Assignment 3:

Projective Geometry

Computer Vision

National Taiwan University, Spring 2025

Announced: 2025/04/11(Fri.)

Due: 2025/05/01(Thu.) 23:59

Outline

Part 1: Homography Estimation

- Familiar DLT estimation method
- Practice forward warping

Part 2: Marker-Based Planar AR

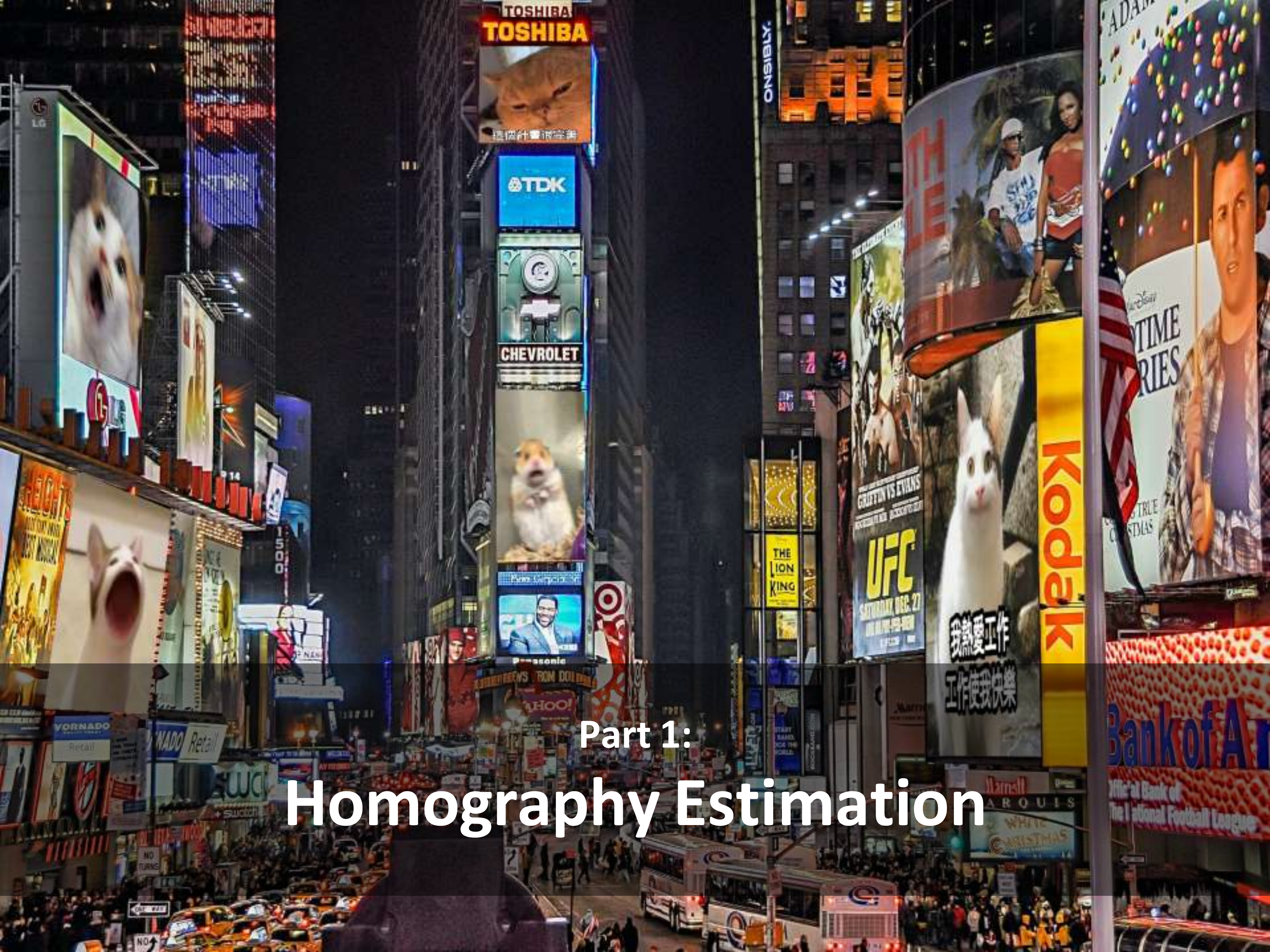
- Familiar with off-the-shelf ArUco marker detection tool
- Practice backward warping

Part 3: Unwarp the Secret

- What can go wrong with practical homography?

Part 4: Panorama

- RANSAC



Part 1:
Homography Estimation

Recap of Homography (1/2)

- Matrix form:

$$\begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix}$$

- Equations:

$$v_x = \frac{h_{11}u_x + h_{12}u_y + h_{13}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

$$v_y = \frac{h_{21}u_x + h_{22}u_y + h_{23}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

Recap of Homography (2/2)

- Degree of freedom
  - There are 9 numbers in $\mathbf{H}$.
  - Q: Are there 9 DoF? Ans: No.
  - We can multiply all h_{ij} by nonzero k without changing the equations:

$$v_x = \frac{kh_{11}u_x + kh_{12}u_y + kh_{13}}{kh_{31}u_x + kh_{32}u_y + kh_{33}} \qquad v_x = \frac{h_{11}u_x + h_{12}u_y + h_{13}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

$$v_y = \frac{kh_{21}u_x + kh_{22}u_y + kh_{23}}{kh_{31}u_x + kh_{32}u_y + kh_{33}} \Rightarrow v_y = \frac{h_{21}u_x + h_{22}u_y + h_{23}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

8 DoF solution

- **Solution 1:** set $h_{33} = 1$

$$v_x = \frac{h_{11}u_x + h_{12}u_y + h_{13}}{h_{31}u_x + h_{32}u_y + 1}$$

$$v_y = \frac{h_{21}u_x + h_{22}u_y + h_{23}}{h_{31}u_x + h_{32}u_y + 1}$$

- **Solution 2:** impose unit vector constraint

$$v_x = \frac{h_{11}u_x + h_{12}u_y + h_{13}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

$$v_y = \frac{h_{21}u_x + h_{22}u_y + h_{23}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

Subject to

$$\sum_{i=1}^{3}\sum_{j=1}^{3} h_{ij} = 1$$

Solution 1 (1/2)

$$v_x = \frac{h_{11}u_x + h_{12}u_y + h_{13}}{h_{31}u_x + h_{32}u_y + 1}$$

- Set $h_{33} = 1$

$$v_y = \frac{h_{21}u_x + h_{22}u_y + h_{23}}{h_{31}u_x + h_{32}u_y + 1}$$

- Multiply by denominator

$$(h_{31}u_x + h_{32}u_y + 1)v_x = h_{11}u_x + h_{12}u_y + h_{13}$$
$$(h_{31}u_x + h_{32}u_y + 1)v_y = h_{21}u_x + h_{22}u_y + h_{23}$$

$$h_{11}u_x + h_{12}u_y + h_{13} - h_{31}u_xv_x - h_{32}u_yv_x = v_x$$
$$h_{11}u_x + h_{22}u_y + h_{23} - h_{31}u_xv_y - h_{32}u_yv_y = v_y$$

Solution 1 (2/2)

- Solve linear system

$$\begin{array}{ccccc}
 & 2N \times 8 & 8 \times 1 & & 2N \times 1 \\
\begin{array}{c}\textbf{Point 1}\\ \\ \textbf{Point 2}\\ \\ \textbf{Point 3}\\ \\ \textbf{Point 4}\\ \\ \end{array} &
\begin{bmatrix}
u_{x,1} & u_{y,1} & 1 & 0 & 0 & 0 & -u_{x,1}v_{x,1} & -u_{y,1}v_{x,1} \\
0 & 0 & 0 & u_{x,1} & u_{y,1} & 1 & -u_{x,1}v_{y,1} & -u_{y,1}v_{y,1} \\
u_{x,2} & u_{y,2} & 1 & 0 & 0 & 0 & -u_{x,2}v_{x,2} & -u_{y,2}v_{x,2} \\
0 & 0 & 0 & u_{x,2} & u_{y,2} & 1 & -u_{x,2}v_{y,2} & -u_{y,2}v_{y,2} \\
u_{x,3} & u_{y,3} & 1 & 0 & 0 & 0 & -u_{x,3}v_{x,3} & -u_{y,3}v_{x,3} \\
0 & 0 & 0 & u_{x,3} & u_{y,3} & 1 & -u_{x,3}v_{y,3} & -u_{y,3}v_{y,3} \\
u_{x,4} & u_{y,4} & 1 & 0 & 0 & 0 & -u_{x,4}v_{x,4} & -u_{y,4}v_{x,4} \\
0 & 0 & 0 & u_{x,4} & u_{y,4} & 1 & -u_{x,4}v_{y,4} & -u_{y,4}v_{y,4}
\end{bmatrix} &
\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} & = &
\begin{bmatrix} v_{x,1} \\ v_{y,1} \\ v_{x,2} \\ v_{y,2} \\ v_{x,3} \\ v_{y,3} \\ v_{x,4} \\ v_{y,4} \end{bmatrix} \\
\textbf{Additional points} & \vdots & & & \vdots
\end{array}$$

Solution 2 (1/2)

- A more general solution by constraining $\sum_{i=1}^{3}\sum_{j=1}^{3} h_{ij} = 1$

$$v_x = \frac{h_{11}u_x + h_{12}u_y + h_{13}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

$$v_y = \frac{h_{21}u_x + h_{22}u_y + h_{23}}{h_{31}u_x + h_{32}u_y + h_{33}}$$

- Multiply by denominator

$$(h_{31}u_x + h_{32}u_y + h_{33})v_x = h_{11}u_x + h_{12}u_y + h_{13}$$
$$(h_{31}u_x + h_{32}u_y + h_{33})v_y = h_{21}u_x + h_{22}u_y + h_{23}$$

$$h_{11}u_x + h_{12}u_y + h_{13} - h_{31}u_xv_x - h_{32}u_yv_x - h_{33}v_x = 0$$
$$h_{21}u_x + h_{22}u_y + h_{23} - h_{31}u_xv_y - h_{32}u_yv_y - h_{33}v_y = 0$$

Solution 2 (2/2)

- Similarly, we have a linear system like this:

$$\underset{2N \times 9}{\mathbf{A}} \; \underset{9 \times 1}{\mathbf{h}} = \underset{2N \times 1}{\mathbf{b}}$$

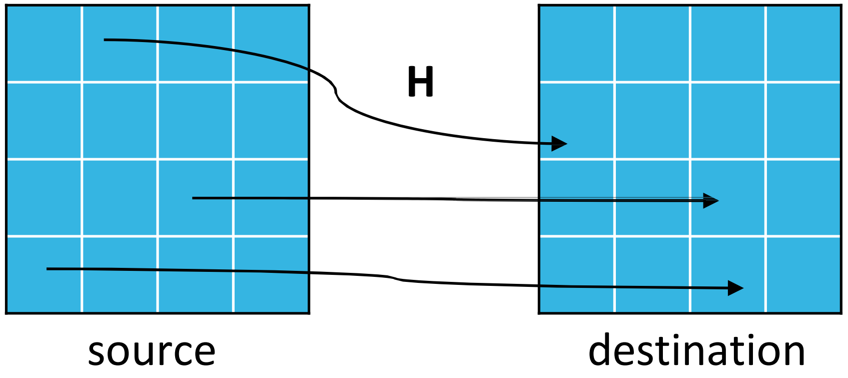
- Here, **b** is all zero, so above equation is a homogeneous system. Hence, we are searching for the null space of A
- Solve:
  - $Ah = 0$
  - SVD of $A = U\Sigma V^T$
  - Let h be the last column of V.
    - i.e. corresponds to the eigenvalue closest to zero.

Input data: Canvas

Input data: Materials

How to warp images?

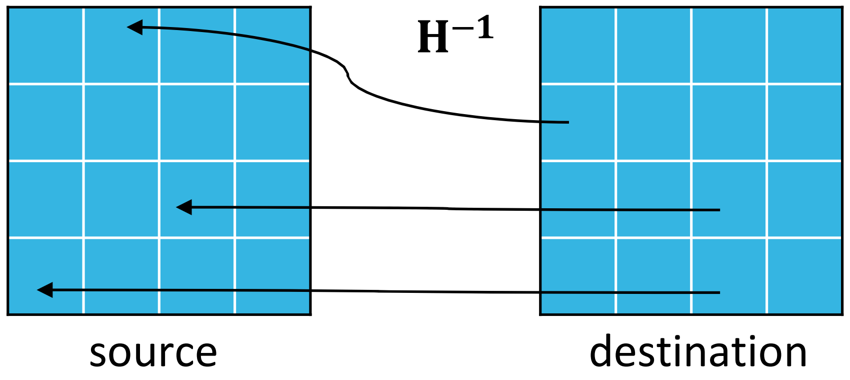
1. Forward warp (intuitive)



- What if we get a sub-pixel location like (95.27, 88.77) ?
  - Nearest neighbor
  - Bilinear interpolation

- What other problems might cause ?
  - Holes!
  - Since possibly not every destination locations is mapped

2.Backward warping



- Now, the destination pixels are densely mapped
- Hence, the transformation must not be singular
    - i.e. the inverse should exist

Assignment Description

- Part 1
  - Goal: Implement solution 1 or 2 for estimating homography, output the ***forward warped*** canvas output1.png
  - Paste output1.png in your report.

Part 2:

Marker-Based Planar AR

ArUco

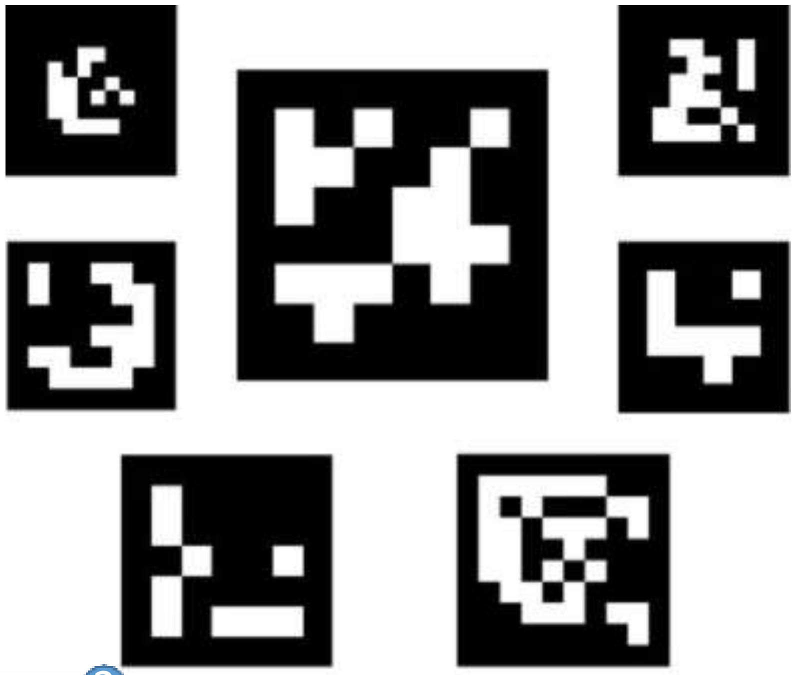
- An off-the-shelf marker generate & detection tool
  - See [here] for details

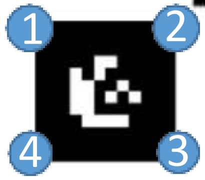
Use function

aruco.detectMarkers( )

to get the detected corners

- The returned corners is arranged in a clock-wise order starting from the left-top corner


Example of markers images



Assignment Description

- Part 2
  - Process the given video and **backward warp** the template image per frame
    - Since we are using backward warping, there should be no holes.
  - The output video should contain the warped template image as if it were there.
  - Paste the function code *solve_homography(u, v)* and *warping( )* in your report. (both forward & backward)
  - Briefly introduce the interpolation method you use

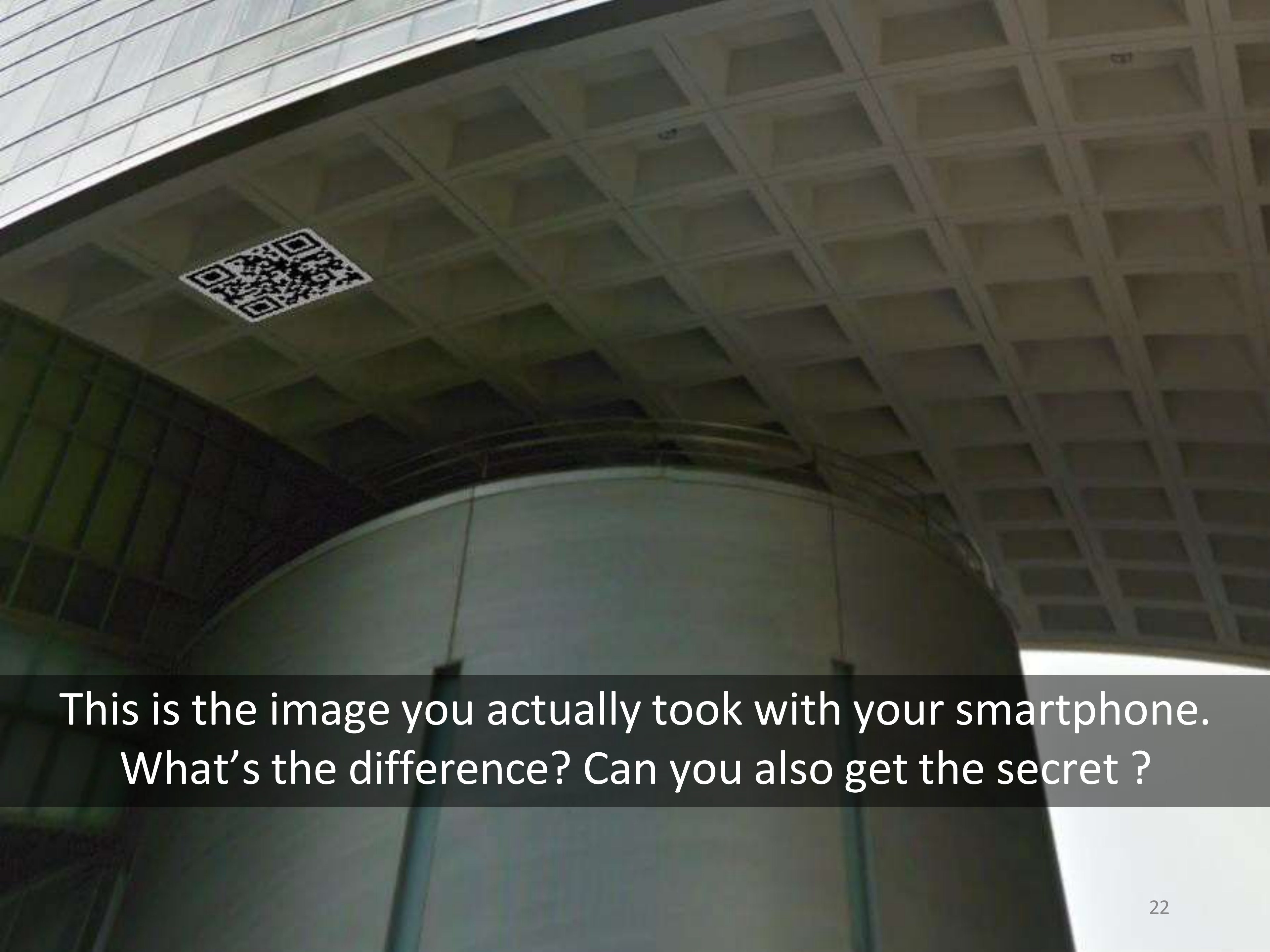
Part 3:

Unwarp the Secret

Once upon the time, you were taking a walk in BL …

ICTAC
THEORETICAL ASPECTS OF COMPUTING

Can you find out BL’s secret?



This is the image you actually took with your smartphone.
What's the difference? Can you also get the secret ?

Assignment Description

- Part 3
  - Unwarp the QR code with backward warping
  - Can you get the correct QR code link from both images?
  - Discuss the difference between 2 source images, are the warped results the same or different?
  - If the results are the same, explain why.
  - If the results are different, explain why.

Part 4:

Panorama

Photo taken at an elementary school near NCCU in a beautiful morning

Field of view (FOV)

- Human have a $\sim 210°$ horizontal FOV without eye movements

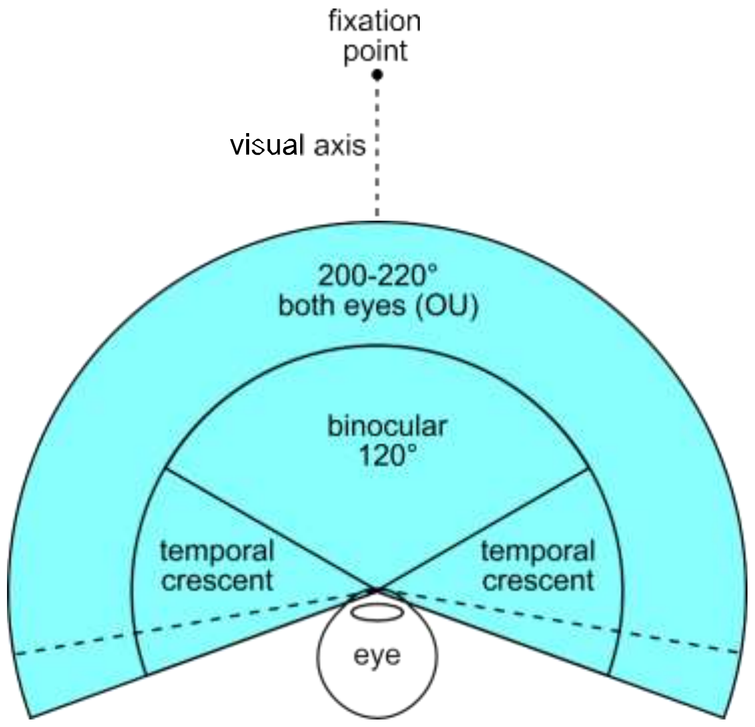
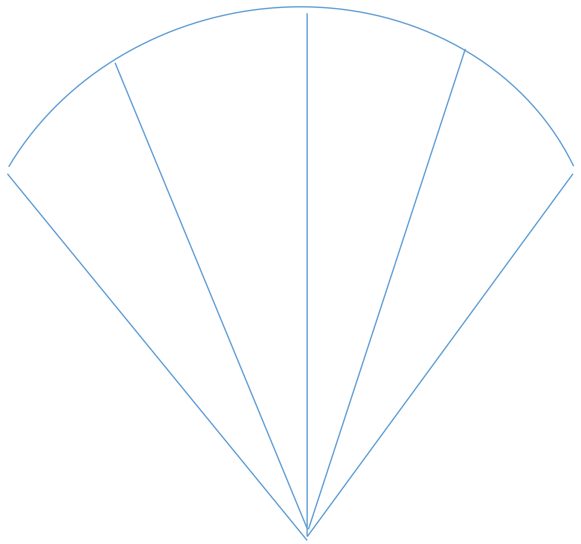


image credit to: Zyxwv99

Typical wide angle camera has a horizontal FOV of around 115 °



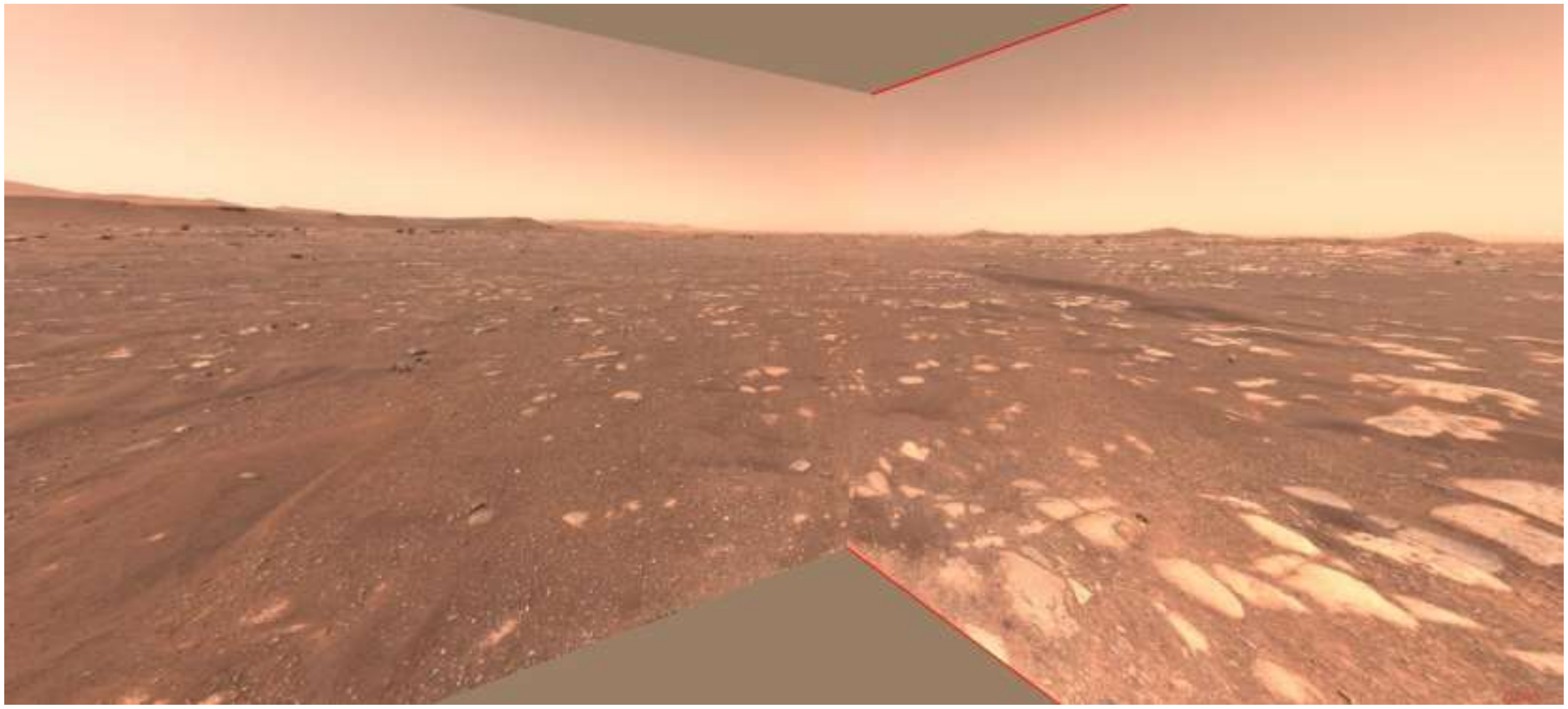
https://en.wikipedia.org/wiki/Field_of_view

A 360-degree panorama of the Milky Way at the Very Large Telescope

A 360-degree panorama of the Milky Way at the Very Large Telescope. Such a panorama shows the entire field of view (FOV) of the telescope in a single image. In the image, the Milky Way appears like an arc of stars spanning horizon to horizon with two streams of stars seemingly cascading down like waterfalls. Image credit: ESO/S. Brunier - Cascading Milky Way (CC BY 4.0)

https://en.wikipedia.org/wiki/Field_of_view#/media/File:Cascading_Milky_Way.jpg

Image taken by the Navigation Cameras aboard the Perseverance rover

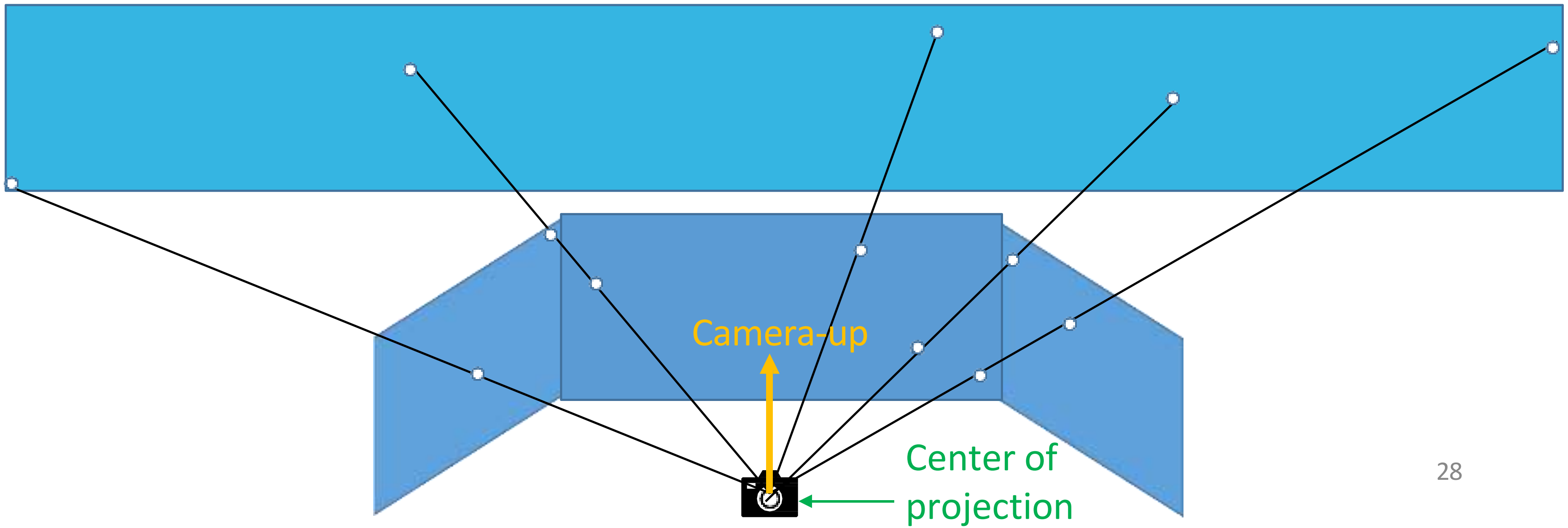


This image shows the flight zone of NASA’s Ingenuity Helicopter from the perspective of NASA’s Mars 2020 Perseverance rover. The flight zone is the area within which the helicopter will attempt to fly.

https://mars.nasa.gov/technology/helicopter/#

Can homography deal with this?

- What is the limitation of homography?
  - Planar!
- We can regard world points are on a plane at infinity if the consecutive images…
  - Have the same center of projection! (e.g. rotating camera by its “up” vector)



Assignment Description

- Part4
  - Implement the function panorama( )
    - Estimate the homography between 3 images
    - Using feature matching, RANSAC to find correct transform
  - Stitch 3 images using backward warping
    - Please call the already written function warping( ) in part 2
  - Feature detection & matching
    - Use opencv built-in ORB detector for keypoint detection.
      - ORB_create( ), detectAndCompute( )
    - You can use opencv brute force matcher for feature matching
      - cv.BFMatcher( ), match( )
    - See this tutorial for details

Assignment Description

- Part4
  - RANSAC
    - Please implement RANSAC efficiently (see P. 47-49)
    - Choose suitable iteration numbers, thresholds for producing reliable homography matrix
      - Set a fix random seed for TA to run & reproduce the result
  - Tips for stitching more than 2 images ($\mathbf{H}^{-1}$is needed for backward warping)
    - Assign frame 1 to destination map D
    - Stitch frame 1 & 2 first with $\mathbf{H_1}$ to destination map D
    - Then, find $\mathbf{H_2}$ between frame 2 & 3
    - Stitch frame 3 to destination map D with $\mathbf{H_3} = \mathbf{H_1}\mathbf{H_2}$
    - …
    - find $\mathbf{H}_n$ between frame n & n+1
    - Stitch frame n+1 to destination map D with $\mathbf{H_{n+1}} = \mathbf{H_1} \dots \mathbf{H_n}$

Q: Can all consecutive images be stitched into a panorama?

E.g.

- Images photoed with camera translation
- Non-planar scene (scene of in-door view)

Answer the question in your report!

- If yes, explain your reason.
- If not, explain under what conditions will result in a failure?

(You can verify with different image sets)

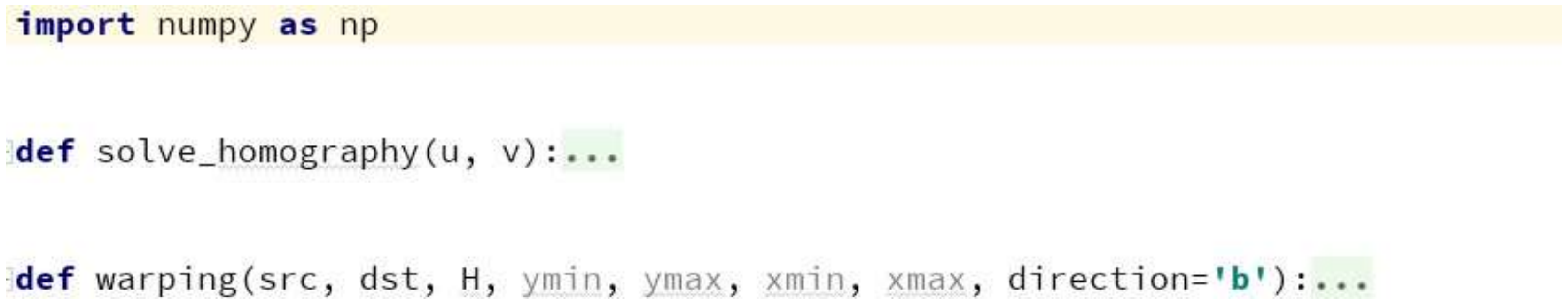
References:

- M. Brown, D. G. Lowe, Recognising Panoramas, ICCV 2003.
- https://docs.opencv.org/master/d9/dab/tutorial_homography.html

HW3 file directory

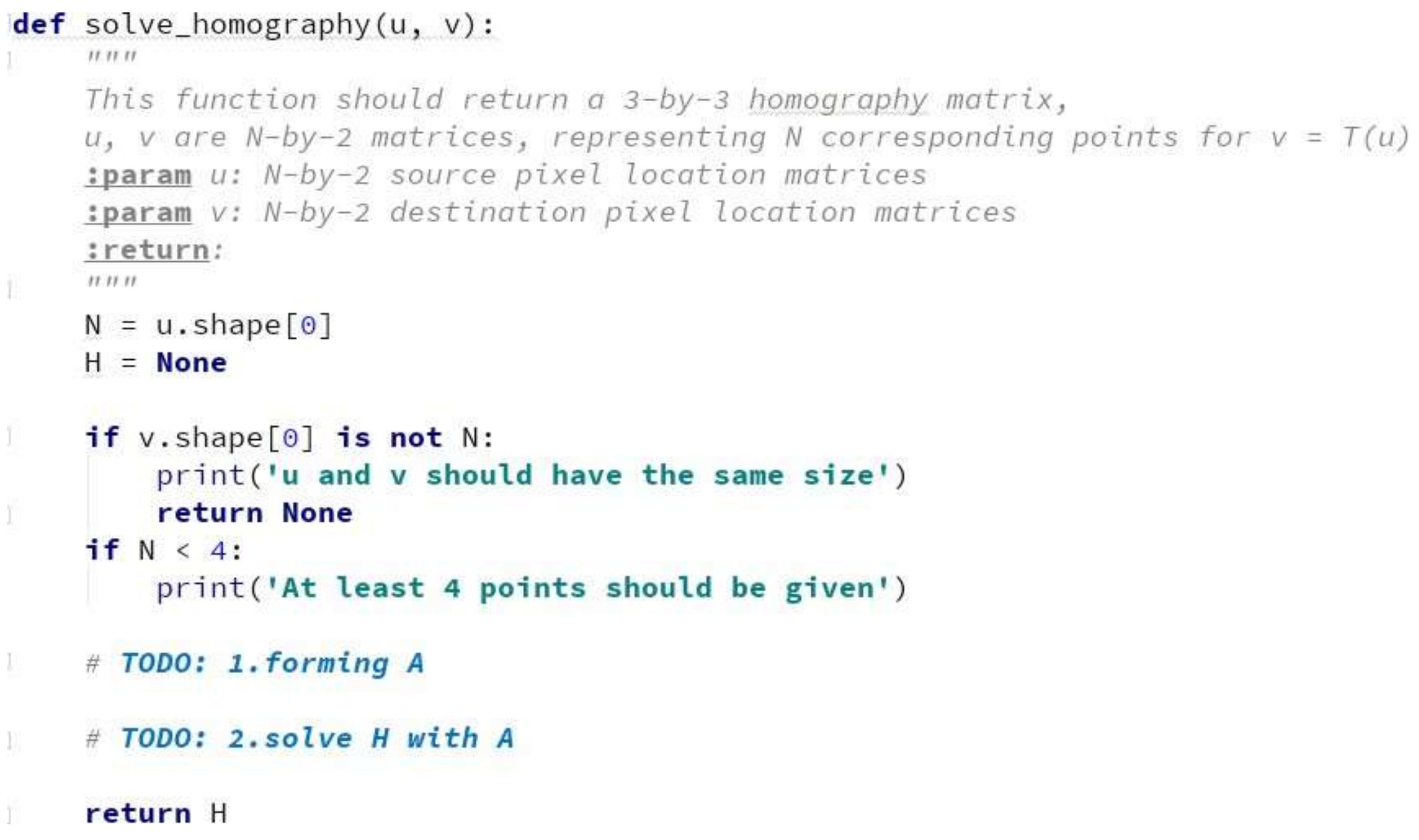
- hw3/
  - requirements.txt
  - report_template.docx
  - resource/
    - times.jpg, img1.jpg, img2.jpg, img3.jpg, img4.jpg, img5.jpg (part1)
    - hehe.jpg, seq0.mp4 (part2)
    - BL_secret1.png, BL_secret2.png (part3)
    - frame1.jpg, frame2.jpg, frame3.jpg (part4)
  - src/
    - part1.py
    - part2.py
    - part3.py
    - part4.py
    - utils.py
    - hw3.sh

utils.py



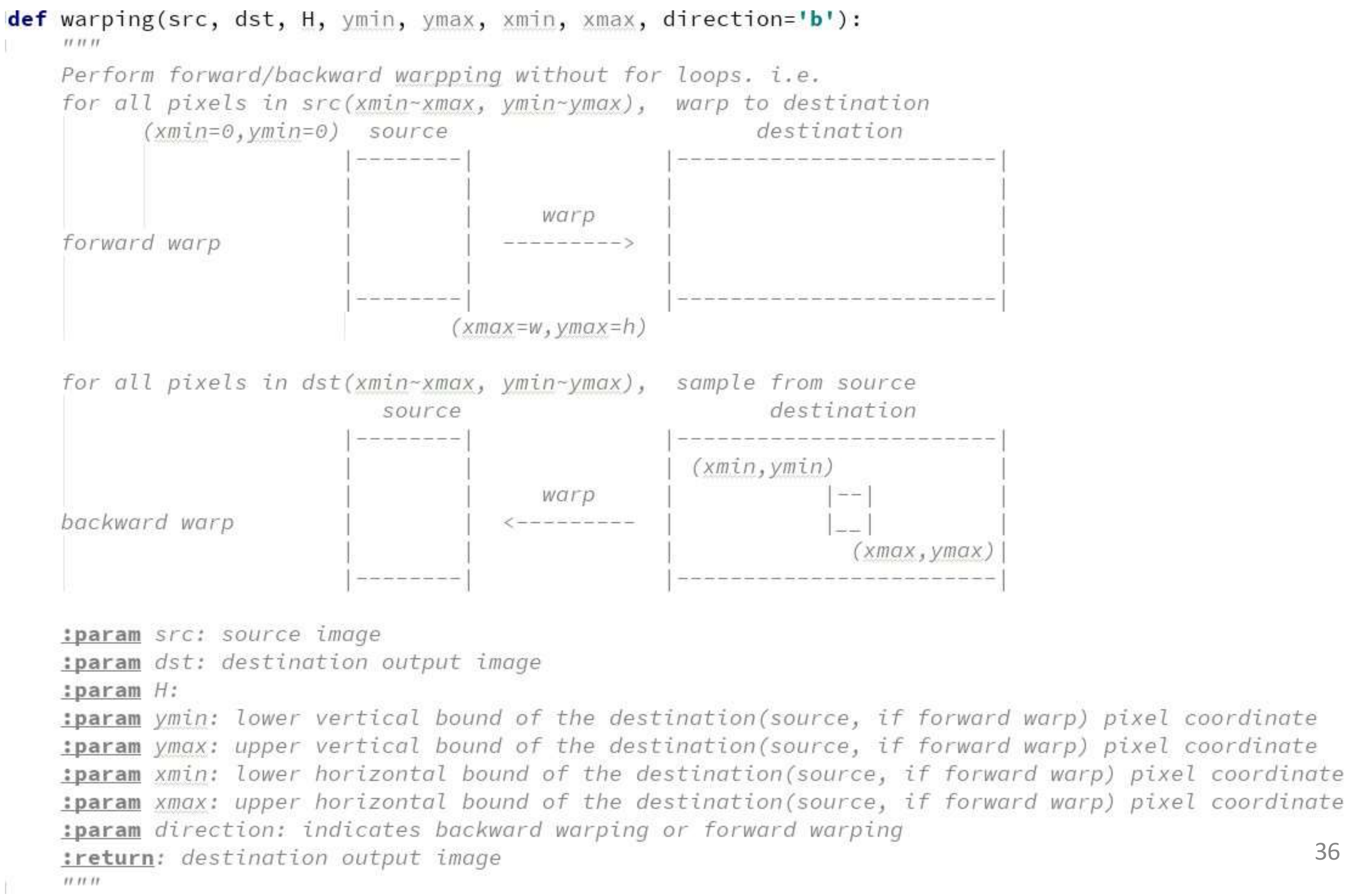
```python
import numpy as np


def solve_homography(u, v):...


def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='b'):...
```

solve_homography(u, v)



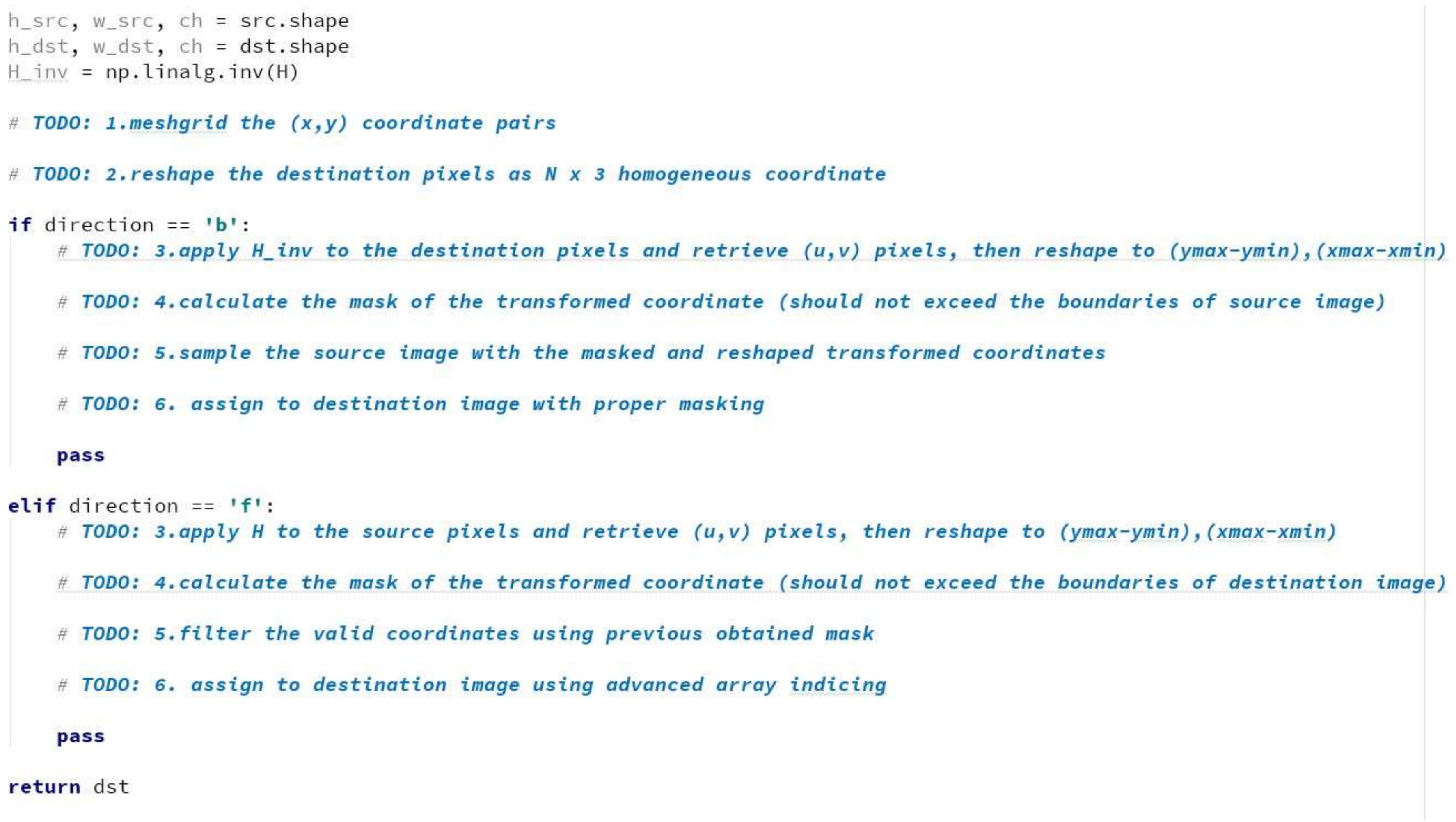
```python
def solve_homography(u, v):
    """
    This function should return a 3-by-3 homography matrix,
    u, v are N-by-2 matrices, representing N corresponding points for v = T(u)
    :param u: N-by-2 source pixel location matrices
    :param v: N-by-2 destination pixel location matrices
    :return:
    """
    N = u.shape[0]
    H = None

    if v.shape[0] is not N:
        print('u and v should have the same size')
        return None
    if N < 4:
        print('At least 4 points should be given')

    # TODO: 1.forming A

    # TODO: 2.solve H with A

    return H
```

warping( ) (1/2)



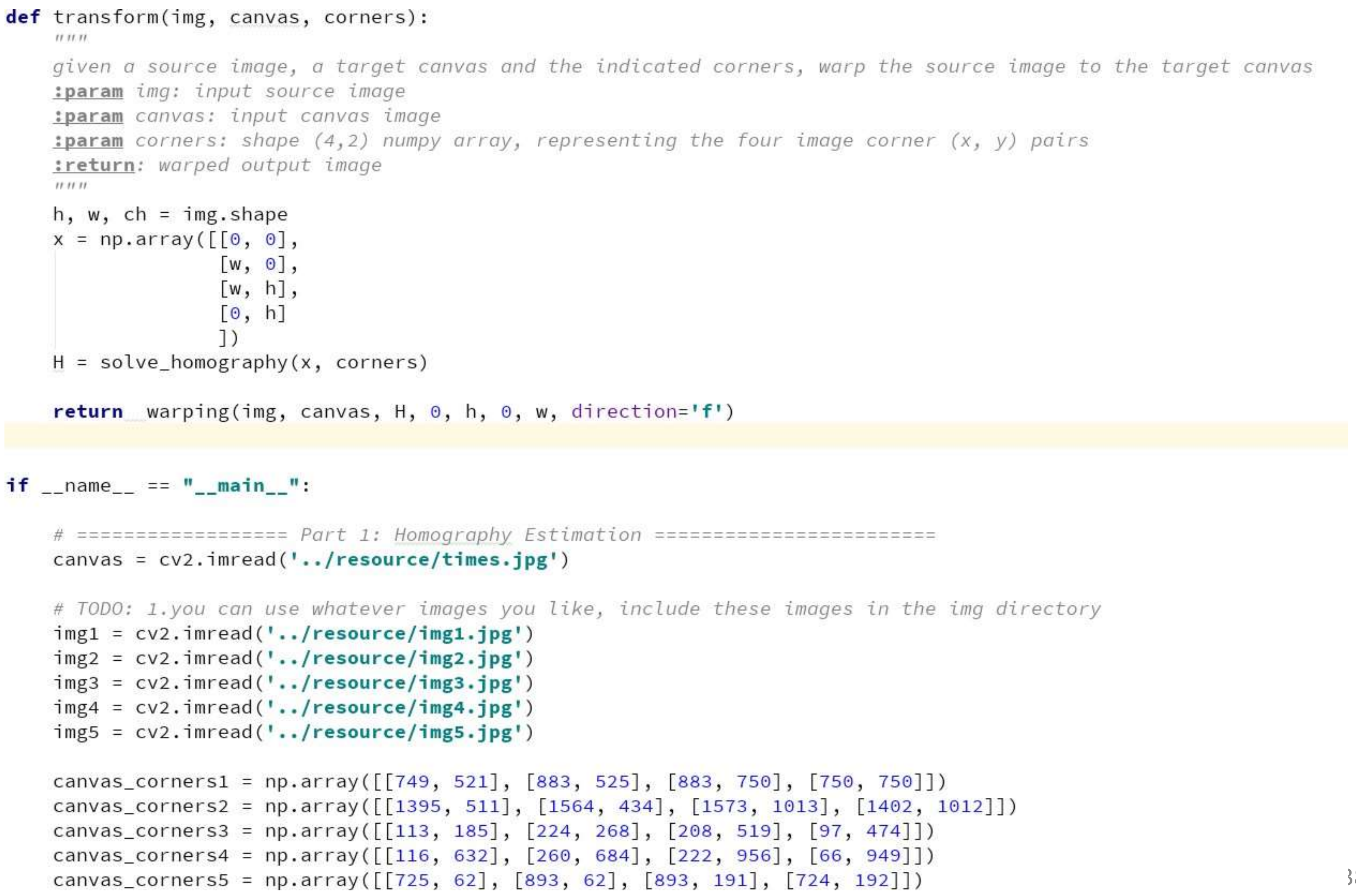
```
def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='b'):
    """
    Perform forward/backward warpping without for loops. i.e.
    for all pixels in src(xmin~xmax, ymin~ymax),  warp to destination
          (xmin=0,ymin=0)  source                       destination
                         |--------|               |------------------------|
                         |        |               |                        |
                         |        |     warp      |                        |
    forward warp         |        |  --------->   |                        |
                         |        |               |                        |
                         |--------|               |------------------------|
                                 (xmax=w,ymax=h)

    for all pixels in dst(xmin~xmax, ymin~ymax),  sample from source
                            source                        destination
                         |--------|               |------------------------|
                         |        |               | (xmin,ymin)            |
                         |        |     warp      |           |--|         |
    backward warp        |        |  <---------   |           |__|         |
                         |        |               |             (xmax,ymax)|
                         |--------|               |------------------------|

    :param src: source image
    :param dst: destination output image
    :param H:
    :param ymin: lower vertical bound of the destination(source, if forward warp) pixel coordinate
    :param ymax: upper vertical bound of the destination(source, if forward warp) pixel coordinate
    :param xmin: lower horizontal bound of the destination(source, if forward warp) pixel coordinate
    :param xmax: upper horizontal bound of the destination(source, if forward warp) pixel coordinate
    :param direction: indicates backward warping or forward warping
    :return: destination output image
    """
```

warping( ) (2/2)



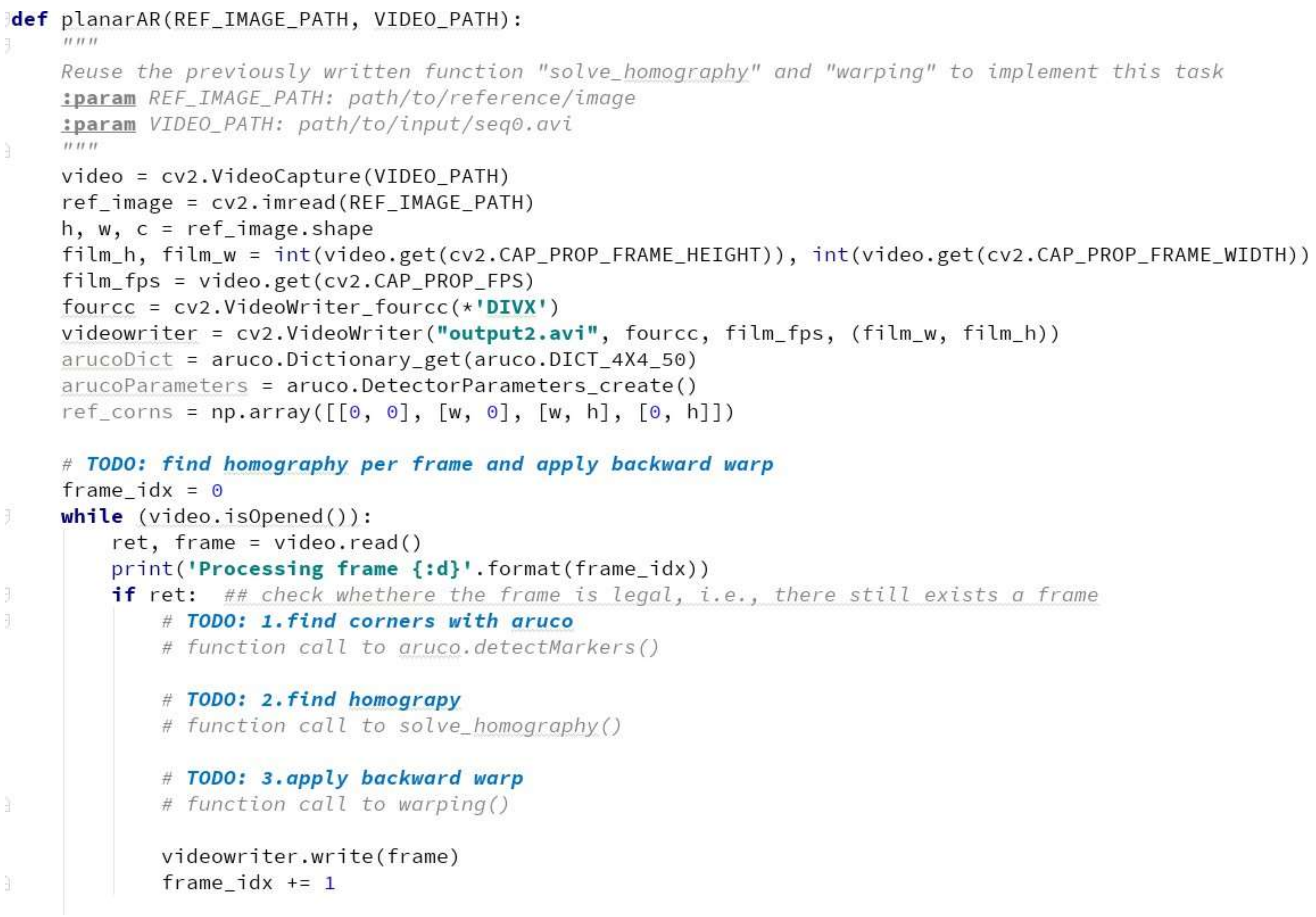
```
h_src, w_src, ch = src.shape
h_dst, w_dst, ch = dst.shape
H_inv = np.linalg.inv(H)

# TODO: 1.meshgrid the (x,y) coordinate pairs

# TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate

if direction == 'b':
    # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)

    # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)

    # TODO: 5.sample the source image with the masked and reshaped transformed coordinates

    # TODO: 6. assign to destination image with proper masking

    pass

elif direction == 'f':
    # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)

    # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)

    # TODO: 5.filter the valid coordinates using previous obtained mask

    # TODO: 6. assign to destination image using advanced array indicing

    pass

return dst
```

part1.py (you don’t need to code)



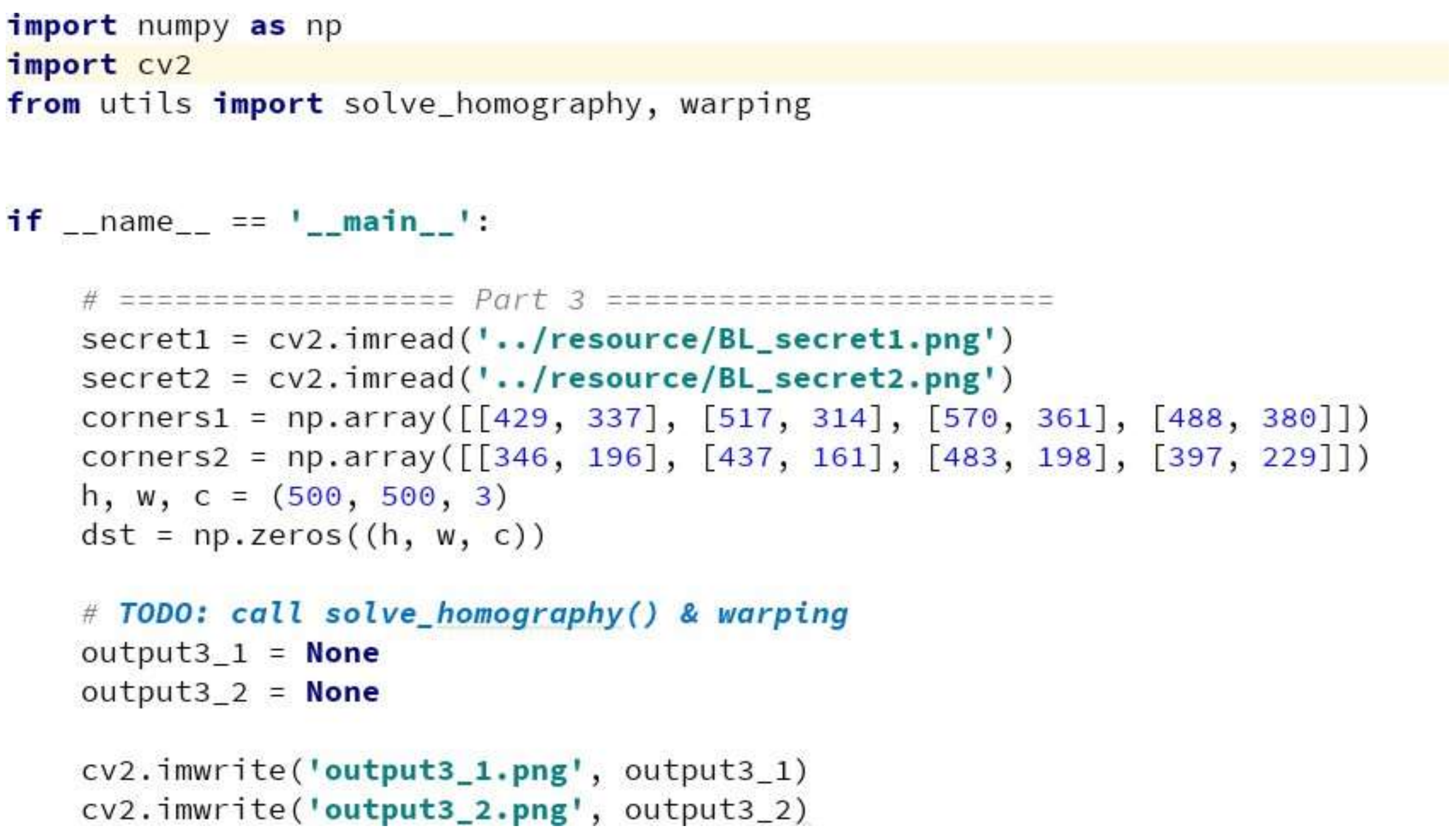
```python
def transform(img, canvas, corners):
    """
    given a source image, a target canvas and the indicated corners, warp the source image to the target canvas
    :param img: input source image
    :param canvas: input canvas image
    :param corners: shape (4,2) numpy array, representing the four image corner (x, y) pairs
    :return: warped output image
    """
    h, w, ch = img.shape
    x = np.array([[0, 0],
                  [w, 0],
                  [w, h],
                  [0, h]
                  ])
    H = solve_homography(x, corners)

    return  warping(img, canvas, H, 0, h, 0, w, direction='f')


if __name__ == "__main__":

    # ================== Part 1: Homography Estimation ========================
    canvas = cv2.imread('../resource/times.jpg')

    # TODO: 1.you can use whatever images you like, include these images in the img directory
    img1 = cv2.imread('../resource/img1.jpg')
    img2 = cv2.imread('../resource/img2.jpg')
    img3 = cv2.imread('../resource/img3.jpg')
    img4 = cv2.imread('../resource/img4.jpg')
    img5 = cv2.imread('../resource/img5.jpg')

    canvas_corners1 = np.array([[749, 521], [883, 525], [883, 750], [750, 750]])
    canvas_corners2 = np.array([[1395, 511], [1564, 434], [1573, 1013], [1402, 1012]])
    canvas_corners3 = np.array([[113, 185], [224, 268], [208, 519], [97, 474]])
    canvas_corners4 = np.array([[116, 632], [260, 684], [222, 956], [66, 949]])
    canvas_corners5 = np.array([[725, 62], [893, 62], [893, 191], [724, 192]])
```

part2.py



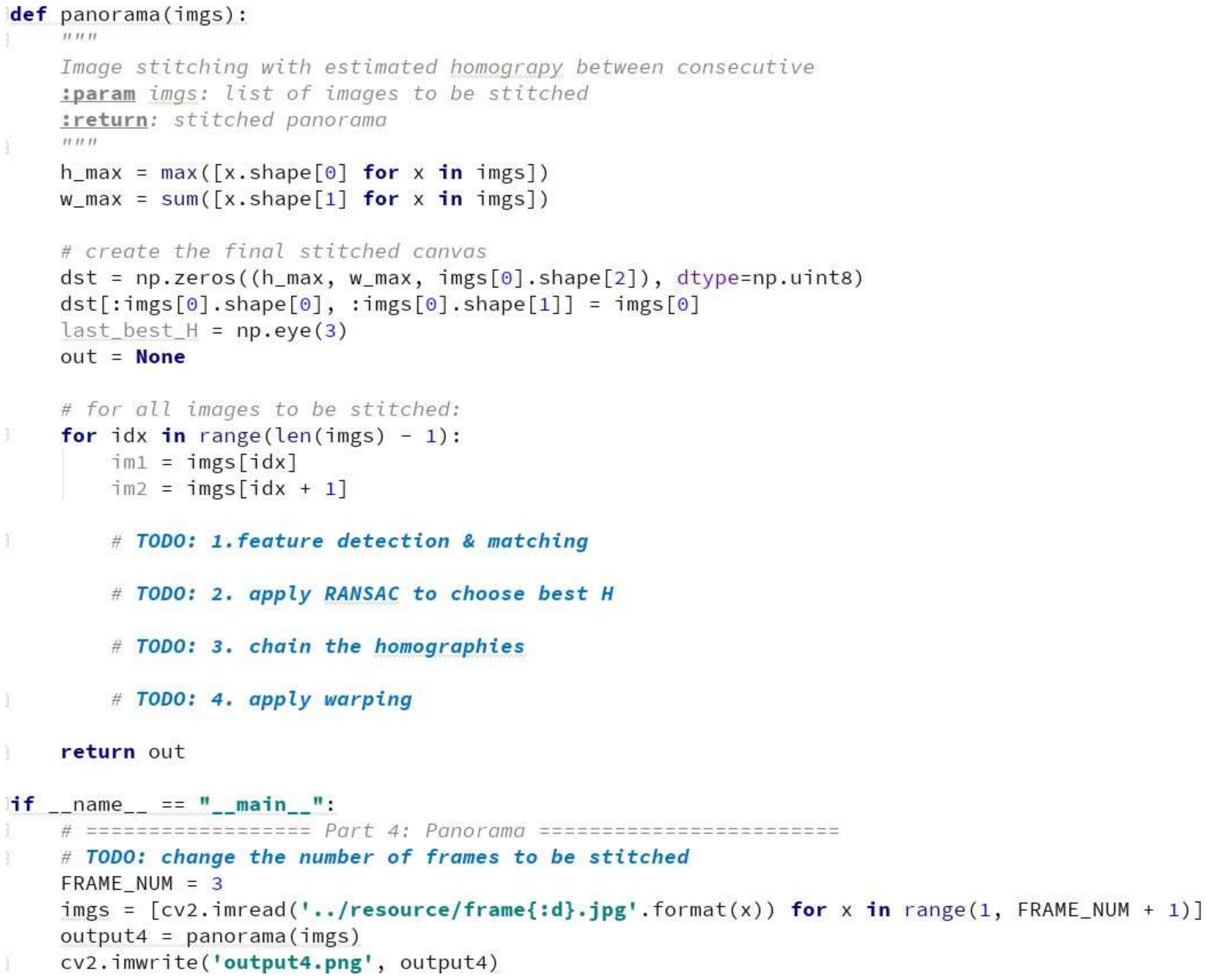
```python
def planarAR(REF_IMAGE_PATH, VIDEO_PATH):
    """
    Reuse the previously written function "solve_homography" and "warping" to implement this task
    :param REF_IMAGE_PATH: path/to/reference/image
    :param VIDEO_PATH: path/to/input/seq0.avi
    """
    video = cv2.VideoCapture(VIDEO_PATH)
    ref_image = cv2.imread(REF_IMAGE_PATH)
    h, w, c = ref_image.shape
    film_h, film_w = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT)), int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    film_fps = video.get(cv2.CAP_PROP_FPS)
    fourcc = cv2.VideoWriter_fourcc(*'DIVX')
    videowriter = cv2.VideoWriter("output2.avi", fourcc, film_fps, (film_w, film_h))
    arucoDict = aruco.Dictionary_get(aruco.DICT_4X4_50)
    arucoParameters = aruco.DetectorParameters_create()
    ref_corns = np.array([[0, 0], [w, 0], [w, h], [0, h]])

    # TODO: find homography per frame and apply backward warp
    frame_idx = 0
    while (video.isOpened()):
        ret, frame = video.read()
        print('Processing frame {:d}'.format(frame_idx))
        if ret:  ## check whethere the frame is legal, i.e., there still exists a frame
            # TODO: 1.find corners with aruco
            # function call to aruco.detectMarkers()

            # TODO: 2.find homograpy
            # function call to solve_homography()

            # TODO: 3.apply backward warp
            # function call to warping()

            videowriter.write(frame)
            frame_idx += 1
```

part3.py



```python
import numpy as np
import cv2
from utils import solve_homography, warping


if __name__ == '__main__':

    # ================== Part 3 ========================
    secret1 = cv2.imread('../resource/BL_secret1.png')
    secret2 = cv2.imread('../resource/BL_secret2.png')
    corners1 = np.array([[429, 337], [517, 314], [570, 361], [488, 380]])
    corners2 = np.array([[346, 196], [437, 161], [483, 198], [397, 229]])
    h, w, c = (500, 500, 3)
    dst = np.zeros((h, w, c))

    # TODO: call solve_homography() & warping
    output3_1 = None
    output3_2 = None

    cv2.imwrite('output3_1.png', output3_1)
    cv2.imwrite('output3_2.png', output3_2)
```

part4.py



```python
def panorama(imgs):
    """
    Image stitching with estimated homograpy between consecutive
    :param imgs: list of images to be stitched
    :return: stitched panorama
    """
    h_max = max([x.shape[0] for x in imgs])
    w_max = sum([x.shape[1] for x in imgs])

    # create the final stitched canvas
    dst = np.zeros((h_max, w_max, imgs[0].shape[2]), dtype=np.uint8)
    dst[:imgs[0].shape[0], :imgs[0].shape[1]] = imgs[0]
    last_best_H = np.eye(3)
    out = None

    # for all images to be stitched:
    for idx in range(len(imgs) - 1):
        im1 = imgs[idx]
        im2 = imgs[idx + 1]

        # TODO: 1.feature detection & matching

        # TODO: 2. apply RANSAC to choose best H

        # TODO: 3. chain the homographies

        # TODO: 4. apply warping

    return out

if __name__ == "__main__":
    # ================== Part 4: Panorama ========================
    # TODO: change the number of frames to be stitched
    FRAME_NUM = 3
    imgs = [cv2.imread('../resource/frame{:d}.jpg'.format(x)) for x in range(1, FRAME_NUM + 1)]
    output4 = panorama(imgs)
    cv2.imwrite('output4.png', output4)
```

Execution of hw3

- TAs will run your code in the following manner:
  - >cd /path/to/src

    source hw3.sh
    - Please ensure the relative paths to the resources are correct
  - You should generate the outputs @:
  - src/
    - (output1.png)
    - (output2.avi)
    - (output3_1.png, output3_2.png)
    - (output4.png)

Assignment Description

- Recommended steps
  - Implement function solve_homography( )
  - Implement function warping( )
    - ※ If you are not familiar with python, we suggest you to use for loop version first, then adjust to array version
    - First, implement direction='f' (forward warp)
    - Second, implement direction='b' (backward warp)
  - Use the above functions to deal with other tasks (marker-based AR, panorama)
  - If the timing does not pass, improve the speed of warping and RANSAC
- You can neglect the template code and TODOs as long as you can meet the rules for each part

Assignment Description

- DO NOT use cv2.findHomography, cv2.warpPerspective or cv2.remap. You should implement homography and warping by yourself!

Assignment Description

- Why limit the speed ?
  - If using naïve for loops, you'll get x60~x100 slower timing
- About the speed test
  - Intel Core i9-9820X @ 3.30GHz CPU
  - Reference time of TA code
    - Part1 ⇒ 0.974 sec
    - Part2 ⇒ 32.455 sec
    - Part3 ⇒ 0.398 sec
    - Part4(3 images) ⇒ 2.94 sec
  - Total 37 sec

Packages

- Python>=3.6
- numpy>=1.19.2
- opencv-python==4.5.1.48
- opencv-contrib-python==4.5.1.48
- tqdm>=4.60.0
- And other standard python packages
- You can use the provided library list
  - pip install –r requirements.txt
- E-mail or ask TA first if you want to import other packages.

Tips for Accelerating Python Code

- Reduce the usage of for-loop to enhance parallel processing
  - We do not use any for-loops for warping
  - We only use two for-loop in entire panorama
    - One for iterating all the images to be stitched
    - Another for RANSAC iteration
- Matrix multiplication for python

This can be generated efficiently via meshgrid & reshape

You can reshape the N points back to shape (h,w) if needed

Pseudo code

```
for $u_y$ in range(h):
  for $u_x$ in range(w):
    calulate $\underset{3\times 1}{\begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix}} \sim \underset{3\times 3}{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix}}$
```

Pseudo code (for N points)

Generate $\mathbf{u} \in \mathbb{R}^{3\times N} = \begin{bmatrix} u_{x1} & \cdots & u_{xN} \\ u_{y1} & \cdots & u_{yN} \\ 1 & & 1 \end{bmatrix}$

Calulate $\underset{3\times N}{\begin{bmatrix} v_{x1} & \cdots & v_{xN} \\ v_{y1} & \cdots & v_{yN} \\ 1 & & 1 \end{bmatrix}} \sim \underset{3\times 3}{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}} \underset{3\times N}{\begin{bmatrix} u_{x1} & \cdots & u_{xN} \\ u_{y1} & \cdots & u_{yN} \\ 1 & & 1 \end{bmatrix}}$

Tips for Accelerating Python Code

- Array masking

Pseudo code

```
for $u_y$ in range(h):
 for $u_x$ in range(w):
   calulate $\begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix}$
   if $(v_x, v_y)$ is out of image range:
     continue
   else:
     output($u_x$, $u_y$) = do something
```

Pseudo code

Generate $\mathbf{u} \in \mathbb{R}^{3\times \mathrm{N}} = \begin{bmatrix} u_{x1} & & u_{xN} \\ u_{y1} & \cdots & u_{yN} \\ 1 & & 1 \end{bmatrix}$

Calulate $\begin{bmatrix} v_{x1} & & v_{xN} \\ v_{y1} & \cdots & v_{yN} \\ 1 & & 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u_{x1} & & u_{xN} \\ u_{y1} & \cdots & u_{yN} \\ 1 & & 1 \end{bmatrix}$

mask = $\mathbf{v}$ (logical operation) image range
output = do something * mask
(with data only valid at the masked place)

Although it seems that the masking version requires for FLOPs (didn't skip any invalid point)
However, using accelerated array libraries instead of for loops speedup dramatically.

See Numpy array operations for details

Tips for Accelerating Python Code

- Advanced Array indicing

Pseudo code

```
for $u_y$ in range(ymin, ymax):
  for $u_x$ in range(xmin, xmax):
    $\mathbf{v}$ = do something to $(u_x, u_y)$
    output($u_x$, $u_y$) = $\mathbf{v}$
```

Pseudo code

Generate $\mathbf{u} \in \mathbb{R}^{3\times\mathrm{N}} = \begin{bmatrix} u_{xmin} & \cdots & u_{xmax} \\ u_{ymin} & \cdots & u_{ymax} \\ 1 & & 1 \end{bmatrix}$

Reshape $\mathbf{u} \in \mathbb{R}^{3\times\mathrm{N}} \Rightarrow \mathbf{u} \in \mathbb{R}^{(ymax-\mathrm{ymin})\times(xmax-\mathrm{xmin})\times 3}$

$\mathbf{v}$ = do something to $\mathbf{u}$

output[$ymin$: $ymax$, $xmin$: $xmax$] = $\mathbf{v}$

($\mathbf{v}$ is an array of same size $\mathbb{R}^{(ymax-\mathrm{ymin})\times(xmax-\mathrm{xmin})\times 3}$)

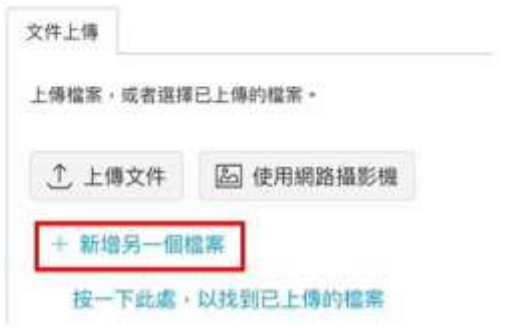
See Numpy array indexing for details

Report

- Your student ID, name
- Part1: Homography estimation
  - Paste your warped canvas
- Part2: Marker-Based Planar AR
  - Paste the function code *solve_homography(u, v)* & *warping( ) (*both forward & backward)
  - Briefly introduce the interpolation method you use
- Part3: Unwarp the secret
  - Paste the 2 warped images and the link you find
  - Discuss the difference between 2 source images, are the warped results the same or different?
  - If the results are the same, explain why.
  - If the results are different, explain why.
- Part4: Panorama
  - Paste your stitched panorama
  - Can all consecutive images be stitched into a panorama?
  - If yes, explain your reason. If not, explain under what conditions will result in a failure?

Submission (1/2)

- R07654321/
  - src/
    - part1.py, part2.py, part3.py, part4.py, utils.py, hw3.sh
    - ※ Please DO NOT upload the output files *.png *.avi (We will run and generate it by ourselves)
  - resource/
    - Images used in src/*.py
- Compress all above files in a zip file named StudentID.zip
  - e.g. R07654321.zip
  - After TAs run "unzip R07654321.zip", it should generate one directory named "R07654321", i.e.
    - R07654321/
      - src/
      - resource/
  - If any of the file format is wrong, you will receive a 30% penalty.



Submission (2/2)

- Submit StudentID.zip and report.pdf to NTU COOL
- Deadline: TBD
  - Late policy: see HW1
- Note that you should NOT hard code any path in your file or script
  - Use relative path instead!
  - If we can not execute your code, you'll get 0 point. But you have a chance to modify your code(see next page).
- Your code has to be finished in 10 mins.
  - Otherwise, you'll get zero point.
- Plagiarism is forbidden!
- We will execute you code on Linux system, so try to make sure your code can be executed on Linux system before submitting your homework.

Penalty

- If we can not execute your code, we will give you a chance to make minor modifications to your code. After you modify your code,
    - If we can execute your code and reproduce your results, you will still receive a 30% penalty in your homework score.
    - If we still cannot execute your code, you will get 0 in this problem.

Grading (Total 15 points)

Code: 80%

- Part 1 : 20%
  - 20%, generate correctly forward warped canvas
  - 0%, others
- Part 2 Code: 20%
  - 20% run without artifact
  - 0%, others
- Part 3 Code: 10%
  - 10%, generate warped 2 images
  - 0%, others
- Part 4 Code: 30%
  - 30%, generate panorama with no artifact (excluding blending artifact)
  - 20%, generate panorama little artifact (e.g. little discontinuity)
  - 0%, others

Grading (Total 15 points)

Report : 20%

- Part 1 (1%)
  - Paste your warped canvas (1%)
- Part 2 (3%)
  - Paste the function solve_homography( ) (1%)
  - Paste the function code *warping( )* (both forward & backward) (1%)
  - Briefly introduce the interpolation method you use (1%)
- Part 3 (8%)
  - Paste the 2 warped QR code and the link you find (1%)
  - Discuss the difference between 2 source images, are the warped results the same or different? (3%)
  - If the results are the same, explain why. If the results are different, explain why. (4%)
- Part 4 (8%)
  - Paste your stitched panorama (1%)
  - Can all consecutive images be stitched into a panorama? (3%)
  - If yes, explain your reason. If not, explain under what conditions will result in a failure? (4%)

If there are still other problems…

- Use NTU COOL Discussion (strongly recommended)
  - TA will answer the questions ASAP and record some common problems you may face.
  - https://cool.ntu.edu.tw/courses/38068/discussion_topics/272963
- E-mail TA
- Google it!

TA information

- Yu-Lun Chiang (江宇倫)
  E-mail: ylchiang@media.ee.ntu.edu.tw
  TA time: Email TA
  Location: BL-421

- Yu-Ching Fan (范宇清)
  E-mail: jackmafan@media.ee.ntu.edu.tw
  TA time: Email TA
  Location: BL-421