Homework 3
M1522.001000 Computer Vision (2021 Fall)
Due: Tuesday Nov 4, 3:30PM

The goal of this homework is to explore Camera, Homographies and Image warping. There are 6 Questions in this assignment, 3 theory and 3 programming questions.

1 Submitting Your Assignment

Your submission for this assignment must include answers to theory questions, the code for your **Python** implementation and a short writeup describing any thresholds and parameters used, interesting observations you made or things you did differently while implementing the assignment. Make sure all the files needed for your code to run (except data) are included. Your writeup should be **typed** with **English**.

Put all your files into a directory called "(studentid)-(yourname)-HW3" and pack it into a **zip** named "**(studentid)-(yourname)-HW3.zip**". For example, 202112345-gildonghong-HW3.zip. Upload your **zip** file to **ETL** until due date. Refer to the **ETL** page for the policies regarding collaboration, due dates, extensions, and late days.

Your submitted zip file should include the files arranged in this layout:

- `theory.txt(theory.doc)` or `theory.pdf` : answers to the theory questions
- Folder `result`
  - `porto1_warped.png` : png image showing the result of warping applied to porto1.png
  - `porto_merged.png` : png image showing the result of combining porto1.png and porto2.png
  - `iphone_rectified.png` : png image showing the result of rectifying iphone.png
  - `writeup.txt(writeup.doc)` or `writeup.pdf` : the writeup describing your experiments
- Folder `code`
  - `main.py`

Refer to Section 3 and Section 4 for the details of the above files. Your zip file should be sent in before the due. Later than that, only one late day is allowed. Good Luck!

2 Warm Up

This section is meant to provide a short introduction to get you used to working with images in this homework. It is recommended that you do this sections early so that you would not be overwhelmed by what the rest of the assignment tasks.

Let us start with the toys image (*toys.jpg*) and perform some simple image warping. Begin with a simple operation that will stretch the image in the horizontal direction. One possible method that can be used is to apply a transformation matrix to each point/pixel in the image. Apply the transform T_h to the image (*i.e.*, $T_h P_{i,j}$, where $P_{i,j} = [i, j, 1]^T$ is a point in the image expressed in homogeneous coordinates).

$$T_h = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The transformation matrix T_h will stretch the image in the horizontal direction by a factor of 2. Properly generate the warped image without holes in the new image by mapping every point/pixel in the new image from a corresponding value in the old image and not the reverse. (This means that your code should iterate for every pixel in the warped image which has twice as much pixels as the original image in the horizontal direction and the same number of pixels in the vertical direction). It is important to observe the relationship between the transformation matrix T_h and the resultant image. Further explore this relationship by modifying the transformation matrix to stretch the image in the vertical direction.

Note that while we use the (horizontal, vertical) format to each point/pixel (*i.e.*, $P_{i,j} = [i, j, 1]^T$), image matrix is referenced using the (row, column) format. You can easily check it in the `main.py` code. (Compare variable `igs_in` to variable `p_in`.) Therefore, pay careful attention to which format you are using while mapping the points to the warped image. If done properly, both sets of points should appear in the same location in their corresponding images.

3 Theory Questions

Question1: Camera model (30 points)

Consider the camera model below.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K} \begin{pmatrix} \mathbf{R} \mid \mathbf{t} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1}$$

$$\mathbf{K} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}, \mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}, \tag{2}$$

The coordinate $[u, v, 1]^T$ and $[X, Y, Z, 1]^T$ represent homogeneous coordinates in the image and world coordinate system, respectively. $\mathbf{K}$ is the camera intrinsic matrix, $\mathbf{R}$ is the rotation matrix, and $\mathbf{t}$ is the translation vector. Note that the rotation matrix has some unique properties, that each row vectors and column vectors are unit vectors, and $\mathbf{R}^{-1} = \mathbf{R}^T$. (You can use these properties without any proof.)

(a) What is the image coordinates of the world coordinate system's origin $O(0, 0, 0)$? [5 points]

(b) What is the coordinate of the camera in the world coordinate system? [5 points]

(c) In the camera coordinate system, camera is looking at $[0, 0, 1]^T$ (*i.e.* $+z$). Convert this direction into the world coordinate system with a unit vector. [8 points]

(d) The straight line in the world coordinate system can be expressed as $[d_1, d_2, d_3]^T t + [x_1, x_2, x_3]^T$ ($t \in \mathbf{R}$). Assume that this line is not parallel to the xy-plane and does not path through the origin in the camera coordinate system. What is the expression of this line in the image? [12 points]

Question2: Camera Calibration (10 points)

A camera is a mapping between the 3D world $\mathbf{X_i} = \begin{bmatrix} X_i & Y_i & Z_i & 1 \end{bmatrix}^T$ and a 2D image $\mathbf{x_i} = \begin{bmatrix} u_i & v_i & 1 \end{bmatrix}^T$ (both are homogeneous coordinates). A camera projection matric $\mathbf{P} = [m_{ij}]$ maps $\mathbf{x_i} = \mathbf{P}\mathbf{X_i}$. Given $n \geq 6$ pairs of correspoinding points, we have an over-determined system of equations, $\mathbf{Ap} = \mathbf{0}$, where $\mathbf{p}$ is the vectorized form of $\mathbf{P}$. Consider the SVD of $\mathbf{A}$ as $\mathbf{A} = \mathbf{U\Sigma V^T}$. Show that the solution $\mathbf{p} = \min_p \|Ap\|, \|p\| = 1$ is the last column of $\mathbf{V}$ corresponding to smallest singularvalue of $\mathbf{A}$.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

Question3: Homography (10 points)

We can use homographies to create a panoramic image from multiple views of the same scene. This is possible 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:

$$\boldsymbol{p_1}^{\boldsymbol{i}} \equiv \boldsymbol{H}\boldsymbol{p_2}^{\boldsymbol{i}} \tag{4}$$

where $\boldsymbol{p_1}^{\boldsymbol{i}}$ and $\boldsymbol{p_2}^{\boldsymbol{i}}$ denote the homogeneous coordinates (*e.g.*, $\boldsymbol{p_1}^{\boldsymbol{i}} \equiv (x, y, 1)^T$) 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. Given N point correspondences and using (4), derive a set of 2N independent linear equations in the form $\mathbf{Ah} = \mathbf{b}$ where $\mathbf{h}$ is a 9×1 vector containing the unknown entries of $\mathbf{H}$.

(a) What are the expressions for $\mathbf{A}$ and $\mathbf{b}$? [3 points]

(b) How many correspondences will be needed to solve for $\mathbf{h}$? [4 points]

(c) How can we get $\mathbf{H}$? (Hint: Question 2) [3 points]

4 Programming

First of all, make sure you have Python 3.6+ installed. You can check your version with command `python3 -V`. To run the program, you will also need Numpy and Pillow

packages. To install them with the package manager for Python, run: `pip3 install numpy pillow`. Usage of third-party image processing library including **OpenCV** for this part is **prohibited**.

Part 1 Estimating transformations from the image points (15 points)

Write a Python function and explain your code in the writeup.

```
H = compute_h(p1, p2)
```

Inputs: `p1` and `p2` should be $N \times 2$ matrices of corresponded $(x, y)^T$ coordinates between two images. Outputs: **H** should be a 3×3 matrix encoding the homography that best matches the linear equation derived above (from `p2` to `p1`). Hint: Remember that **H** will only be determined up to scale, and the solution will involve an SVD. You are allowed to use existing SVD function including `numpy.linalg.svd`, `scipy.linalg.svd`, etc.

```
H = compute_h_norm(p1, p2)
```

This version should normalize the coordinates `p1` and `p2` (*e.g.*, from 0 to 1). Normalization improves numerical stability of the solution. Note that the resulting **H** should still follow Eq. (4). Hint: Express the normalization as a matrix.

Part 2 Mosaicing (20 points)

Write a Python function and explain your code in the writeup.

```
p_in, p_ref = set_cor_mosaic()
```

which returns `p_in` and `p_ref`, $N \times 2$ matrices of corresponded $(x, y)^T$ coordinates between two images. Set a criterion and select the correspondences from given images. From there, you will be able to calculate **H** for `warp_image` function. Explain the criterion and the number of correspondences used.

```
igs_warp, igs_merge = warp_image(igs_in, igs_ref, H)
```

which takes an input as image *igs_in*, a reference image *igs_ref*, and a 3×3 homography, and returns 2 images (*igs_warp* and *igs_merge*) as outputs.

The first output image is *igs_warp*, which is the image *igs_in* warped according to **H** to be in the frame of the reference image *igs_ref*. **H** must be derived by you from the given image.

The second output image is *igs_merge*, a single mosaic image with a larger field of view containing both the input images.

In order to avoid aliasing and sub-sampling effects, consider producing the output by solving for each pixel of the destination image rather than mapping each pixel in the input image to a point in the destination image (which will leave holes). Also note that the input and output images will be different dimensions.

Part 3 Rectification (15 points)

The rectification is to make the new image plane parallel to the wall as best as possible.

Write a Python function and explain your code in the writeup.

```
c_in, c_ref = set_cor_rec()
```

which returns `c_in` and `c_ref`, $N \times 2$ matrices of corresponded $(x, y)^T$ coordinates between two images. You can select the four corner points (the intersections of the four boundary straight lines; *i.e.* `c_in`) and target corner locations(*i.e.* `c_ref`) to compute the homography matrix. From there, you will be able to calculate **H** for `rectify` function.

```
igs_rec = rectify(igs, p1, p2)
```

Compute the 3×3 homography matrix to rectify the following image of an iphone.

Finally, note that we will use a code similarity checker to detect plagiarism. You are expected to work on the assignment individually. I firmly believe that every students can do his or her own work. For your sake, please do not copy and paste others code! SWEET AFTER BITTER!