HOMEWORK ASSIGNMENT #2
Edge Detection and Geometrical Modification

Poy Lu 呂栢頤
D09944015
網媒所博一
ariapoy@gmail.com

April 9, 2021

1 Problem 1: EDGE DETECTION

In this problem, please design several edge detection algorithms to satisfy the following requirements.

1.1 (a)

Given an image, **sample1.jpg**.

Original image **sample1.jpg** for question (a).

1.1.1 (1)

Perform 1^{st} order edge detection and output the edge maps as **sample1.jpg**.

Motivation From the 1^{st} order derivative, we could observe the difference within the nearest points. Use non-parametric approaches to conduct edge detection.

Approach We could follow the steps of discrete case orthogonal gradient in *Lec 3 page 12 – 17*.

1. Design the mask/filter. I have implemented *n-point* with $3, 4, 9$ in *Lec 3 page 12, 14, 15*.
2. Calculate **magnitudes/gradients** by convolution/weighted average with `scipy.signal.convolve2d`.

Figure 1: **sample1.jpg**

3. Draw the histogram of magnitudes/gradients.
4. Deciside the threshold by observing histogram.

Performance of results In the end, I choose **9-point** with $k = 2$, i.e. *Sobel Mask* as my mask. And give the threshold as 25 according to **result1.jpg** Histogram of magnitudes.

Result of problem 1(a)(1): **result1.jpg** 1^{st} order edge detection.

Discussion For checking my approach is right or not, I also observe the **column** and **row** magnitudes in intermediates: **result1.jpg** Column magnitudes & **result1.jpg** Row magnitudes.

With a different threshold, I try **result1.jpg** with threshold $T = 2$ and find out we keep too many edges.

1.1.2 (2)

Perform 2^{nd} order edge detection and output the edge maps as **result2.jpg**.

Motivation We find out there is some tricky problem of deciding the threshold in 1^{st} order edge detection. So we could get the 2^{nd} order derivative to obtain the **zero-crossing** points to support us.

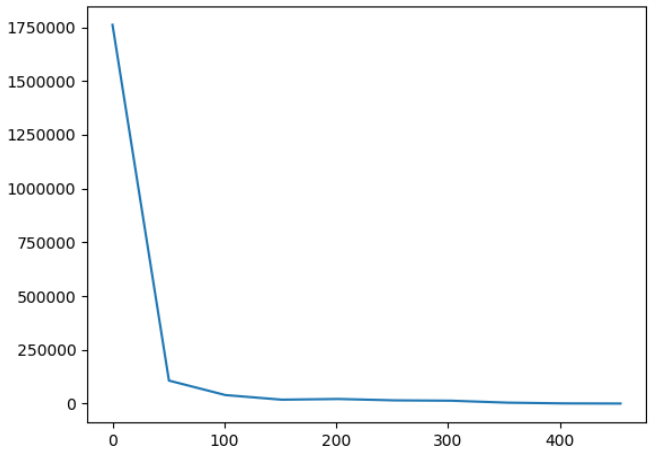

Figure 2: **result1.jpg** Histogram of magnitudes

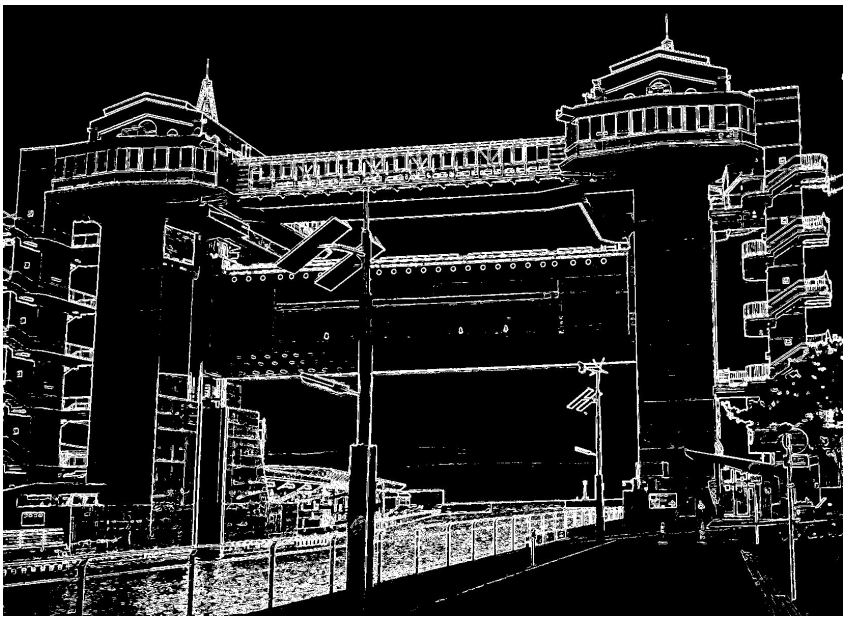

Figure 3: **result1.jpg** 1st order edge detection

Figure 4: **result1.jpg** Column magnitudes

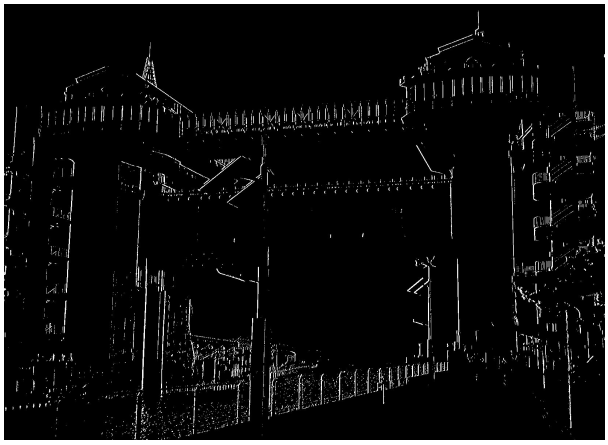

Figure 5: **result1.jpg** Row magnitudes

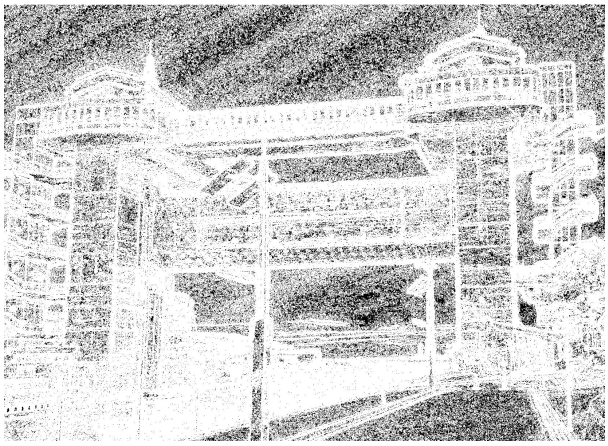

Figure 6: **result1.jpg** with threshold $T = 2$

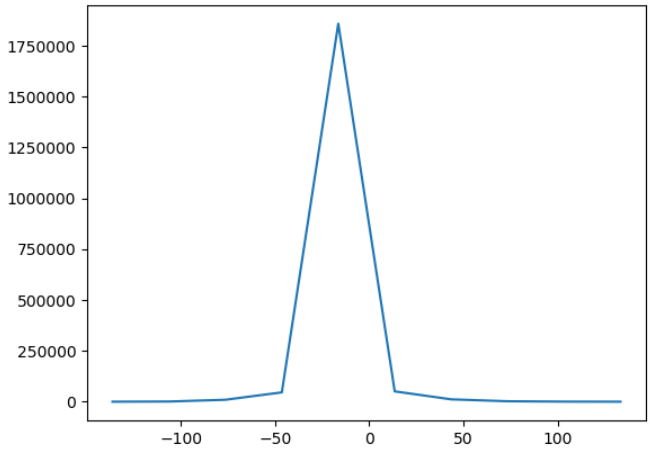

Figure 7: **result2.jpg** Histogram of Laplacian

Approach We could follow the steps of discrete approximation of Laplacian in *Lec 3 page 26 – 37*.

1. Design the mask/filter. I have implemented *n-neighbor* with $4, 8$. For *8-neighbor*, there are *separable*, *non-separable* and other mask array in *Lec 3 page 29*.
2. Compute **Laplacian** by convolution/weighted average with `scipy.signal.convolve2d`.
3. Draw the histogram of magnitudes/gradients.
4. Set up a threshold to separate **zero** and **non-zero** by observing histogram, output as G'.
5. For $G'(j,k) = 0$, decide whether (j,k) is a **zero-crossing** points by its **8-neighbor** contains -1 & 0 simultaneously.

Performance of results In the end, I choose **non-separable 8-nieghbor** as my mask. And give the threshold as 5 according to **result2.jpg** Histogram of Laplacian.

Result of problem 1(a)(2): **result2.jpg** 2^{nd} order edge detection.

Discussion I try the different threshold to observe the variation **result2.jpg** Threshold $T = 25$ & **result2.jpg** Threshold $T = 50$.

1.1.3 (3)

Perform *Canny* edge detection and output the edge maps as **result3.jpg**.

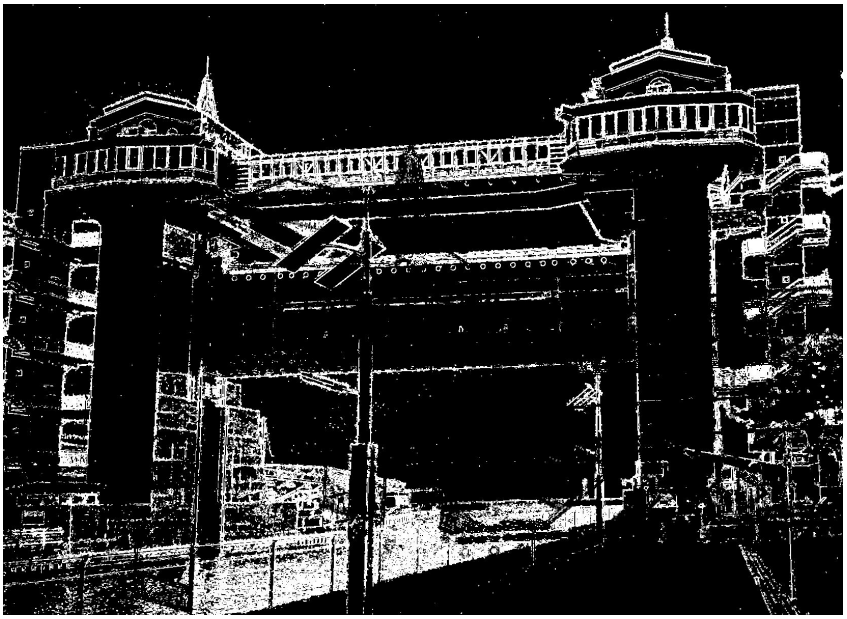

Figure 8: **result2.jpg** 2nd order edge detection

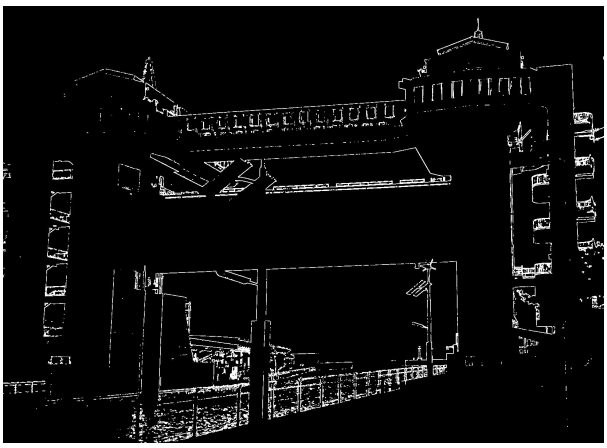

Figure 9: **result2.jpg** Threshold $T = 25$

Figure 10: **result2.jpg** Threshold $T = 50$

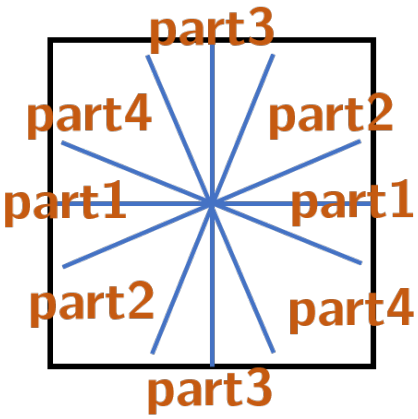

Figure 11: Illustration of non-maximal suppression

Motivation *Canny* edge dector is famous methods. It has good detection, localization and single response constraint, i.e. the edge should be as **thin** as possible. Finally, its steps are simple to implement.

Approach We could follow the five-step of *Canny* method in *Lec 3 page 19 – 23*.

1. Noise reduction with **Gaussian filter** mask.
2. Compute gradient magnitude and orientation by reusint 1^{st} order detection methods.
3. Non-maximal suppression, follow the Illustration of non-maximal suppression, I split the **8-neighbor** of $G(j,k)$ as **4 direction parts**. And according to the formula in *Lec 3 page 21* to calculate $G_N(j,k)$.
4. Hysteretic thresholding, label each pixels according to two threshold T_H & T_L. And obtain **exact edge pixels** and **candidate pixels**.
5. Connected component labeling method, I implement **8-neighbor** methods that if the neighbor of candidate pixels contains **any other** edge pixels **or any other** candidate pixels. I'll return it as edge pixels.

Performance of results In the end, I choose $T_H = 50$ & $T_L = 15$ after several tries.

Result of problem 1(a)(3): **result3.jpg** Canny edge detection.

Discussion I reveal the intermediates of Canny edges.

- **result3.jpg** Step1 Noise reduction
- **result3.jpg** Step2 Compute gradient magnitude and orientation

Figure 12: **result3.jpg** Canny edge detection

Figure 13: **result3.jpg** Step1.

- **result3.jpg** Step3 Non-maximal suppression
- **result3.jpg** Step4 Hysteretic thresholding

1.1.4 (4)

Apply an edge crispening method to the given image, and output the result as **result4.jpg**. Please also generate an edge map of **result4.jpg** as **result5.jpg**.

For result4.jpg

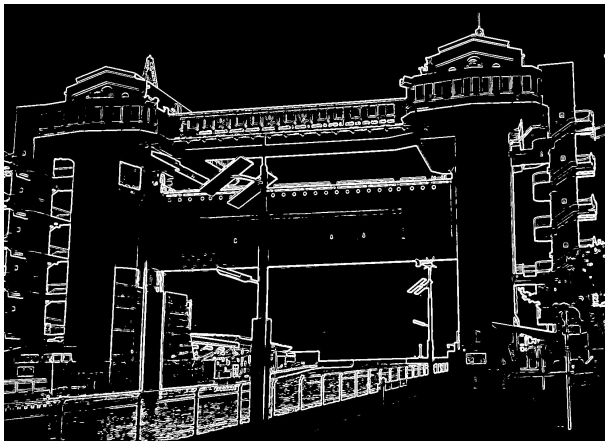

Figure 14: **result3.jpg** Step2.

Figure 15: **result3.jpg** Step3.

Figure 16: **result3.jpg** Step4.

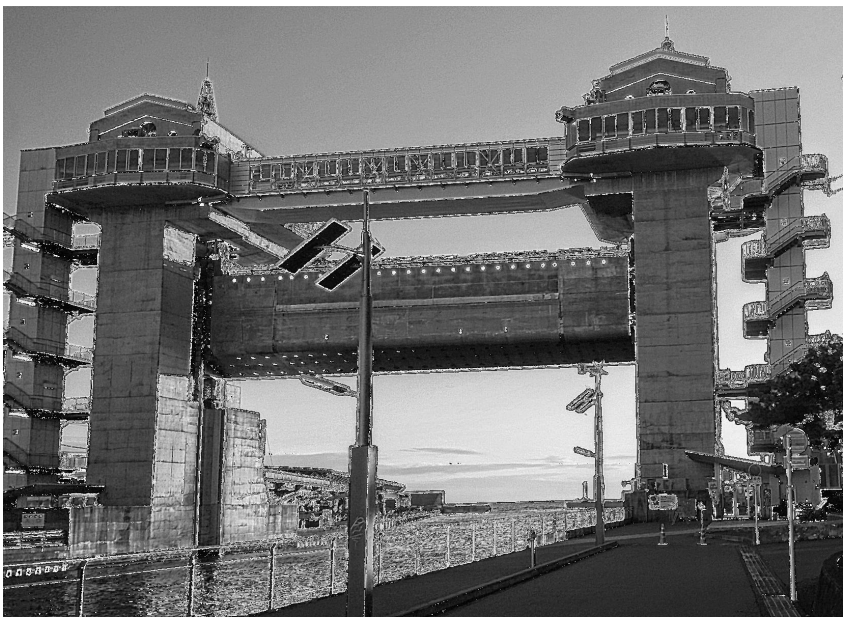

Figure 17: **result4.jpg** Edge crispening

Motivation We could treat edge as **high frequency** and conduct **high-pass filtering**. But remember that, if there are some **noise** in the image, we also **amplify** the noise at the same time. So I try **Unsharp masking** first. It combine low-pass and all-pass to get better results.

Approach We could follow the unsharp masking and its combination in *Lec 3 page 6*.

1. Build low-pass filtering matrix. Reuse homework 1.
2. Compute low-pass result $F_L(j,k)$ by convolution/weighted average with `scipy.signal.convolve2d`.
3. Compute $G(j,k)$ by combining all-pass $F(j,k)$ and low-pass $F_L(j,k)$ with formula eq. (1).

$$G(j,k) = \frac{c}{2c-1}F(j,k) - \frac{1-c}{2c-1}F_L(j,k) \tag{1}$$

where $\frac{3}{5} \leq c \leq \frac{5}{6}$

Performance of results In the end, I choose $c = \frac{3}{5}$.

Result of problem 1(a)(4): **result4.jpg** Edge crispening.

Discussion For different parameters c, **to do...**

Figure 18: **result5.jpg** Edge crispening

For result5.jpg

Approach I use **Canny edge detection** on **result4.jpg** Edge crispening.

Performance of results In the end, I keep $T_H = 50$ & $T_L = 15$ as the former.

Result of problem 1(a)(4): **result5.jpg** Edge crispening.

1.1.5 (5)

Compare **result1.jpg**, **result2.jpg**, **result3.jpg** and **result5.jpg**. Provide some discussions and findings in the report.

From 1^{st} order to 2^{nd} order In constrast to 1^{st} order edge detection **result1.jpg** 1^{st} order edge detection, 2^{nd} order edge detection could sketch more **particulars**. But we could see there some **dust** on the background in 2^{nd} order edge map.

Both of them we could observe the **histogram** of magnitudes and Laplacian respectively.

From 1^{st} & 2^{nd} order to Canny I consider that it is difficult of choosing T_H & T_L. But in contrast to 1^{st} and 2^{nd} order edge detection, Canny edge detection is good methods that it is **common/reasonable(?) edge map** for me.

Figure 19: **sample2.jpg**

Edge crispening than Canny It is interesting that we could get more **details** after **edge crispening**. We could see the *streamline over the river*. But it makes me fear too intensive.

Conclusion of Problem 1 (a) The tricky points of **edge detection** is choose a **good threshold** that keep/remove points to construct edge.

And it seems "NO FREE LAUNCH". If we want to use **Canny detection** to get better results, we need to carefully check parameters. The future work may design the mechanism of choosing threshold automatically by different user preference.

1.2 (b)

Please design an algorithm to obtain the edge map of **sample2.jpg** as best as you can. Describe the steps in detail and provide some discussions.

Original image **sample2.jpg** for question (b).

Motivation The difficulty of the problem is that the **sample2.jpg** is too bright. But if we **increase its contrast**. You could see some **dust pixel** start appearing *on the sky around of buildings*.

So we should carefully design the **image enhancement** and **denoising** (if necessary) as **data preprocessing** then do **edge detection**.

Approach In the end, my approach is

Figure 20: Result of problem 1 (b)

1. Use **10 times** **low-pass filter** to denoise.
2. Use **power law transformation** with $p = 1.4$ to improve the contrast of **with region**, *sky region*.
   Note: An amazing is that I make transformation on $(0, 255)$ **instead** **normalization** on $(0, 1)$.
3. Use **edge crispening** to make edge more clear.
4. Use **Canny edge detection** with $T_H = 60$ & $T_L = 30$ to get result.

Performance of results Result of problem 1(b): Result of problem 1 (b).

Discussion I think there are two key points:

- Denoise to remove the **dust**. I consider that it is why we see the image with **veil**. So I choose **low-pass filtering**.
- The other one is make **really strong contrast** on buildings and sky.

Here is my intermediates, enjoy it! :-)

- Problem 1 (b) Step1 10-times low-pass filtering
- Problem 1 (b) Step2 Non-normalization of power-law transformation
- Problem 1 (b) Step3 Edge crispening

Figure 21: Problem 1 (b) Step1.

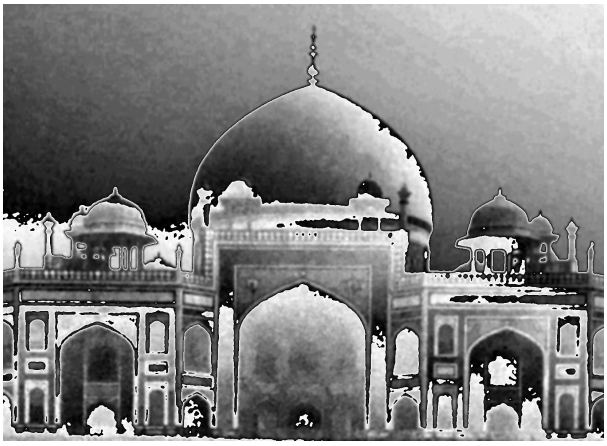

Figure 22: Problem 1 (b) Step2.

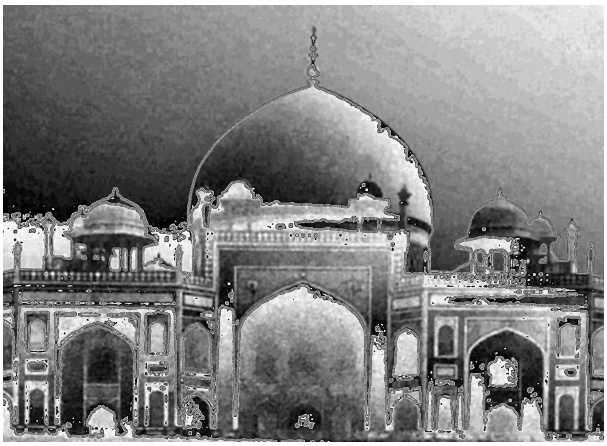

Figure 23: Problem 1 (b) Step3.

Figure 24: **sample3.jpg**

2 Problem 2: GEOMETRICAL MODIFICATION

In this problem, please design several geometrical modification algorithms to meet the following requirements. Your results don't have to be exactly the same as sample images, just try to make the effects.

2.1 (a)

Please design an algorithm to make **sample3.jpg** become **sample4.jpg**. Output the result as **result6.jpg**. Please describe your method and implementation details clearly. (hint: you may perform rotation, scaling, translation, etc.)

Original image **sample3.jpg** for question (a).

Motivation I think we could observe Reference line of problem 2 (a) and get:

- Rotate 78°.
- Scale up 150%.
- Shift from current to $(-200, 150)$.

Approach We could follow the structure of **generalized linear geometrical transformation**, and use **backward treatment** in *Lec 3 page 56 & 50*.

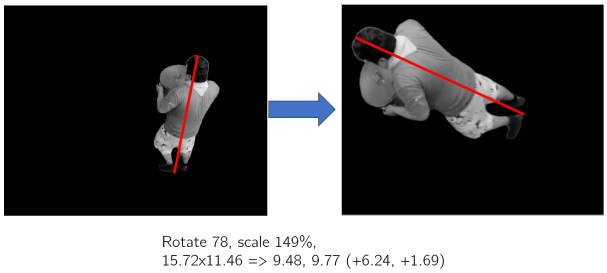

Figure 25: Reference line of problem 2 (a)

1. Design the **translation, scaling, rotation** matrix as operators in *Lec 3 page 54*.
2. Build operators A^{-1}, which is inverse of multiplication of above operations A.
3. Build **grid map** of *xy vectors*.
   Remember shift it to center by minus $(\frac{\text{width}}{2}, \frac{\text{height}}{2})$ of image size.
   As original point is the center of image.
4. Get *uv vectors* by applying inverse transform A^{-1} on *xy vectors*.
5. Shift *uv vectors* & *xy vectors* to original position by add $(\frac{\text{width}}{2}, \frac{\text{height}}{2})$.
6. Get pixels within image boundaries. That means check whether *uv vectors* are all in the image range.
7. Create the blank canvas and draw from the original image on it.
   I use the **nearest neighbor interpolation** by rounding *xy vectors* as well as *uv vectors* into integer.

Performance of results As former observation, I select rotate 78°, scale up 150% and shift to $(-200, 150)$.

Result of problem 2(a): **result6.jpg** of problem 2 (a).

Discussion I also try the **forward treatment**, and we could see some **black grid** over the image **result6.jpg** of problem 2 (a).

2.2 (b)

Imagine that there is a black hole in the center absorbing **sample5.jpg**. Please design a method to make **sample5.jpg** look like **sample6.jpg** as much as possible and save the output as **result7.jpg**.

Original image **sample5.jpg** for question (b).

Figure 26: **result6.jpg** of problem 2 (a)

Figure 27: **result6.jpg** problem 2 (a) by forward treatment

Figure 28: **sample5.jpg**

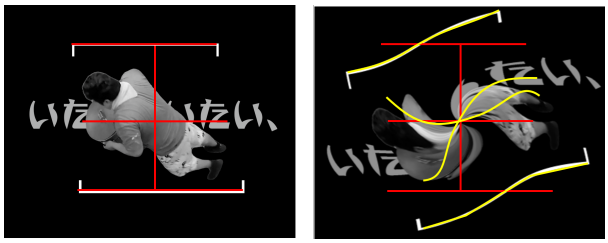

Figure 29: Reference line of problem 2 (b)

Motivation After observing Reference line of problem 2 (b), I feel maybe we could start from *Polar coordinate system*. Because it seems the **twisted** of original image grid map.

Approach Thanks to TAs and my friends explanation. I try this steps of *blackhole*:

1. Set up the parameters:
   - `rotation_radius`: radius of rotation from center.
   - `twist_scale`: The scale of twist.
   - `hole_radius`: the radius of holw.
     Note: it is a litter different from `rotation_radius`, it affects **size of absorbing center**.
2. Calculate radius r & angle θ from center of image.
3. Calculate **absorbing scale** w.r.t radius and `rotation_radius` with formula absorb_scale $= 0.4(1.0 - \frac{r}{\text{rotation\_radius}})^8$

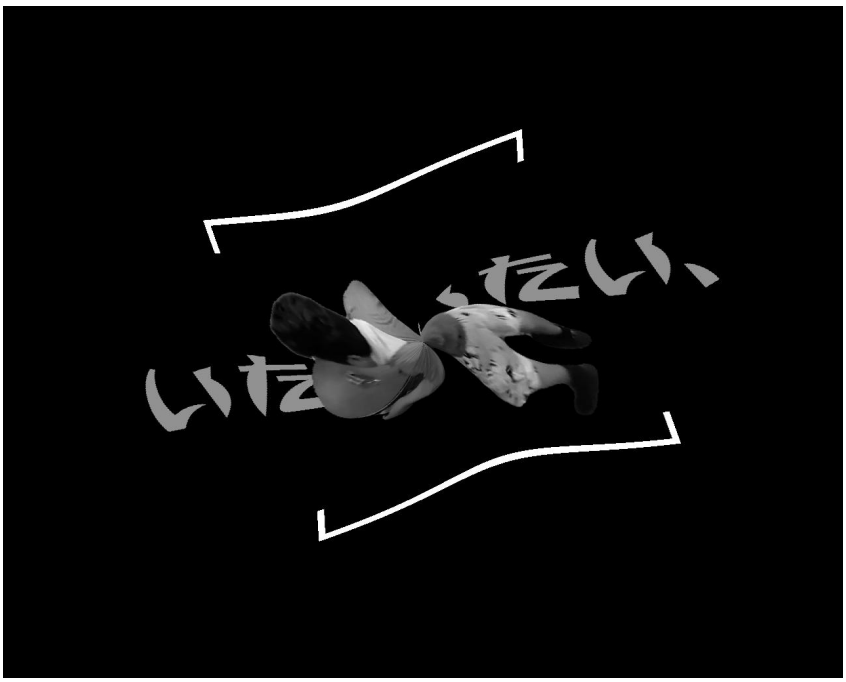

Figure 30: **result7.jpg** of problem 2 (b)

4. We only change the **absorb scale** > 0 as we could absorb the pixels to radius $= 0$.

5. Create the blank canvas and draw from the original image on it.
   I use the **nearest neighbor interpolation** by rounding *xy vectors* as well as *uv vectors* into integer.
   Note: A little difference here is we need to give **different scale** for `absorb_scale`.
   So I go back to use `for loop` instead of matrix.

Performance of results In the end, I select the following parameters as the last submission.

- `rotation_radius = 1000.0`
- `twist_scale = 0.4`
- `hole_radius = 100`

Result of problem 2(a): **result7.jpg** of problem 2 (b).

Discussion The original idea is making a **vortex**. But we couldn't get the operators A & A^{-1} at this time.

1. Build **grid map** of *xy vectors*.
   Remember shift it to center by minus $(\frac{\text{width}}{2}, \frac{\text{height}}{2})$ of image size.
   As original point is the center of image.

2. Convert from *Cartesian Coordinates* (x, y) to *Polar Coordinates* (r, θ) (radius and angle).

3. Design **spiral transformation**, this is my **main parts**:
   Let we inverse (r, θ) backward to (ρ, ϕ), given scalar k, s.t.

   - $\phi = \theta + \frac{k}{r+\epsilon}$
   - $\rho = r$

   Where ϵ is a small number to prevent `divide by zero`.

4. Get *uv vectors* by converting from *Polar Coordinates* (ρ, ϕ) to *Cartesian Coordinates* (u, v).

5. Shift *uv vectors* & *xy vectors* to original position by add $(\frac{\text{width}}{2}, \frac{\text{height}}{2})$.

6. Get pixels within image boundaries. That means check whether *uv vectors* are all in the image range.

7. Create the blank canvas and draw from the original image on it.
   I use the **nearest neighbor interpolation** by rounding *xy vectors* as well as *uv vectors* into integer.

It is really difficult to me that I can't figure out how to make **absorbing effects**. My approach is creating the **spiral**. But it keeps the center as *vortex* not *black hole*.

Maybe it is not a good way to conduct geometrical transformation on *Polar coordinates*.

Result of problem 2(b): **result7_spiral.jpg** of problem 2 (b).

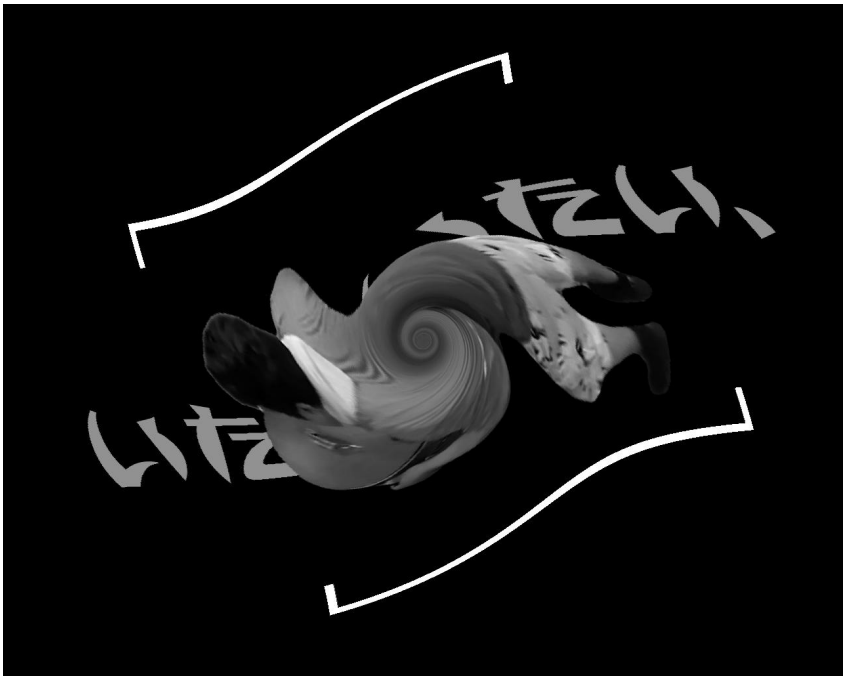

Figure 31: **result7_spiral.jpg** of problem 2 (b)