Homework 1

- ***Department:*** Electrical Engineering
- ***Name:*** Ching-Hsiang Wu (武敬祥)
- ***ID:*** R11921080

Content

Problem1: Homography estimation

Result

- k = 4
    - 1-0.png → 1-1.png

- Error compares (DLT vs. NDLT vs. NDLT+RANSAC)

```
Error DLT :  138.81717720490943
Error NDLT :  138.8171771783534
Error NDLT+RANSAC :  138.81717717832524
```

- 1-0.png → 1-2.png

  - Error compares (DLT vs. NDLT vs. NDLT+RANSAC)

```
Error DLT :  344.97096045336076
Error NDLT :  344.9709604503662
Error NDLT+RANSAC :  344.9709604503649
```

- k = 8
  - 1-0.png → 1-1.png

  - Error compares (DLT vs. NDLT vs. NDLT+RANSAC)

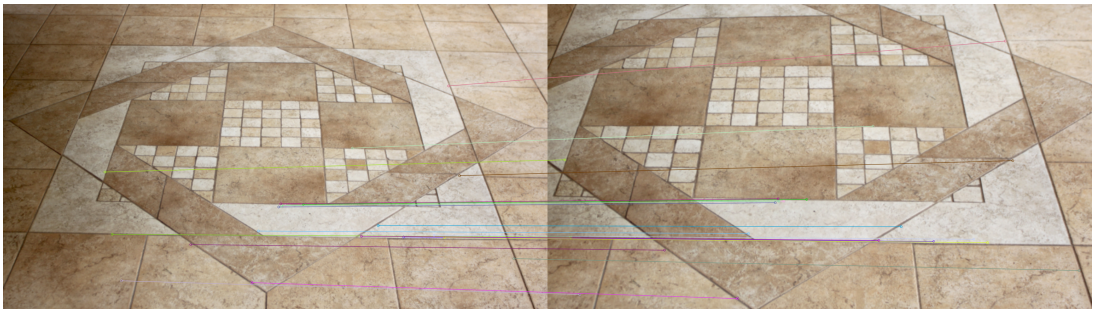
```
Error DLT :  1.5022241505957994
Error NDLT :  1.4366317072232373
Error NDLT+RANSAC :  1.2221525700258276
```

  - 1-0.png → 1-2.png

- Error compares (DLT vs. NDLT vs. NDLT+RANSAC)

```
Error DLT :  530.8248710367938
Error NDLT :  674.7975197541583
Error NDLT+RANSAC :  17.031289345570 68
```

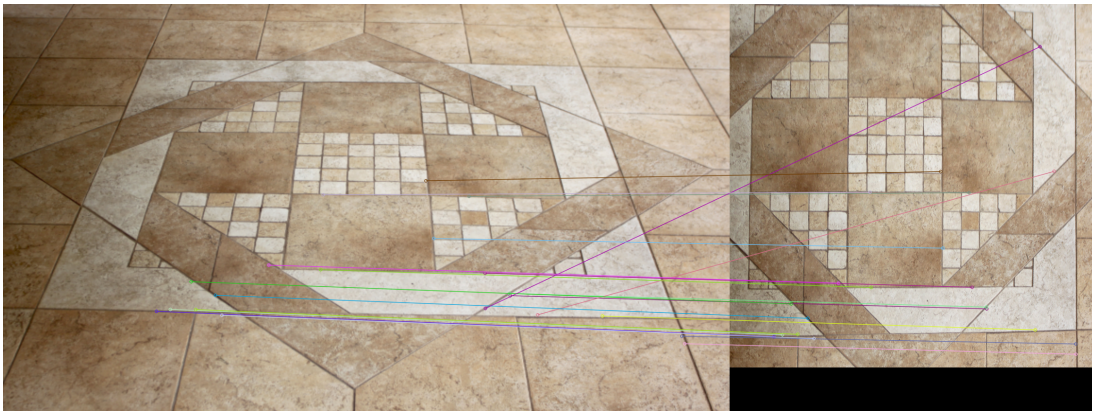
- k = 20
  - 1-0.png → 1-1.png



- Error compares (DLT vs. NDLT vs. NDLT+RANSAC)

```
Error DLT :  0.2883449711074633
Error NDLT :  0.2798449331565398
Error NDLT+RANSAC :  0.2682353786798386
```

  - 1-0.png → 1-2.png



- Error compares (DLT vs. NDLT vs. NDLT+RANSAC)

```
Error DLT :  552.7159341061513
Error NDLT :  651.5138691770323
Error NDLT+RANSAC :  3.2533197946775
```

Procedure

1. Finding the key points and descriptors from two images with **sift algorithm (cv.SIFT_create())**.
2. Obtaining the top 2 nearest key points pairs with the **brutal force matching method(cv.BFMatcher() & knnMatch())**
3. Obtaining the key points pairs by comparing the distance of the nearest key points pair and the second nearest. If the nearest key points pair is less than 0.75 times the second nearest key points pair, then we consider the nearest key points pair to be a good pair.
4. After obtaining the key points pair, we calculate the homography with
   a. **Direct Linear Transform(DLT),**
   b. **Normalized Direct Linear Transform(NDLT),**
   c. **Normalized Direct Linear Transform(NDLT) + RANSAC.**
5. Calculating the **Root Mean Square Error(RMSE)** between the estimated and the ground truth key points of the train image.

Discussion

1. The missed **scaling factor** of the homography.

One of the difficulties I have encountered is the scaling factor of the homography I missed. When computing the reprojection error, the formula is,

$$\hat{p_{t_i}} = \lambda_i \cdot \mathcal{H} \cdot p_{s_i}$$

$$error = \frac{1}{N} \sum_{i=1}^{N} \| p_{t_i} - \hat{p_{s_i}} \|$$

However, I forgot to times the scaling factor (λ_i) when calculating the $\hat{p_{t_i}}$, which resulted in a very large reprojection error at first. It took me a little time to correct it.

2. **RANSAC** implementation.

   RANSAC is short for **RAN**dom **SA**mple **C**onsensus, which is raised to eliminate the effect posed by the outliers. We summarize the error in the table below,

(DLT, NDLT, NDLT+RANSAC)	k=4	k=8	k=20
1-0 → 1-1	(138.82, 138.82, 138.82)	(1.50, 1.44, 1.22)	(0.29, 0.28, 0.27)
1-0 → 1-2	(344.97, 344.97, 344.97)	(530,82 674.80, 17.03)	(552.72, 651.51, 3.25)

Let's focus on the case of **DLT vs. NDLT** and ignore the case of **NDLT+RANSAC** first. For the case **1-0 → 1-1**, using NDLT has a better performance than the case of using DLT in general. However, when speaking to the case of **1-0→1-2**, the opposite is true. From this phenomenon, I infer it is because of the effect of the outliers. As a result, I implemented **RANSAC** to eliminate the effect of the outliers, and the result is getting normal as I expected. The parameter I used in the RANSAC is as follows:

- **threshold = 3,** the maximum allowed reprojection error to treat a point pair as an inlier.
- **iteration**: **2000,** the maximum number of RANSAC iterations.

On the other hand, RANSAC is very sensitive to the parameter (threshold, iteration). In my experiment, the error will explode sometimes. Here comes an example,

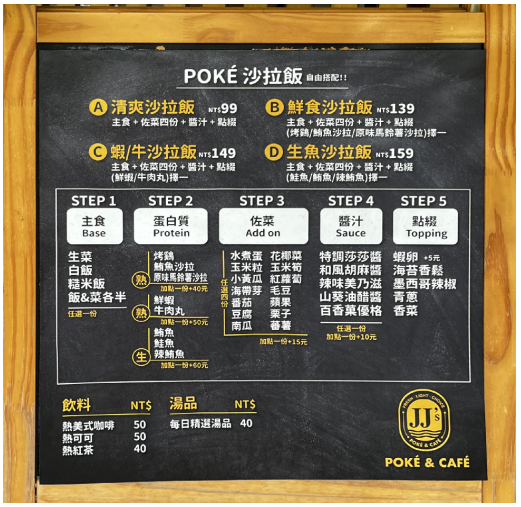
```
Error DLT :  530.8248710367938
Error NDLT :  674.7975197541583
Error NDLT+RANSAC :  1407.6415420947885
```

Problem2: Document rectification

Result

- Input image

- Rectified image



Procedure

1. Select **four corners** (left-top corner first) from the input image **clockwise** by modifying the example code `mouse_click_example.py` .

2. the rectified image size is not the same size as the input one due to the following reasons
    a. **Warping time:** the resolution of the input image is 3024x4032 (captured by a 12 million-pixel camera), it will take too much time in the warping procedure.

b. **Image distortion:** the aspect ratio of the cropped image and the input image won't necessarily match each other, which will result in image deformation.

As a result, the rectified image size $(x * y)$ will follow the equations

$$x = \|p_1 - p_2\|$$
$$y = \|p_1 - p_3\|$$

to better match the cropped image size and also decrease the processing time of warping.

3. Using **Normalized Direct Linear Transform** to obtain the homography.
4. Using **Backward warping** and **Bilinear interpolation** to obtain the rectified image. (It may take a few seconds).

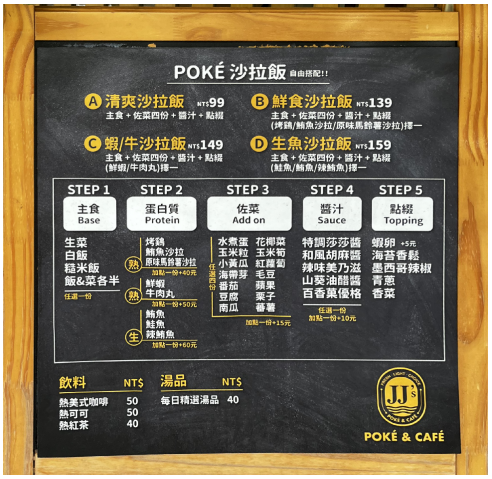
Discussion

1. **Nearest-neighbor** interpolation vs. **Bilinear** interpolation.

   When implementing the backward warping, we can choose to use the Nearest-neighbor interpolation or the Bilinear interpolation. Here are the results of these two methods.

- Nearest-neighbor interpolation

- Bilinear interpolation



At first glance, we cannot tell what's the difference. However, once we zoom in on these images,

- Nearest-neighbor interpolation
- Bilinear interpolation

the difference is obvious. The image after performing the bilinear interpolation is much smoother. what's more, if we compare the rectified image after performing the bilinear interpolation with the input image, they have a high similarity.

- Bilinear interpolation

- Input image

Package Introduction

cv3D.py

- `findHomographyDLT(points1, points2, k)`

  Finding the homography(H) by Direct Linear Transform(DLT) method.

  Parameters:

- **points1**: the key points of the query(first) image.
- **points2**: the key points of the train(second) image.
- **k**: the number of points to calculate the homography.

Return:

- **H**: the tomography matrix.

- `findHomographyNDLT(points1, points2, k, method = 0, threshold=3, iteration=2000, numPick=4)`

  Finding the homography(H) by Normalized Direct Linear Transform(NDLT) method.

  Parameters:

  - **points1**: the key points of the query(first) image.
  - **points2**: the key points of the train(second) image.
  - **k**: the number of points to calculate the homography.
  - **method**: the method used to compute a homography matrix. The following methods are possible:
    - 0: a regular method using all the points (default).
    - “RANSAC”: RANSAC-based robust method.
  - **threshold**: maximum allowed reprojection error to treat a point pair as an inlier (used in the RANSAC methods only). (default = 3).
  - **iteration**: the maximum number of RANSAC iterations. (default = 2000).
  - **numPick**: number point pairs to calculate the homography in one iteration. (default=4, **numPick** > 4, **numpick** $\leq k$).

  Return:

  - **H**: the tomography matrix.

- `reprojectionErrorEstimation(H, gt)`

  Calculating the error between the ground truth points from the train image and the estimated key points from multiplying the key points of the query image by the homography.

  Parameters:

  - **H**: the homography matrix between the query image and the train image

- **gt**: the ground truth pairs of the query image and the train image. The data shape is $(2, 100, 2)$

Return:

- **error**: the Root Mean Square Error(RMSE) between the estimated and the ground truth key points of the train image.

- `warpPerspective(H , srcimage, dstimage, method = "NN"):`

  Calculating the rectified image by multiplying the source image by the homography.

  Parameters:

  - **H**: the homography matrix between the source image and the rectified image.
  - **srcimage**: the image we want to rectify.
  - **dstimage**: the rectified image.
  - **method**: the method to get the dstimage when performing the backward warping. The following methods are possible:
    - "NN": the Nearest-neighbor interpolation (default).
    - "Bilinear": the Bilinear interpolation.

  Return:

  - **dstimage**: the rectified image.

How to Execute?

- **Environment**
  - `Python == 3.6.13`
  - `Opencv == 4.5.1`
  - `Numpy == 1.19.5`
- **Problem1**
  1. Run the following command on the terminal: `python 1.py ./images/1-0.png ./images/1-{i}.png ./groundtruth_correspondences/correspondence_0{i}.npy {k}`
     - **i**: image index.
     - **k**: number of pairs of points that are used in finding homography.

- **Problem2**
  1. Run the following command on the terminal: `python 2.py ./images/4.jpg`
  2. Select(click) **four corners** (left-top corner first) from the input image **clockwise,** then **press “Esc”.**

Youtube Link

3DCV__Homework1

This is a demonstration showing how to execute the code of 3DCV homework 1.

https://youtu.be/bvpJhs9JNoo