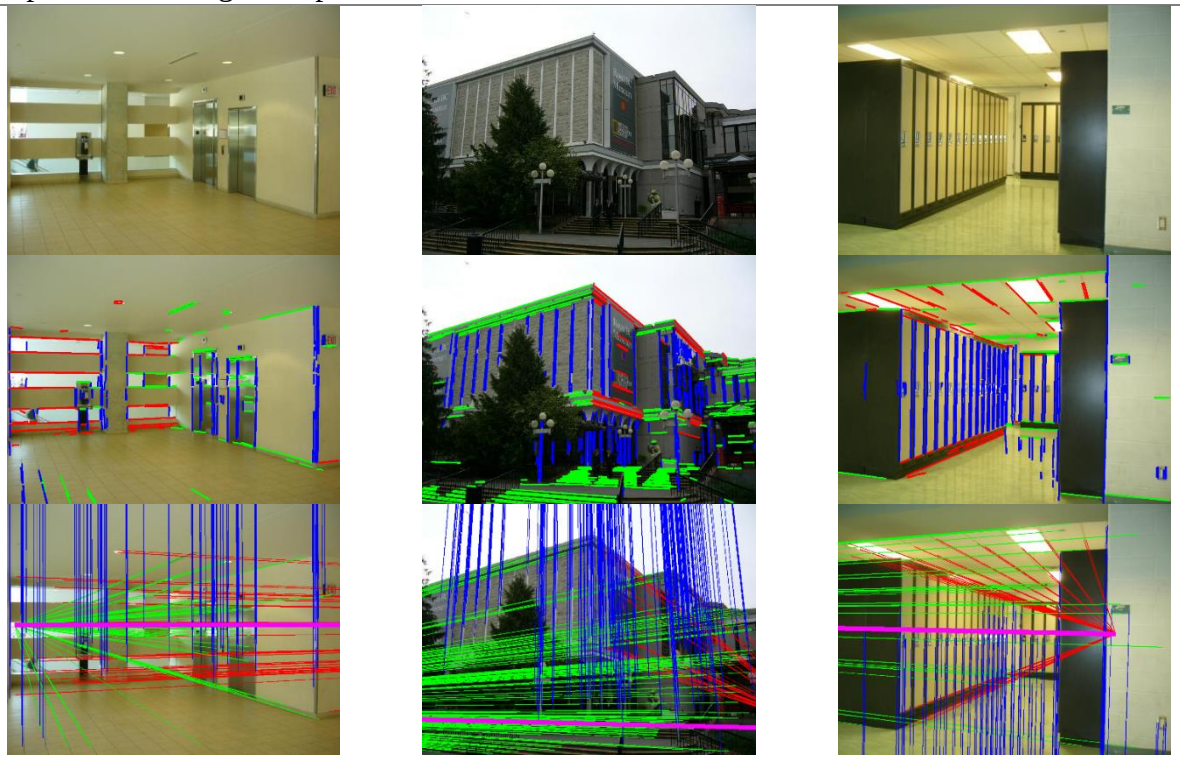
This function takes a set of line segments and uses RANSAC to detect three vanishing points that correspond to three orthogonal directions in the scene. The main steps are (1) randomly select a set of edge pairs, compute their intersections as vanishing points hypothesis; (2) check the consistency of these vanishing points hypothesis; (3) clustering the line segments into different groups corresponding to vanishing points.

For (3), you may use the clustering code we included.
`lineLabel = clusterLineSeg(PrefMat);`
This function takes a binary matrix PrefMat of size [NumLines x NumVP] where PrefMat(i, j) = 1 indicates line segment `i` is compatible with vanishing point `j` (i.e., `i-th` line is an inlier of `j-th` vp).

Input:
- lines: a matrix of size [NumLines x 5] where each row represents a line segment with (x1, y1, x2, y2, lineLength)

Output:
- VP: [2 x 3] each column corresponds to a vanishing point in the order of X, Y, Z
- lineLabel: [NumLine x 3] each column is a logical vector indicating which line segments correspond to the vanishing point.

Your algorithm should be able to get results similar to the following. First row, original image. Second row, the line segments grouped into three orthogonal vanishing points. Third row, the vanishing points and the position of the ground plane.



2) Use the detected vanishing points to calibrate the internal parameters (focal length f and optical centers u0, v0) (up to 8 pts)

```
[f, u0, v0, R] = camCalibFromVP(VP);
```

This function performs camera calibration from the detected VP. You will use the algorithm you derived from sub-problem A, B, and C.

Input:

- VP: [2 x 3] vanishing points

Output:

- f: focal length (in pixels)
- u0, v0: optical center
- R: a [3 x 3] rotation matrix

You can use the code `runThis.m` to evaluate the accuracy of your algorithm over the 50 images taken from the YorkUrbanDatabase. Four evaluation metrics against groundtruth data are included.

Intrinsic parameters

1) Focal length (in mm)
2) Distance to optical center (in pixels)

Extrinsic parameters

3) Angular difference between the estimated zenith point (the third vanishing point) and the groundtruth
4) Horizon line estimation error

Write-up: Include your code with your electronic submission. In your write-up, include:

- Description of any design choices and parameters, e.g., criterions of the vanishing point to line segment fitting. Threshold for detecting inlier points.
- For part (a), choose 5 example images, show input images, grouped line segment, and the vanishing points with estimated horizon line. You may use the `tight_subplot.m` to display the figures in a 3 x 5 (or 5 x 3) array.
- For part (b), report the quantitative evaluation of the four metrics on camera calibration. You may report the median of the errors.

Related papers:

- Tardif J.-P., Non-iterative Approach for Fast and Accurate Vanishing Point Detection, 12th IEEE International Conference on Computer Vision, Kyoto, Japan, September 27 - October 4, 2009
- B. Caprile, V. Torre, Using Vanishing Points for Camera Calibration, International Journal of Computer Vision, 1990