FALL2021 CSE473/573 PROJECT#2 TUTORIAL

TA: Zhanghexuan Ji

DATE: 11/18/2021

Please read the instruction '*project2.pdf*' carefully!
Check the APIs and functions that are allowed/excluded!

Task 1:

- Any Python Standard Library is allowed.
- Any APIs provided by Numpy is allowed.
- You can use OpenCV **EXCEPT FOR** *cv2.findHomography()* and APIs that have 'stitch', 'Stitch', 'match' or 'Match' in their names or functionality.
- Please include a pdf report for task1 showing the methods and steps you implemented, also include the OpenCV version you used in your report so that we can run your code.

Task 2:

- Any Python Standard Library is allowed.
- You can use Numpy **EXCEPT FOR** any function/operation directly related to convolution or correlation.
- You are **NOT** allowed to use OpenCV or any other external libraries.

Task 3:

- Any Python Standard Library is allowed.
- You can use Numpy **EXCEPT FOR** any function/API related to morphology operations.
- You are **NOT** allowed to use OpenCV or any other external libraries.

Please do not ask TAs to check whether your methods or results are correct or not. No TA is allowed to answer such question.

When grading, we will run your codes and check your saved .jpg image results. Therefore, all the returned images must be **uint8 or int** dtype, so that they can be saved as '.jpg' correctly.

When grading, we will test your code on other similar images, thus, do **NOT 'hard-code'** any functions/variables related to the input images.

TASK 1 – Image Stitching

1. Extract Keypoint | 2. Match Keypoint | 3. Estimate Homography | 4. Warp & Stitch Images

Two steps: keypoint detection + feature extraction (description).

- If your keypoint detector and descriptor are different, e.g. SIFT detector + SURF descriptor, then you need to:
  1. Create keypoint detector and .detect(img, …) the keypoints,

     e.g. sift = cv2.xfeatures2d.SIFT_create(); kp = sift.detect(img, None);
  2. Create keypoint descriptor and .compute(img, kp) the feature for each keypoint,

     e.g. surf = cv2.xfeatures2d.SURF_create(); kp, des = surf.compute(img, kp);
- If you use same detector and descriptor method, e.g. SIFT detector + SIFT descriptor, then OpenCV provides .detectAndCompute(img, …) to do them both in the same time.

  e.g. kp, des = sift.detectAndCompute(img, None);
- Read the OpenCV document for details.

  https://docs.opencv.org/3.4.2/db/d27/tutorial_py_table_of_contents_feature2d.html

(NOTE: all examples in this tutorial is based on 'opencv-contrib-python=3.4.2.17'.

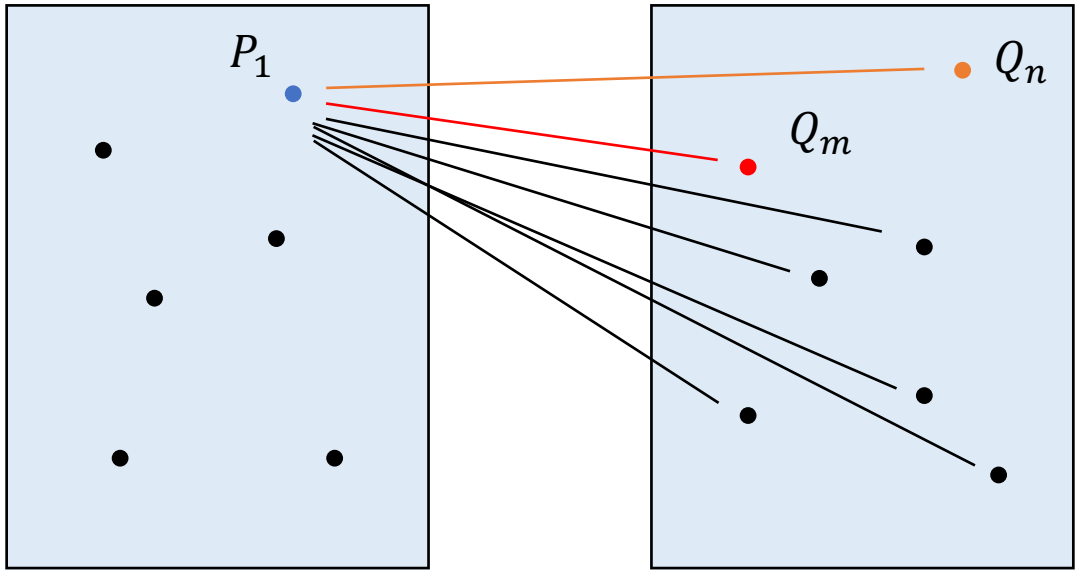
If you are using different version of OpenCV, please check the corresponding documents.)

TASK 1 – Image Stitching

1. Extract Keypoint | 2. Match Keypoint | 3. Estimate Homography | 4. Warp & Stitch Images

Use KNN with k=2 for keypoint matching.

- For each keypoint P_i in one image, calculate and rank the feature distances (2-norm distance) between P_i and every keypoint in the other image.
- Return top 2 best matching keypoints Q_m, Q_n



$$\begin{bmatrix} d(P_1,Q_1) \\ d(P_1,Q_2) \\ d(P_1,Q_3) \\ d(P_1,Q_4) \\ d(P_1,Q_5) \\ d(P_1,Q_6) \\ d(P_1,Q_7) \end{bmatrix} \xrightarrow{\text{rank}} \begin{bmatrix} d(P_1,Q_m) \\ d(P_1,Q_n) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

Get the top 2 best matching keypoints and save them for future matching.

(P_1, Q_m, Q_n)

TASK 1 – Image Stitching

1. Extract Keypoint → 2. Match Keypoint → 3. Estimate Homography → 4. Warp & Stitch Images

Important: matrix calculation in numpy is faster than 'for loop'!

e.g. Given a vector x and a list of vectors y_list, where all the vectors have the same length as x. What is the smallest 1-norm distance between x and the vectors in y_list?

```
Given vector x and a list of vectors y_list
least_distance = inf
For y in y_list:
    y_distance = sum(x-y)
    If y_distance < least_distance:
        least_distance = y_distance
    End if
End for
```

```
Given vector x and a list of vectors y_list
x_vector = np.asarray(x)
y_matrix = np.asarray(y_list)
distance_list = np.sum(x_vector – y_matrix, axis=1)
Rank the distance_list
Least_distance = distance_list[0]
```

Which one is faster?

TASK 1 – Image Stitching

1. Extract Keypoint | 2. Match Keypoint | 3. Estimate Homography | 4. Warp & Stitch Images

Too many matching candidates from KNN, which also contains ambiguous matches.

Need filtering the matching candidates!

Cross check: a good match should be symmetric.

- For a pair of keypoint (P_i, Q_j) to considered valid, P_i in left image needs to match Q_j in right image, and Q_j in right image should match P_i as the closest match as well.
- Match twice: you need to match keypoints in left image to all the keypoints in right image, and symmetrically match keypoints in right image to all the keypoints in left image.
- If P_1 matches Q_2 from left to right, then Q_2 should also match P_1 from right to left.

$$\begin{bmatrix} d(P_1,Q_1) \\ d(P_1,Q_2) \\ d(P_1,Q_3) \\ d(P_1,Q_4) \\ d(P_1,Q_5) \\ d(P_1,Q_6) \\ d(P_1,Q_7) \end{bmatrix} \xrightarrow{\text{rank}} \begin{bmatrix} d(P_1,Q_2) \\ \vdots \\ . \\ . \\ . \\ . \\ . \end{bmatrix}$$

Match P_1 from left to right, Q_2 is the best match.

$$\begin{bmatrix} d(Q_2,P_1) \\ d(Q_2,P_2) \\ d(Q_2,P_3) \\ d(Q_2,P_4) \\ d(Q_2,P_5) \\ d(Q_2,P_6) \\ d(Q_2,P_7) \end{bmatrix} \xrightarrow{\text{rank}} \begin{bmatrix} d(Q_2,P_m) \\ \vdots \\ . \\ . \\ . \\ . \\ . \end{bmatrix}$$

Match Q_2 from right to left, P_m is the best match.

- If $P_m = P_1$, then (P_1, Q_2) is a valid match;
- If $P_m \neq P_1$, then (P_1, Q_2) is an ambiguous match and should be removed from matching candidates.

TASK 1 – Image Stitching

1. Extract Keypoint | 2. Match Keypoint | 3. Estimate Homography | 4. Warp & Stitch Images

Too many matching candidates from KNN, which also contains ambiguous matches.

Need filtering the matching candidates!

Ratio test: the top 2 best matches should not be ambiguous.

- From KNN with k=2, each keypoint P_i has top 2 best matches (Q_m, Q_n).
- If the distance between (P_i, Q_m) is much smaller than the distance between (P_i, Q_n) by a certain ratio η, then (P_i, Q_m) should be a good match; otherwise, the best match Q_m is very similar to the 2nd best match Q_n, and (P_i, Q_m) is ambiguous and should be removed from matching candidates.
- Check whether $d(P_i, Q_m) < \eta \cdot d(P_i, Q_n)$; True → good match; False → ambiguous match.
- You can try and adjust the ratio $\eta \in (0,1)$. Smaller η filters out more matching candidates. You can start from $\eta = 0.75$.

If you still think there are too many candidates after filtering, you can even rank the ratio distances of all matching candidates and only keep the top 200-300 matching pairs. (this may reduce the stitching performance)

TASK 1 – Image Stitching

1. Extract Keypoint | 2. Match Keypoint | 3. Estimate Homography | 4. Warp & Stitch Images

2D homography matrix H: mapping between two projection planes with the same center of projection.

$$p' = Hp$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The degree of freedom of H is 8, since scaling H does not change the the projection results.

$$\lambda w \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left(\lambda \cdot \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Thus, you can normalize H by any non-zero element.

In OpenCV, H is normalized by h_{22} so that the last element of H is 1.0.

(NOTE: in some special cases, h_{22} can be 0)

TASK 1 – Image Stitching

1. Extract Keypoint → 2. Match Keypoint → 3. Estimate Homography → 4. Warp & Stitch Images

Solving for homography matrix H

We already get enough matching candidates from step2, which can be used to estimate homography by solving a least squares problem. Recall Project1 – Task2:

Preliminary.2.
Let $X_wY_wZ_w$ be the world coordinate and xy be the image coordinate, we have the transformation matrix $M \in \mathbb{R}^{3\times4}$:

$$s\begin{bmatrix}x\\y\\1\end{bmatrix} = \begin{bmatrix}m_{11} & m_{12} & m_{13} & m_{14}\\ m_{21} & m_{22} & m_{23} & m_{24}\\ m_{31} & m_{32} & m_{33} & m_{34}\end{bmatrix} \cdot \begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix} \tag{6}$$

$$\begin{aligned} sx &= m_{11}X_w + m_{12}Y_w + m_{13}Z_w + m_{14},\\ sy &= m_{21}X_w + m_{22}Y_w + m_{23}Z_w + m_{24},\\ s &= m_{31}X_w + m_{32}Y_w + m_{33}Z_w + m_{34}. \end{aligned} \tag{7}$$

We can solve m_{ij} with the equation below:

$$\begin{bmatrix} X_w^1 & Y_w^1 & Z_w^1 & 1 & 0 & 0 & 0 & 0 & -x^1X_w^1 & -x^1Y_w^1 & -x^1Z_w^1 & -x^1\\ 0 & 0 & 0 & 0 & X_w^1 & Y_w^1 & Z_w^1 & 1 & -y^1X_w^1 & -y^1Y_w^1 & -y^1Z_w^1 & -y^1\\ & & & & & & \vdots & & & & & \\ X_w^n & Y_w^n & Z_w^n & 1 & 0 & 0 & 0 & 0 & -x^nX_w^n & -x^nY_w^n & -x^nZ_w^n & -x^n\\ 0 & 0 & 0 & 0 & X_w^n & Y_w^n & Z_w^n & 1 & -y^nX_w^n & -y^nY_w^n & -y^nZ_w^n & -y^n \end{bmatrix} \cdot \begin{bmatrix}m_{11}\\m_{12}\\m_{13}\\m_{14}\\m_{21}\\m_{22}\\m_{23}\\m_{24}\\m_{31}\\m_{32}\\m_{33}\\m_{34}\end{bmatrix} = \mathbf{0}, \tag{8}$$

where the first matrix is with size $2n \times 12$ (n is the number of available points).

Preliminary.3.
Solve the homogeneous linear equation $\mathbf{Ax} = 0$, where $\mathbf{x}$ is the vector of N unknowns, and $\mathbf{A}$ is the matrix of $M \times N$ coefficients. A quick observation is that there are infinite solutions for $\mathbf{Ax} = 0$, since we can randomly scale x with a scalar λ such that $\mathbf{A}(\lambda\mathbf{x}) = 0$. Therefore, we assume $\|\mathbf{x}\| = 1$. Solving the equation can be converted to

$$\min \|\mathbf{Ax}\| \tag{9}$$

The minimization problem can be solved with Singular Value Decomposition (SVD). Assume that $\mathbf{A}$ can be decomposed to $\mathbf{U\Sigma V}^T$, we have

$$\min \|\mathbf{Ax}\| = \|\mathbf{U\Sigma V}^T\mathbf{x}\| = \|\mathbf{\Sigma V}^T\mathbf{x}\|. \tag{10}$$

Note that $\|\mathbf{V}^T\mathbf{x}\| = \|\mathbf{x}\| = 1$, then let $\mathbf{y} = \mathbf{V}^T\mathbf{x}$, so we have

$$\begin{aligned} \min \|\mathbf{Ax}\| &= \|\mathbf{\Sigma y}\|\\ &= \left\| \begin{bmatrix}\sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ 0 & \cdots & 0 \\ \vdots & \vdots & \vdots \\ 0 & \cdots & 0\end{bmatrix} \begin{bmatrix}y_1\\\vdots\\y_n\end{bmatrix} \right\|, \end{aligned} \tag{11}$$

where $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$. Recall that $\|\mathbf{y}\| = 1$, we can set

$$\begin{bmatrix}y_1\\\vdots\\y_n\end{bmatrix} = \begin{bmatrix}0\\\vdots\\1\end{bmatrix}. \tag{12}$$

So $\mathbf{x}$ should be the last row of $\mathbf{V}^T$.

TASK 1 – Image Stitching

1. Extract Keypoint → 2. Match Keypoint → 3. Estimate Homography → 4. Warp & Stitch Images

Solving for homography matrix H

$$\begin{bmatrix} wx_i' \\ wy_i' \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x_i' = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \qquad y_i' = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$h_{00}x_i + h_{01}y_i + h_{02} - h_{20}x_i'x_i - h_{21}x_i'y_i - h_{22}x_i' = 0$$

$$h_{10}x_i + h_{11}y_i + h_{12} - h_{20}y_i'x_i - h_{21}y_i'y_i - h_{22}y_i' = 0$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

A (2nx9) **h** (9x1) **0** (2nx1)

Minimize $\|Ah - 0\|^2$, can be solved with SVD

$$U\Sigma V^T = SVD(A)$$

Solution: $\hat{h}$ should be the last row of V^T

In order to solve H, at least 4 keypoint matching pairs are needed → $n = 4$ in RANSAC.

TASK 1 – Image Stitching

1. Extract Keypoint → 2. Match Keypoint → 3. Estimate Homography → 4. Warp & Stitch Images

RANSAC:

Determine:
  n—the smallest number of points required (e.g., for lines, $n = 2$,
    for circles, $n = 3$)
  k—the number of iterations required
  t—the threshold used to identify a point that fits well
  d—the number of nearby points required
    to assert a model fits well
Until k iterations have occurred
  Draw a sample of n points from the data
    uniformly and at random
  Fit to that set of n points
  For each data point outside the sample
    Test the distance from the point to the structure
      against t; if the distance from the point to the structure
      is less than t, the point is close
  end
  If there are d or more points close to the structure
    then there is a good fit. Refit the structure using all
    these points. Add the result to a collection of good fits.
end
Use the best fit from this collection, using the
  fitting error as a criterion

Algorithm 10.4: RANSAC: Fitting Structures Using Random Sample Consensus.

Computer Vision – A Modern Approach, 2ed, D. Forsyth, J. Ponce.

- Set $n = 4$ to estimate H
- Set k for the largest number of iteration (e.g. k=5000)
- Set threshold t to identify inliers/outliers (e.g. t=5)
- Fit $\widehat{H}$ using the random sampled 4 keypoint pairs
- Test the residual distance on keypoint pairs outside the above samples against t and count inliers
- After RANSAC loops, re-estimate final H using all the inlier keypoint matching pairs

TASK 1 – Image Stitching

1. Extract Keypoint → 2. Match Keypoint → 3. Estimate Homography → 4. Warp & Stitch Images

RANSAC:

An easier way to implement RANSAC
(faster, may not be as good as the one from D. Forsyth & J. Ponce book)

Given m keypoint matching pairs $\{(p'_i, p_i)\}_{i=1}^{m}$

Until k iterations have occurred:

- Sample 4 keypoint pairs $\{(p'_j, p_j)\}_{j\in\{1,\dots,m\}}^{4}$ at random
- Solve homography $\hat{H}$ using least-squares or SVD
- Compute $\hat{H}p_i$ (normalize the result coordinates by w; p_i are right-keypoints outside the above 4 samples)
- Count inliers where $\|p'_i - \hat{H}p_i\| < t$ (test residual distance)
- Keep the largest set of inliers S (update if needed)

Re-compute final H using all of the inliers in S via least-squares fitting or SVD.

Since $\hat{H}p_i = w\hat{p}'_i$, in order to get the correct predicted keypoint coordinates $\hat{p}'_i = (\hat{x}'_i, \hat{y}'_i)$, you need to normalize the results from $\hat{H}p_i$ by the last element w before testing residual.

TASK 1 – Image Stitching

1. Extract Keypoint → 2. Match Keypoint → 3. Estimate Homography → 4. Warp & Stitch Images

- Use the homography matrix to warp one image to another and stitch the two given images into a single panorama.
- OpenCV provides some useful APIs for warping, e.g. cv2.perspectiveTransform() and cv2.warpPerspective().
- Your final panorama image should NOT be cropped.
- Calculate the maximum/minimum x and y coordinates after homography tranformation (pay attention to the 8 corners of the two images).

Expected panorama

✔

Warp and stitching the left image to the right image is also accepted.

Cropped panorama

✗

✗

TASK 1 – Image Stitching

1. Extract Keypoint > 2. Match Keypoint > 3. Estimate Homography > 4. Warp & Stitch Images

Optional Reading (not required in this project)

How to make a perfect panorama without any seam or dividing line.

- Image blendering;
- Seamless (seam-driven) image stitching;
- Cylindrical projection;
- ……

You can get more information from OpenCV library.

TASK 2 – Grayscale Image Processing

25	111	12
210	0	152
175	58	90

→ 90

1. Denoise → 2. Edge Detect → 3. Diagonal Edge

- Complete *filter(img)* in *task2.py*.
- Apply 3 x 3 Median Filter to remove salt-and-pepper noise: read slides "8_Nonlinear_Image_Smoothing.pdf".
- Use zero-padding. Your denoised image should keep the same size as your input.

Salt-and-pepper noise image

Median Filter →

Denoised image

TASK 2 – Grayscale Image Processing

1. Denoise | 2. Edge Detect | 3. Diagonal Edge

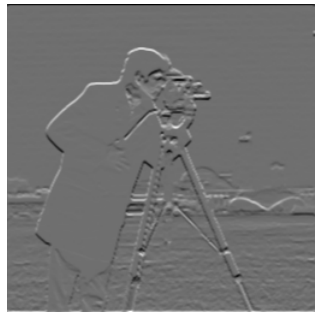
- Implement *convolve2d(img, kernel)* first, remember to flip the kernel.
- Complete *edge_detect(denoised_img)*, Sobel x/y operators are given in *task2.py*.
- Zero-padding. Your edge images should keep the same size as your input.
- Normalize your edge images before returning them: $img\_norm = 255 \times \frac{img - \min(img)}{\max(img) - \min(img)}$
- Images dtype must be uint8 or int dtype, in order to be saved as '.jpg' correctly.

Denoised image

Conv Sobel_x

Conv Sobel_y

Edge X



Edge Y

Edge magnitude

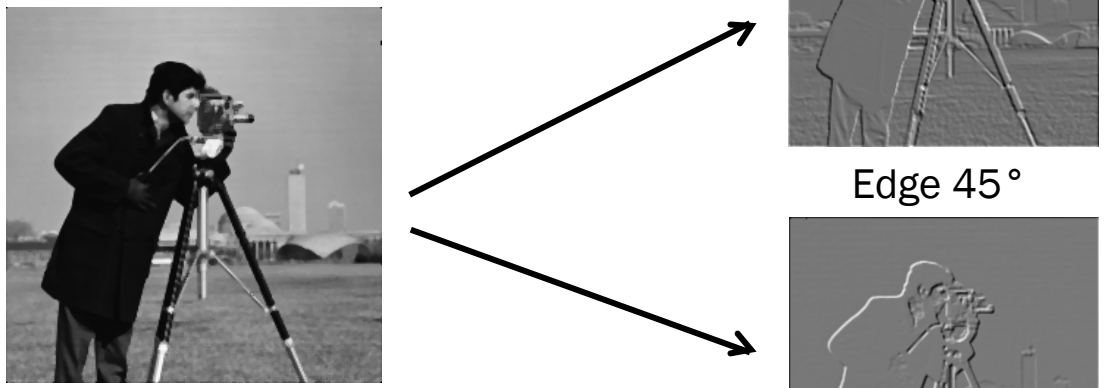
$$\sqrt{(edge\_x)^2 + (edge\_y)^2}$$

NOTE: do NOT use normalized edge_x and edge_y for magnitude calculation.

TASK 2 – Grayscale Image Processing

1. Denoise → 2. Edge Detect → 3. Diagonal Edge

- Design two 3x3 kernels to detect the edges along 45° and 135° directions (x direction as 0°).
- Complete *edge_diag(denoised_img)*, print out the kernels you designed.
- Zero-padding. Your edge images should keep the same size as your input.
- Normalize your edge images before returning them: $img\_norm = 255 \times \frac{img - \min(img)}{\max(img) - \min(img)}$
- Images dtype must be uint8 or int dtype, in order to be saved as '.jpg' correctly.



Denoised image

Edge 45°

Edge 135°

These are just sample results.

Your results will depend on the kernels you designed.

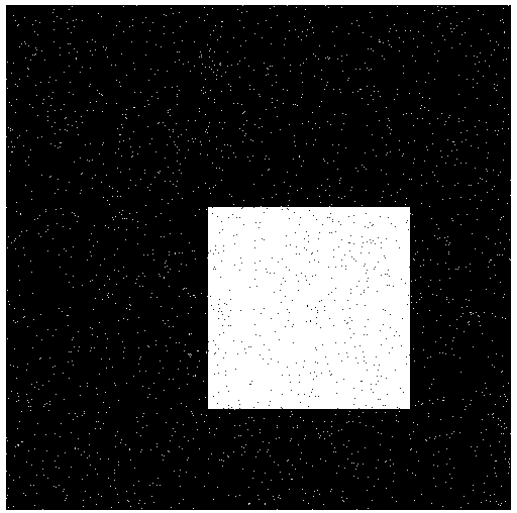
TASK 3 – Morphology Image Processing

1. Morphological Denoise
2. Morphological Boundary Detect

- Read slides “9_Morphological_Image_Processing.pdf”.
- Use 3x3 squared structuring element of all 1’s for all operations.
- First implement *morph_erode(img)*, *morph_dilate(img)*.
- Then implement *morph_open(img)*, *morph_close(img)* using erode and dilate operations.
- Complete *denoise(img)* in *task3.py* using the above functions.
- Zero-padding. Your denoised image should keep the same size as your input.

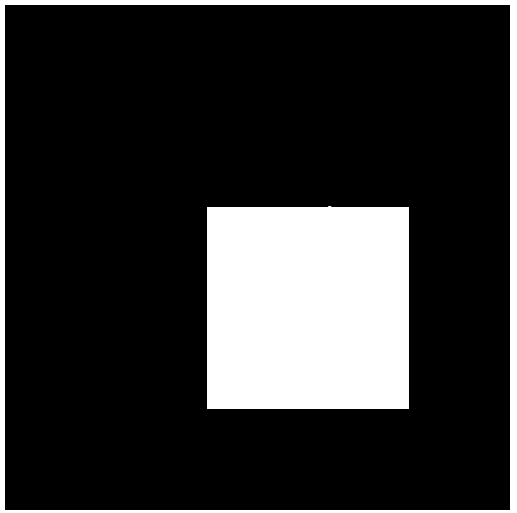
1	1	1
1	1	1
1	1	1

Structuring Element



Binary image with noise

Morphological Filtering



Denoised binary image

TASK 3 – Morphology Image Processing

1. Morphological Denoise
2. Morphological Boundary Detect

Decide what image dtype to use in morphological (logical) operations.

- Binary images are read as *uint8*, and only have 0 and 255 values.
- One possible solution: keep using *uint8* in morphological operations, and treat 0 as False and 255 as True.
- A better solution: convert the input image to *bool* image in *denoise(img)*, so that 0→False and 1→True. Then you can use logical operations for morphology.
- Remember to convert the images dtype back to uint8/int and scale the value back as False→0 and True→255 before returning them, in order to be saved as '.jpg' correctly.

0	0	0	0
0	255	255	0
0	255	255	0
255	255	255	255

uint8 image matrix

Convert dtype from uint8 to bool

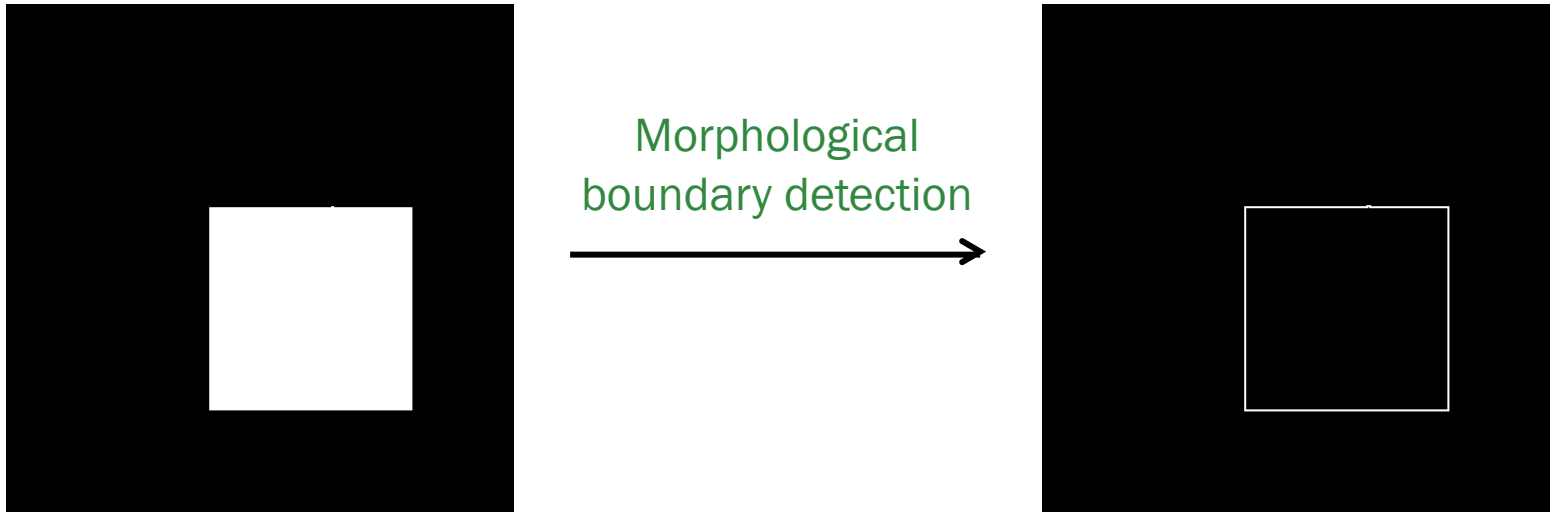
F	F	F	F
F	T	T	F
F	T	T	F
T	T	T	T

bool image matrix

TASK 3 – Morphology Image Processing

1. Morphological Denoise
2. Morphological Boundary Detect

- Complete *boundary(denoised_img)* in *task3.py* based on the four morphological operations, i.e. erode, dilate, open and close.
- Zero-padding. Your denoised image should keep the same size as your input.
- Images dtype must be uint8 or int dtype and scaled to 0/255, in order to be saved as '.jpg' correctly.



Denoised binary image

Boundary image

Q & A