16-822: Geometry-based Methods in Vision (F16) Released: Sep-19th. Due: Oct. 3rd.

Homework 1

Lecturer: Martial Hebert *TA: Nick Rhinehart*

1 Theory (45 Points)

1.1 H for mirror reflection image (8 Points)

Suppose that a scene is a plane and that it is projected to an image assuming the normal pinhole model. Suppose that the points on the plane are related by a symmetry about a line L (for example, imagine a painting in which the "left" and "right" half are identical).

- Show that there exists a planar homography H in the image such that any point p on the image matches other point $p' = Hp$ on the image plane.
- Show that $H^2 = I$.
- Show that H has an eigenvector which is the vanishing point of L.

1.2 Fun with pencils of lines (12 Points)

- A line in the plane is represented by a 3-vector $l = [u\ v\ w]^T$. What is the slope s of the line? (Ignoring the one case of infinite slope)
- Consider a pencil of lines in the plane. Explain why any line in the pencil can be represented canonically by a linear combination of two distinct lines represented by two 3-vectors l_1 and l_2. (Simple answer, just use an argument based on the dimension of the subspaces involved.)
- Now consider two pencils of lines and a homography H between the two pencils. What is the dimension of the matrix representing H? (again, no equations; 1-sentence answer based of the dimensions of the spaces involved) Be careful that we are asking about the homography from one set of lines to another, not about a planar homography.
- Show that if l' is the transformed of l by the line homography H, then the slopes are related by a function of the form (ignoring the infinities):

$$s' = \frac{as + b}{cs + d}$$

- Conversely, assume that a transformation between the two pencils is of that form, state what condition the coefficients a, b, c, d must satisfy for the transformation to be a line homography? What is the matrix representing that homography?

1.3 Rodrigues's rotation formula (and some motion) (25 Points)

- Rodrigues's rotation formula is a convenient way of expressing rotations of a vector $\mathbf{p}$ as a function of the rotation angle θ and the axis of rotation $\mathbf{w}$:

$$\mathbf{Rp} = \mathbf{p} + \sin(\theta)\mathbf{w} \times \mathbf{p} + (1 - \cos(\theta))\mathbf{w} \times (\mathbf{w} \times \mathbf{p})$$

  Write Rodrigues's formula in matrix form:

$$\mathbf{Rp} = \ldots$$

- Suppose now that θ is very small (e.g., the small rotation between two consecutive frames in a video sequence). What would be an approximation of $\mathbf{RP}$, denoting $\omega = \theta\mathbf{w}$?
- Consider now 2 cameras separated by a small rotation R and a translation t. Assuming $K = I$ (calibrated cameras) and using the approximation above, what is the essential matrix E?
- We know that the epipole in one image is t. What is the epipole in the other image if the axis of rotation w is parallel to t?
- Consider now a pixel of coordinates (u, v) in one image, projection of a point (X, Y, Z). The corresponding point in the other image is (u', v'). Assuming again small motion we have: $u' = u + du$ and $v' = v + dv$ (we can think of du and dv as the apparent motion of the point corresponding to (u, v) in the image.)

  Express du and dv as function of u, v, Ω, t, Z (remember to write first that (u, v) are the projection in the image of X, Y, Z from the trivial projection matrix $[I\ 0]$.)
- Show that if P is known to be on a plane (n, d), then (du, dv) does not depend on Z, the "depth" of the point.
- In that case, also show that (du, dv) are quadratic expressions in u, v.
- The reasoning above used derivatives (du and dv). Another angle on the problem is to compute the homography H induced by a plane (n, d) assuming small rotation Ω and a translation t. Write an expression for H using the small-angle rotation expression above.
- Given this expression of H, which point remains invariant under H if Ω is orthogonal to n? (e.g., if the plane is the ground plane and the cameras are on a robot moving on the ground plane).

2 Having fun by implementing what we have learnt! (55 Points)

In this part of the homework, we will try to generate metrically correct view of a warped image. You can use **any method you prefer** as long as it conforms to the descriptions below:

What you have to do:

- Given an example image, write a function that returns metrically rectified image as well as the H used to perform rectification. Follow this function definition:

```
function [rectI, H] = rectifyImage(filename, debug)
```

If `debug=1`, you should provide whatever manual input (annotation) you require (e.g., marking parallel or perpendicular lines, planes etc.). `MATLAB's ginput` might come in handy.

```
[x y] = ginput(4); pt1 = [x(1) y(1) 1]'; pt2 = [x(2) y(2) 1]';
```

Save all the information you entered, so that it can be evaluated. You can use any format for saving, as long as it is consistent with your load commands.
If `debug=0`, you should load whatever information you manually entered and return `[rectI, H]`. Follow this convention for saving/loading:

```
[d, fname] = fileparts(filename);
data = load(fullfile(strrep(d, '/images', '/data'), [fname '.mat']));
```

where `fname.mat` is the file that stores the annotations (above code assumes images and annotated data are in `'anypath/images/'` and `'anypath/data/'` directories respectively.)

- Capture (or find) **3** your own images. Ideally, capture/find images that have substantial perspective distortion. (See figure 1)
- Run your code on at least **7 out of the given 13** images as well as on the images you captured/found. Put your images in the sub-folder `./images/` and the `mat` files with annotations in the sub-folder `./data/`.
- See what you need to submit in the end.

(a) Good input! (b) Bad input...

Figure 1: The good and bad inputs.

What you can use:

- You can use **any method you prefer**. You might get ideas from material taught in class (16822) and Computer Vision (16720). Also, material from Hartley&Zisserman, and Faugeras might be helpful. Any variations on the ideas for rectification mentioned in stated materials, e.g., there are other combinations of parallel/perpendicular points and lines, as well as methods using distances which Martial mentioned in class are fine.
- You can also use advanced material from any papers/research/reports that you find, as long as you understand them.
- Code to manually select points or planar regions in the image. (e.g., MATLABs ginput)
- `MATLAB`'s functions or your code from 16720 used for warping. This might be helpful as well:

```
function rectI = applyH(img, H)
  tform = maketform('projective',H');
  %H or H' depending on your convention
  [boxx boxy]=tformfwd(tform, [1 1 size(img,2) size(img,2)], [1 ...
      size(img,1) 1 size(img,1)]);
  minx=min(boxx); maxx=max(boxx);
  miny=min(boxy); maxy=max(boxy);
  rectI =imtransform(img,tform,'XData',[minx maxx],'YData',[miny ...
      maxy],'Size',[size(img,1),round(size(img,1)*(maxx-minx)/(maxy-miny))]);
  % imshow(rectI);
end
```

- Non-linear distortion compensation code if the images you choose are too distorted. (Try not to choose such images).

What you *can not* do:

- Download code, except for the distortion and manual input (see above.)
- Use any measurement in the scene provided apriori (the only information available as input is the image itself, and whatever you annotate).
- Use any calibration parameters for the camera.
- Make any assumption about the camera parameters, except that it is a perspective camera.

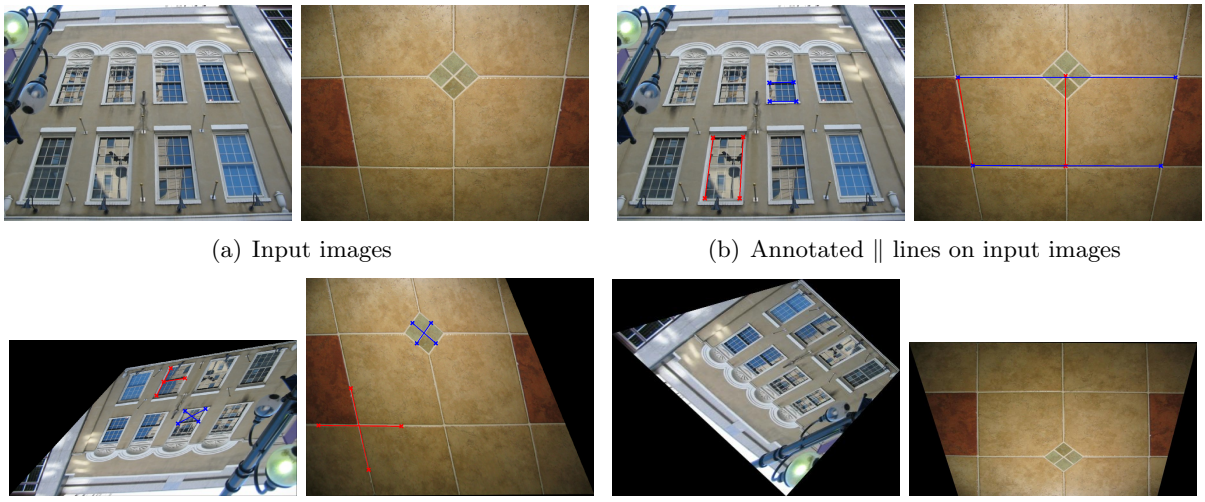
What you have to submit:

- Input images
- Output images: Rectified image and intermediate images color coded annotation that you used. Display of the lines that were used, including the vanishing line if you use it in your method. (See figure 2)
- Angles (more precisely, the cosines) of at least 3 pairs of perpendicular lines before and after rectification (provide annotations in the mat file as well). This will show how far from 0 the cosines are in the original image and how close to correct the final metric rectification is. (See figure 3 and table 1).
- Description of your implementation (i.e., the algorithm you followed with relevant equations, what were the annotations used) and (most importantly) what problems were encountered (e.g., numerical issues, etc.).
- **CODE**.

Before	After
-0.15073	-0.00624
0.10344	0.004659
0.159543	-0.001270

Table 1: An example of cosine θ between lines.

Tips:



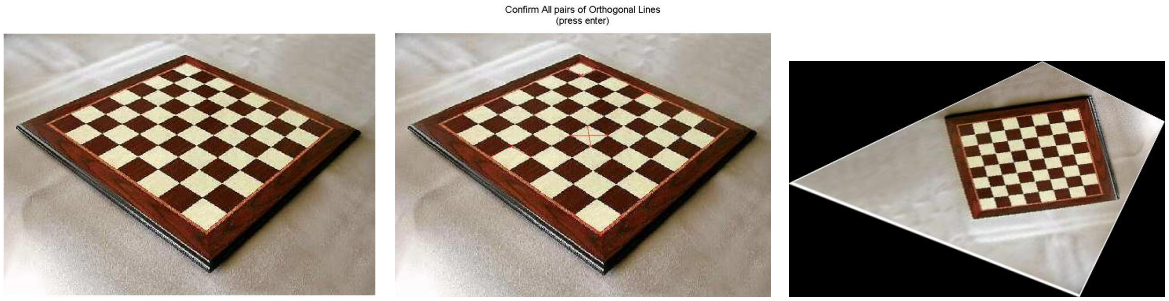
(a) Input images

(b) Annotated $\parallel$ lines on input images

(c) Annotated $\perp$ lines on affine rectified images

(d) Rectified image

Figure 2: An example of Output Images to be submitted for the rectification algorithm



(a) Input image

(b) Annotated lines for which angles are computed

(c) Rectified Image

Figure 3: An example of Output Images to submit for angle between lines.

- Make sure to keep "sanity checks" in your code while you are debugging, by checking an equation that you know should be true. (e.g., $l^T C m = 0$). Keep such code commented when you submit, so that I can see and appreciate it!
- Nomalize coordinates of points before doing anything.
- Remember the transformations that you are estimating are up to a scale.
- Be careful about all the things Martial mentioned in class!
- Have fun!!