16-822: Geometry-based Methods in Vision (F17) Released: Oct-04. Due Oct-25

Homework 2

Lecturer: Martial Hebert *TA: Aayush Bansal*

1 Theory (30 + 15 Points)

1.1 Ambiguities in 2-views (15 Points)

Here, we explore cases in which there are ambiguities in recovering scene/camera geometry from multiple views. We show this for the 2-view case, but the reasoning is similar in other problems. Consider the problem of estimating the fundamental matrix from a set of 8 correspondences $p_i, p'_i, i = 1 \dots 8$. We denote the image coordinates by u and v. Writing the entries of F as F_{ij}, the constraint for each correspondence is: $p'^T F p = 0$.

1. Setting $F_{33} = 1$ (because F is defined up to scale), show that this is equivalent to the set of 8 linear equations: $Af = -1_8$, where 1_8 is the 8-vector of constant 1, $f = [F_{11} F_{12} \dots F_{32}]^T$, and A is the 8×8 matrix such that row i of A is: $[u'_i u_i \; u'_i v_i \; u'_i \; v'_i u_i \; v'_i v_i \; v'_i \; u_i \; v_i]$.

2. The system is degenerate if A is singular. Using the fact that a matrix is singular iff there exists a non-trivial linear combination of its columns equal to zero, show that there exists a 3×3 matrix Q such that, for every correspondence i:

$$p'^T_i Q p_i = 0$$

3. Conclude that the eight points in space P_i corresponding to the correspondences p_i, p'_i must lie on a quadric surface in space.

4. Show that the optical centers C and C' of the two cameras lie on this quadric.

This shows that the 2-view recovery problem degenerates if the 8 points and the camera centers lie on a quadric.

1.2 Plane+Parallax (15 Points)

Consider two images and a plane in space inducing a homography H between two images (denoted by 1 and 2). In class, we saw that for any correspondence between two points p_1 and p_2, the points p_2, Hp_1, and e' are aligned and there exists a scalar "projective" depth relating the two. Now we prove these more formally.

1. Show that, for any correspondence between two points p_1 and p_2, the points p_2, Hp_1, and e' (the epipole in the second image) are aligned.

2. Show that for any correspondence, there exists a scalar r (the "projective" depth) such that: $p_2 = Hp_1 + re'$.

3. Assume that the cameras are calibrated (you can then assume $K = I$; this is not essential but it simplifies the math). Let us assume that we create a new image (denoted by 3) by applying the homography H to the first image. In other words, a pixel at position p_1 in the old image is related to a pixel at position p_3 in the new image by $p_3 = Hp_1$. Show that images 2 and 3 are related by a pure translation. (Several ways of showing this. One possible approach is to manipulate the various equation to derive the epipolar relation $p_2^T E p_3 = 0$ and conclude by inspection that it is the epipolar geometry of a translation.)

4. What is the translation vector between images 1 and 3[1]?

1.3 Extra Credits: Calibration of a moving set of cameras (15 Points)

Let us assume that we have a set of (uncalibrated) cameras on a robot. We can generate a *projective* reconstruction of the world from any position of the robot. We want to show that it is possible to generate a metric reconstruction (which means auto-calibration of the cameras) from two positions of the robot. In fact, we show here only that we can recover the plane at infinity from two robot positions; earlier results from class show how to complete the auto-calibration once the plane at infinity is known.

We denote by P the coordinates of a point in the projective reconstruction at the first position of the robot, and by P' the coordinates at the second position. We denote by H the 4×4 projective transformation between the two reconstructions, i.e., for any point: $P' = HP$. We assume that H is known (e.g., by looking at corresponding features between the two robot positions). We denote by Q the (unknown) correction matrix used to convert the projective reconstructions to a Euclidean one: $P_o = QP$ and $P'_o = QP'$, where the subscript $_o$ indicates that the coordinates are expressed in the Euclidean reconstruction. Finally, we denote by H_o the (unknown) Euclidean transformation between the two reconstructions: $P'_0 = H_0 P_0$.

- Show that, if v is an eigenvector of H_o, then $Q^{-1}v$ is an eigenvector of H with the *same* eigenvalue.
- Show that, for any 4×4 Euclidean point transformation T (i.e. $p' = Tp$) the plane at infinity is an eigenvector of the corresponding plane transformation ($\pi' = T^* \pi$, for some T^*) with real eigenvalue (in fact it is the only one with a real eigenvalue, but you don't need to prove that).
- Conclude that the plane at infinity π_∞ is the eigenvector of H^{-T} with real eigenvalues. This shows that π_∞ can be computed given reconstructions at two different positions of the robot.

[1]This is actually a powerful property. It basically says that there is equivalence between planar homography and rotations and that knowledge of one planar motion in the image is sufficient to "remove" the rotational component; something that can be used in image stabilization, for example.

2 Let us do some implementation! (75 + 20 + 20 Points)

2.1 Vanishing Point Detection (35 Points)

In this part of the homework, we explore the estimation of vanishing points. Given your experience with first homework, we thought it is a good time to *get your hands dirty* with this notorious beast: the vanishing point detection! Rules are simple, given an image (satisfying manhattan world assumption), you have to detect three orthogonal vanishing points and corresponding line segments (or lines). You can use **any method you prefer** as long as it conforms to the descriptions below.

2.1.1 What can you use?

Only input is the image itself and the knowledge that it has at least three vanishing directions. Input images are provided in `./images/input/` folder. Choose based on these rules:

- One or more images from set 1_`XXX.jpg`.
- One or more images from either of these sets: 2_`XXX.jpg`, 3_`XXX.jpg` or `aX.jpg`.
- Two or more images from `P10XXXXX.jpg`.
- One or more images found/clicked by you.

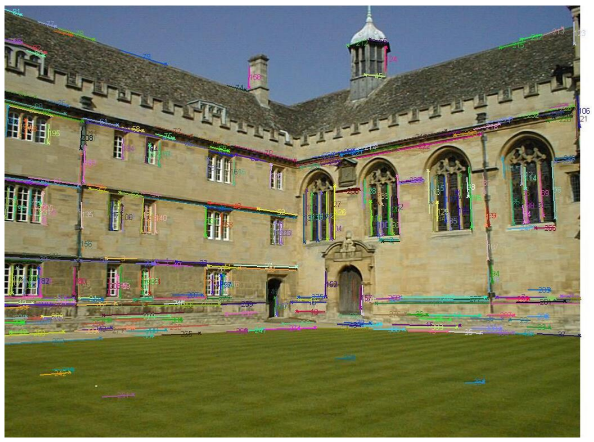
2.1.2 What can you *not* do?

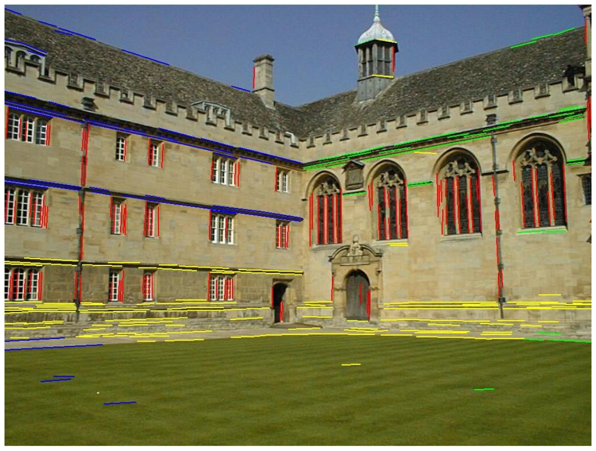
You can not use any manual annotations. Apart from manhattan world assumption, you can not assume anything else about the input images.

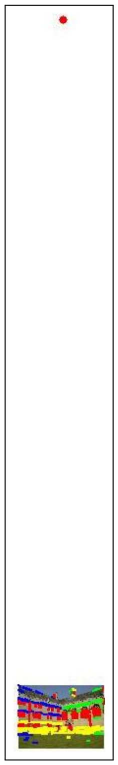
2.1.3 What you have to submit?

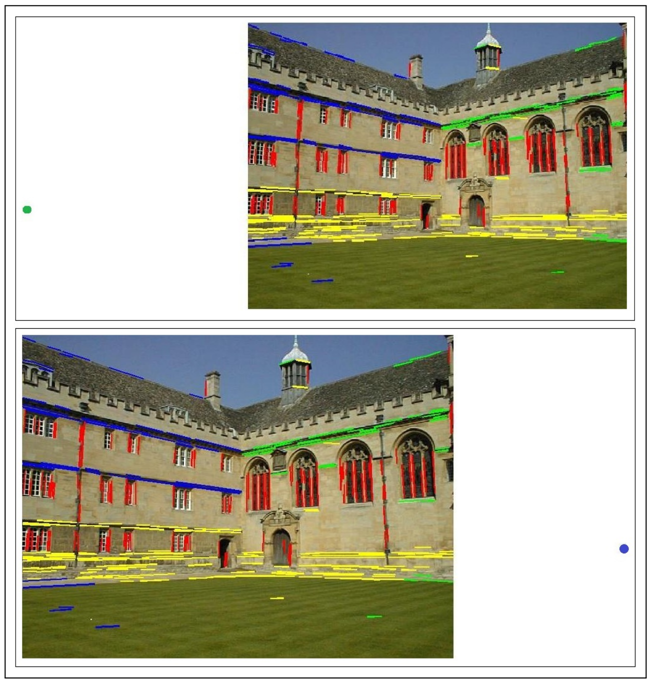
You should submit vanishing point detection results on at least 5 images (based on the rules mentioned in section 2.1.1.) The more images the better! Please follow instructions carefully. Include the following:

- Input images
- Output images: Images with vanishing points plotted (color coded) and their corresponding line segments (or lines). Submit one image with all three vanishing points plotted, and three images with individual vanishing points along with corresponding lines. (see figure 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.). You should properly describe your method (couple of lines do not work!).
- Make sure you include intermediate steps to describe your algorithm.
- Make sure you include *all* your result images in the report.
- CODE (including readme/script on how to use it)


(a) Line segments


(b) Color coded line assignments (red: vertical vp, green: left vp, blue: right vp, yellow: outliers)


(c) Vertical VP


(d) Left and Right VPs

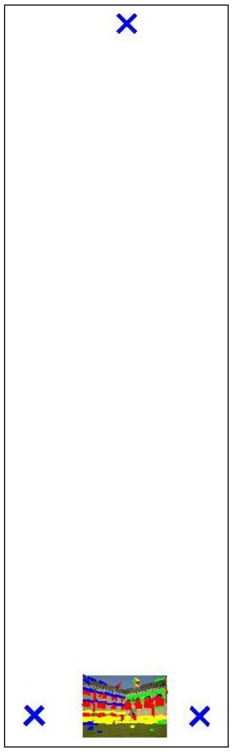

(e) All VPs

Figure 1: Few sample outputs. Feel free to have better visualizations, but include all these details.

2.1.4 Suggestions

- You can use any code for line segment detection. Three different codes to find long lines is given in `./code/lineCodes/` directory.
- We would recommend that you first try a naive approach (like covered in class) and then move to complicated approaches.
- We do not expect a very fancy generic vanishing point detection code, but a decent code that finds vanishing points on the given images.

2.2 Extra Credits: Vanishing Point Detection for Indoor Scenes (20 Points)

So far, we have seen images for outdoor environment alone. What if you use your algorithm from Section 2.1 for indoor scenes (like the ones shown in Figure 2)?

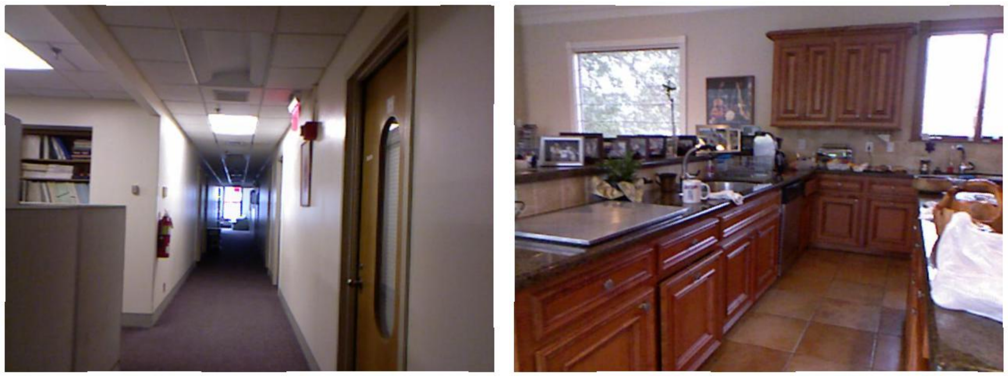


Figure 2: Example images of indoor scene from NYU RGBD dataset

2.2.1 What you have to submit?

We have provided a few images in `images/extracredits/` directory. You should submit vanishing point detection results on these images using your vanishing point detection algorithm from Section 2.1. The codes for a popular vanishing point detection approach[2] is available in `code/hedauvp/` directory. Run (`getVPHedauRaw(img)`) this provided code (Hedau et al.) on the given images. Now compare your approach with Hedau et al. Write a report that show similarity or differences between two approaches, and how you can improve your formulation or what could be done to improve the formulation of Hedau et al.

2.3 Homographies for Plane Detection and 3D Reconstruction (40 + 20 Points!)

In this part of the homework, we explore the use of homographies for 3D reconstruction. The motivation is that, in principle, using homographies will enable us to correctly reconstruct the planar surfaces and to enforce constraints such as orthogonality (if known). In contrast, using point matches, as the first step, makes it harder to correct the resulting noisy reconstruction later

[2] V. Hedau, D. Hoiem, D. Forsyth. *Recovering the spatial layout of cluttered rooms*; ICCV 2009

for planarity constraints. We explore these ideas on a few pairs of images of urban scenes, which contain lots of planar surfaces. You can use **any method you prefer** as long as it conforms to the descriptions below. The winning homework (**gets a bonus 10 points**) is the one that has the correct math and 3D result with the smallest number of such manual designation of geometric constraints.

2.3.1 What are you supposed to do?

Without pre-calibrating the camera, you need to do two things:
Step 1: Segment the images into planar regions and compute the homography corresponding to each plane. *Important*: Obviously, you can not find each and every plane in the image or assign each pixel (like sky/tree) to a real plane so we are happy if the major planes (e.g., ground, buildings) are correctly segmented. It is of course fine if your code is so good that it can approximate smaller regions (like people) by planes, but that is not required.
Step 2: Produce a "correct" display of the planes in 3D. *Be careful*: If you take the images yourself with a camera, make sure that you dont have radial distortion. If you download images from the Web, be aware that images are often cropped thus causing problems in estimating K sometimes (depending how you do it!).

2.3.2 Rules of the games!

For **Step 1**

- The plane segmentation should be automatic. You may use any technique you can think of: e.g. find homographies through RANSAC on feature points and go back to pixels to find regions; or reason about vanishing points to find the planes; or a combination of both.
- You need to implement your own math and algorithms to estimate the homographies and the regions but you may use existing code for any of the low-level operations like interest points, SIFT (or other descriptor), or line extraction, etc. (see section 2.5)

For **Step 2**

- Use whatever you want for the display as long as you display the points and the mapped regions in 3D and in 2D in different images. (see figure 3). If you prefer, you can also submit a video or vrml files.
- Of course, the regions are used only for display; the meat of the problem is to recover the metric projection matrices (i.e., K, E, etc.) from the homographies and perhaps the vanishing points (depending how you do this).
- Since we want a correct metric representation, you will need some metric thing in the scene, like angles. For that, you are allowed to designate manually things that are orthogonal in 3D (e.g., ground plane and a facade, or two sets of lines, etc.). Make sure you submit clearly annotated images in your report. (see section 2.4.4)

2.3.3 Input Images

Input images are provided in `./images/input/` folder. Choose based on these rules:

- One or more pairs of images from set 1_`XXX.jpg`.
- Two or more pairs of images from either of these sets: 2_`XXX.jpg`, 3_`XXX.jpg` or `aX.jpg`.
- One or more pairs of images found/clicked by you.

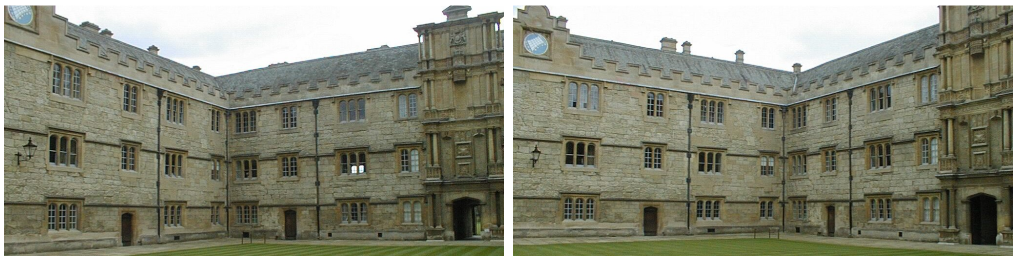
2.3.4 What you have to submit?

You should submit at least 4 pairs of images based on the rules mentioned in section 2.4.3. The more images the better! Please follow instructions carefully. Include the following:

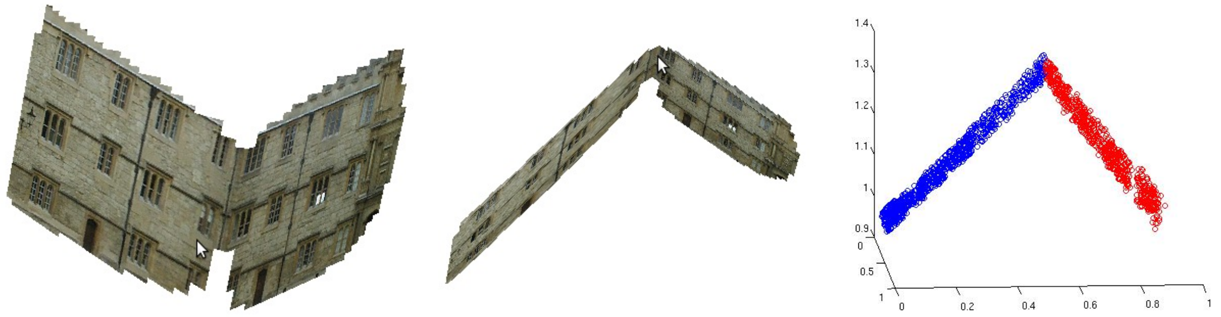
- Input images
- Output images (and videos, if you prefer): Segmented planes in the image (color coded) and their reconstruction in 3D. Display of the annotations that you used for your algorithm, including the vanishing points if you use it in your method. (see figure 3)
- Description of your implementation (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.). You should properly describe your method (couple of lines don't work!).
- Make sure you include *all intermediate steps with illustrations* to describe your algorithm.
- Make sure you include *all* your result images in the report.
- CODE (including readme/script on how to use it)

2.3.5 Suggestions!

- Note that, if you go back to the notes from the beginning of class, there are many different ways of doing this depending on which constraints you use on planes and lines and how you set up the problem. All of them are acceptable.
- Finally, as usual once you start working with pixels, Step 1 might be time consuming. It's perfectly fine if you start with Step 2 (i.e., use manually defined regions and corresponding homographies) to work out the math. And then apply the same using input from Step 1.
- For either steps, you may use existing code for estimating vanishing points from lines in the image, if you need that (**10 Bonus points!** for using your own implementation of vanishing points from section 2.1).
- For the actual reconstruction, you will get most points if you use 3D **planes** directly, i.e., draw the planes by deriving their equations from the homographies and them map
- If that saves time, it is fine to share the examples of image pairs (but not the code!)



(a) Example Inputs



(b) Example Reconstructions

Figure 3: Few different example Output Images to be submitted for the reconstruction algorithm

2.3.6 What can you *not* do?

- For the metric upgrade you should not assume 3D coordinates of any control points.
- You can not use Google Sketchup or equivalent.

2.4 Help & Tips:

- Three different codes to find long lines is given in `./code/lineCodes/` directory.
- 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., $l^T Cm = 0$). Keep such code commented when you submit, so that I can see and appreciate it!
- Normalize coordinates of points before doing anything.
- Remember the transformations that you are estimating are up to a scale.
- Keep in mind all the things Martial mentioned in class!
- **Start early...**
- Have fun!!