#5703: Computer Vision Homework 1

Due Date: Feb. 11

Instructions

A complete homework submission consists of two parts. A pdf file with answers to the theory questions and the Python files for the implementation portions of the homework. You should submit your completed homework by placing all your files in a directory named after your andrew id and upload it to your submission directory on Blackboard. The submission location is the same section where you downloaded your assignment. Do not include the handout materials in your submission.

All questions marked with a **Q** require a submission. For the implementation part of the homework please stick to the function headers described.

Final reminder: **Start early! Start early! Start early!**. Python is a powerful and flexible language. It means you can do thins swiftly by a few lines of Python code; it also means the program may go wrong without reporting any errors. Even for advanced Python users, it is often the case that a whole day can be spent debugging.

1 Homographies (25 points)

In class you have learnt that a homography is an invertible transformation between points on planes. The homography between planes is a powerful tool that finds use in a whole bunch of cool applications, some of which you will implement in this assignment. Let's begin by exploring the concept of a homography in a little more detail.

1.1 Definition and rank of a homography

Consider two points $\mathbf{p}$ and $\mathbf{q}$ on planes Π_1 and Π_2. Let us assume that the homography that maps point $\mathbf{p}$ to $\mathbf{q}$ is given by $\mathbf{S}$.

$$\mathbf{p} \equiv \mathbf{S}\mathbf{q} \tag{1}$$

Answer the following questions regarding $\mathbf{S}$:

Q1.1 What is the size (rows x columns) of the matrix $\mathbf{S}$? [1]

Q1.2 What is the rank of the matrix $\mathbf{S}$? Explain.

[1] Remember to use projective co-ordinates.

Q1.3 What is the minimum number of point correspondences ($\mathbf{p}$,$\mathbf{q}$) required to estimate $\mathbf{S}$? Why?

1.2 Homography between cameras

Q1.4 Suppose we have two cameras $\mathcal{C}_1$ and $\mathcal{C}_2$ [2] viewing a plane Π in 3D space. A 3D point P on the plane Π is projected into the cameras resulting in 2D points $\mathbf{u} \equiv (x_1, y_1, 1)$ and $\mathbf{v} \equiv (x_2, y_2, 1)$. Prove that there exists a homography between the points $\mathbf{u}$ and $\mathbf{v}$.

2 Panoramas (40 points)

We can also use homographies to create a panorama image from multiple views of the same scene. This is possible for example when there is no camera translation between the views (e.g., only rotation about the camera center). In this case, corresponding points from two views of the same scene can be related by a homography[3]:

$$\mathbf{p}_1^i \equiv \mathbf{H}\mathbf{p}_2^i, \tag{2}$$

where $\mathbf{p}_1^i$ and $\mathbf{p}_2^i$ denote the homogeneous coordinates (e.g., $\mathbf{p}_1^i \equiv (x, y, 1)^T$) of the 2D projection of the i-th point in images 1 and 2 respectively, and $\mathbf{H}$ is a 3×3 matrix representing the homography.

2.1 Homography estimation

Q2.1 Given N point correspondences and using (2), derive a set of $2N$ independent linear equations in the form $\mathbf{A}\mathbf{h} = \mathbf{b}$ where `h` is a 9×1 vector containing the unknown entries of $\mathbf{H}$. What are the expressions for $\mathbf{A}$ and $\mathbf{b}$? How many correspondences will we need to solve for `h`?

Q2.2 Implement a function `def computeH(p1, p2)` . Inputs: `p1` and `p2` should be `2 x N` matrices of corresponded $(x, y)^T$ coordinates between two images. Outputs: `H` should be a 3x3 matrix encoding the homography that best matches the linear equation derived above (in the least squares sense). Hint: Remember that $\mathbf{H}$ will only be determined up to scale, and the solution will involve an SVD.

Q2.3 Implement a function `def computeH_norm(p1, p2)` . This version should normalize the coordinates `p1` and `p2` (e.g., from -1 to 1) prior to calling `compute_H`. Normalization improves numerical stability of the solution. Note that the resulting `H` should still follow Eq. (2). Hint: Express the normalization as a matrix.

[2] You can assume that the camera matrices of the two cameras are given by M_1 and M_2.

[3] For an intuition into this relation, consider the projection of point $\mathbf{P}^i = (X, Y, Z)^T$ into two views $\mathbf{p}_1^i \equiv \mathbf{K}\mathbf{P}^i$ and $\mathbf{p}_2^i \equiv \mathbf{K}\mathbf{R}\mathbf{P}^i$, where $\mathbf{K}$ is a (common) intrinsic parameter matrix, and $\mathbf{R}$ is a 3×3 rotation matrix. In this case: $\mathbf{p}_1^i \equiv \mathbf{K}\mathbf{R}^{-1}\mathbf{K}^{-1}\mathbf{p}_2^i$.

Q2.4 Select enough corresponding points from images *taj*[1, 2].*png*. They should be reasonably accurate and cover as much of the images as possible. Use `computeH_norm` to relate the points in the two views, and plot image *taj*1.*png* with overlayed points `p1` (drawn in red), and (drawn in green) points `p2` transformed into the view from image 1 (call this variable `p2_t`). Save this figure as `q2_4.jpg`. Additionally, save these variables to `q2_4`:

```
H2to1 = computeH_norm(p1, p2):
 ...
np.savez("q2_4.npz", H2to1, p1, p2, p2_t)
```

Write an expression for the average error (in pixels) between `p1` and the transformed points `p2`. This expression should be in terms of $\mathbf{H}$, $\mathbf{p}_1$, and $\mathbf{p}_2$. Compare this to the error that was minimized in items 1 and 2.

2.2 Image warping

The provided function `warp_im=warpH(im, H, out_size)` warps image `im` using the homography transform `H`. The pixels in `warp_im` are sampled at coordinates in the rectangle `(0, 0)` to `(out_size[0]-1, out_size[1]-1)`. The coordinates of the pixels in the source image are taken to be `(0, 0)` to `(im.shape[1]-1,im.shape[0]-1)` and transformed according to `H`.

The following code:

```
import warpH as warpH
H2to1 = computeH_norm(p1, p2)
warp_im = warpH(im2, H2to1, (im1.shape[1], im1.shape[0]))
```

should produce a warped image *im*2 in the reference frame of *im*1. Not all of *im*2 may be visible, and some pixels may not be drawn to.

Figure 1: Original images and warped.

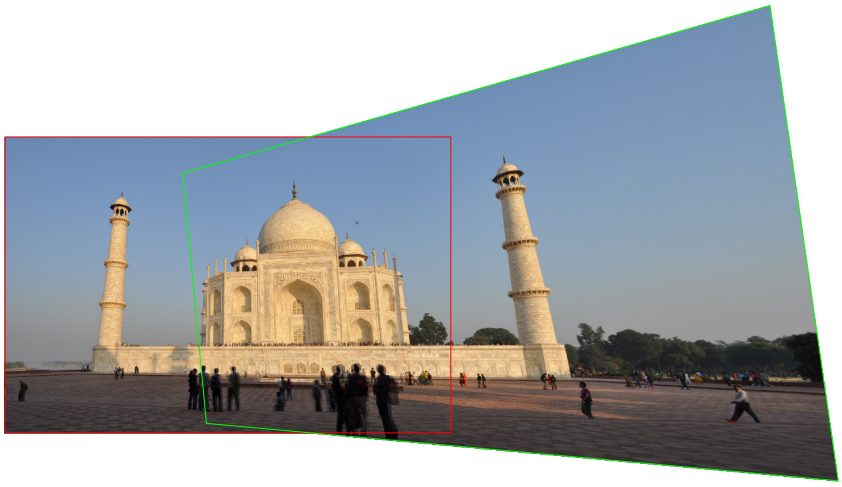
(a) *taj*1.*jpg* (b) *taj*2.*jpg* (c) *taj*2 warped into *taj*1

Q2.5 Write a few lines of Python code to find a matrix `M`, and `out_size` in terms of `H2to1` and `im1.shape` and `im2.shape` such that,

```
warp_im1 = warpH(im2, M, out_size)
warp_im2 = warpH(im2, M @ H2to1, out_size)
```

produces images in a common reference frame where all points in `im1` and `im2` are visible (a mosaic or panorama image containing all views) and out_size[1] == 1280 . **M** should be scaling and translation only, and the resulting image should fit snugly around the transformed corners of the images.

Figure 2: Final mosaic view.



2.3 Image blending

Q2.6 Use the functions above to produce a panorama or mosaic view of the two images, and show the result. Save this image as `q2_7.jpg`. For overlapping pixels, it is common to blend between the values of both images. You can simply average the values. Alternatively, you can obtain a blending value for each image:

```
import scipy.ndimage.distance_transform_bf as bwdist
mask = np.zero(shape = img.shape[:2])
mask[0, :] = mask[-1, :] = mask[:, 0] = mask[:, -1] = 1
mask = bwdist(mask, metric = "cityblock")
mask = mask / mask.max()
```

The function `bwdist` computes the distance transform of the binarized input image, so this mask will be zero at the borders and 1 at the center of the image.

Q2.7 In items $Q2.1-2.3$ we solved for the 8 degrees of freedom of a homography. Given that the two views were taken from the same camera and involve only a rotation, how might you reduce the number of degrees of freedom that need to be estimated?

QX1 (Extra Credit 10 points) Assume that the rotation between the cameras is only around the y axis. Using `H2to1`, and assuming[4] no skew and $\alpha = \beta = 1274.4$,

[4]The previous notation used $\alpha = f_x$, $\beta = f_y$, $u_0 = c_x$, $v_0 = c_y$, and no skew.

$u_0 = 814.1$, and $v_0 = 526.6$, estimate the angle of rotation between the views. You should be able to approximate this with elementary trigonometry.

2.4 Extra credit: Laplacian pyramid blending

Linear blending produces ghosting artifacts when the scene changes between the views (e.g., the people in the example images). In these regions, we should take the pixels from only one image. Unfortunately, having hard seams between images often introduces noticeable discontinuities, as for example an abrupt change in the color of the sky.

Laplacian blending alleviates this problem by blending features at different frequency bands.

QX2 (Extra Credit 10 points) Implement laplacian pyramid blending for images. First construct a gaussian pyramid of each image and mask. Then, construct the laplacian pyramid using the difference of gaussians. Now blend each of the levels together with the corresponding mask, and reconstruct the image. Show your impressive results.

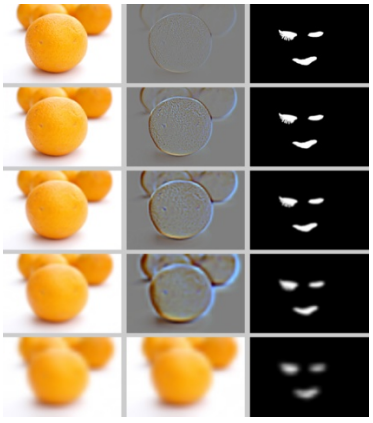
Figure 3: An Annoying Clockwork Orange.

(a) *oranges.jpg*

(b) *alex.jpg*

(c) *mask.jpg*

(d) Gaussian and laplacian pyramids.



(e) Blended result, normal (top), and 5-level pyramid (bottom).

3 Camera Estimation (35 points)

In class we learnt that the camera matrix $\mathcal{M}$ can be decomposed into its intrinsic parameters given by the matrix $\mathbf{K}$ and extrinsic parameters given by the rotation $\mathbf{R}$ and translation $\mathbf{t}$

$$\mathcal{M} = \mathbf{K}\,[\mathbf{R}\ \mathbf{t}] \tag{3}$$

and the camera projection equation is given by:

$$p \equiv \mathcal{M}P \tag{4}$$

The rotation and translation of the camera is also commonly referred to as the pose of the camera. A problem that commonly arises in areas such as model-based object recognition, augmented reality and vision-based robot navigation is that of estimating the pose of the camera given 3D co-ordinates of points on an object/scene and their corresponding 2D projections in the image. In this question we shall explore the use of pose estimation in a problem that arises in motion capture.

3.1 Problem Statement

Given the locations of 3D optical markers placed at the joints of a human subject, and the corresponding locations of the markers in the image, estimate the pose of the camera relative to the human subject.

Begin by loading the file *hw1_prob3.mat* into your Python workspace. You will find two variables in your workspace:

- xy - a 23×2 matrix of the 2D locations of the joints in the image
- XYZ - a 23×3 matrix of the 3D locations of the joint markers in the scene.

For this assignment the 2D image points and the 3D joint marker positions have been generated synthetically, we will first visualize our data (First rule of implementing a computer vision algorithm- *visualize, visualize, visualize*). You have been given two functions *visualize2Dpoints* and *visualize3Dpoints* [5] to help you with this.

3.2 Estimating the camera

Q3.1 Your primary task in this problem is to write the following function and answer the questions that follow:

```
def estimateCamera(xy, XYZ):
  return R, t
```

[5] Type 'help *functionname*' in Python to get information on usage.

which takes as input the 23×2 matrix $\mathbf{xy}$ of 2D locations and the 23×3 matrix of 3D locations $\mathbf{XY}$ and returns as output the rotation R and translation t of the camera in the object co-ordinate system. We will assume that the camera intrinsic parameter matrix K is given and is equal to the identity matrix.

The camera matrix $\mathcal{M}$ that you estimated needs to be separated into a rotation $\mathbf{R}$ and a translation $\mathbf{t}$.

Q3.2 You could imagine simply taking the first 3×3 part of $\mathcal{M}$ to be the rotation of the camera. However, this would be incorrect, explain why.

Q3.3 Under what condition is the 3×3 matrix a rotation matrix ?

Q3.4 How can we convert the first 3×3 portion of the matrix $\mathcal{M}$ into a rotation matrix? [6][7]

Q3.5 Write the following function by making the change you derived in the previous question to correct the camera estimate

```
def estimateCamera2(xy, XYZ):
  return R, t
```

3.3 Visualization

Q3.6 Visualize the given 3D joint marker points using the provided *visualize3Dpoints* function and place the estimated camera in the scene using the provided *drawCam* function.

3.4 Epilogue

In this problem we have implemented a simple version of pose estimation which might not very stable for real applications. There are several current state of the art approaches for estimating the perspective camera pose from 2D-3D correspondences (also known as the Perspective-N-Points problem), such as in the following paper:

> EPnP: Accurate O(n) Solution to the PnP Problem - *Francesc Moreno-Noguer, Vincent Lepetit, Pascal Fua. (IJCV 2008)*

[6] This estimate will actually be the best rotation in a least-squares sense.
[7] Hint: The solution uses an SVD.

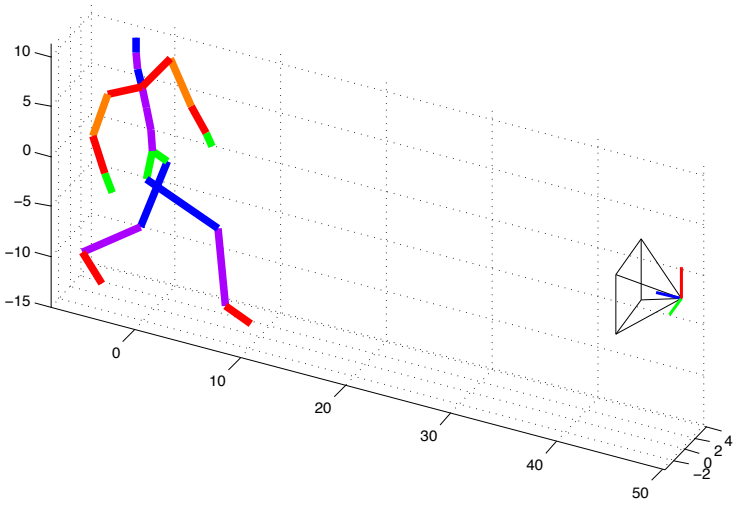


Figure 4: Human skeleton visualization with estimated camera position.