DVE HW2

Photos

meta data

- **type** : JPEG image
- **width** : 2664 pixels
- **height** : 4000 pixels
- **aperture** : f/8.0
- **shutter speed** : 1/60 sec
- **focal length** : 10.1 mm
- **Sensor** : 1/2.3"
- **number of photos** : 26

cylinder projection

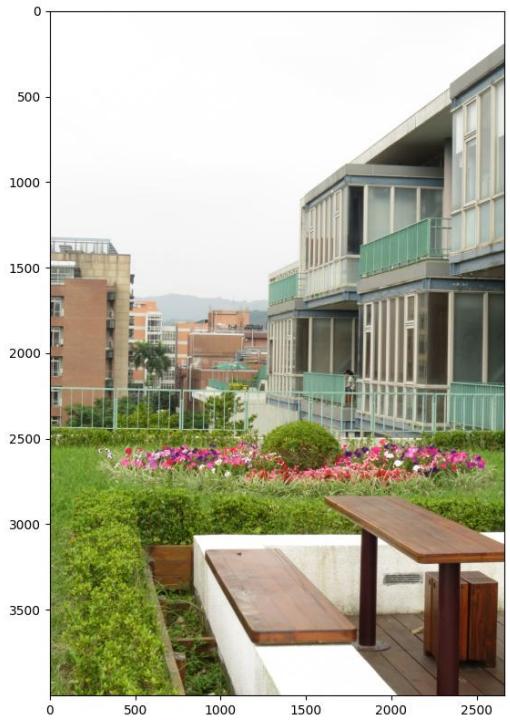
- **focal length**

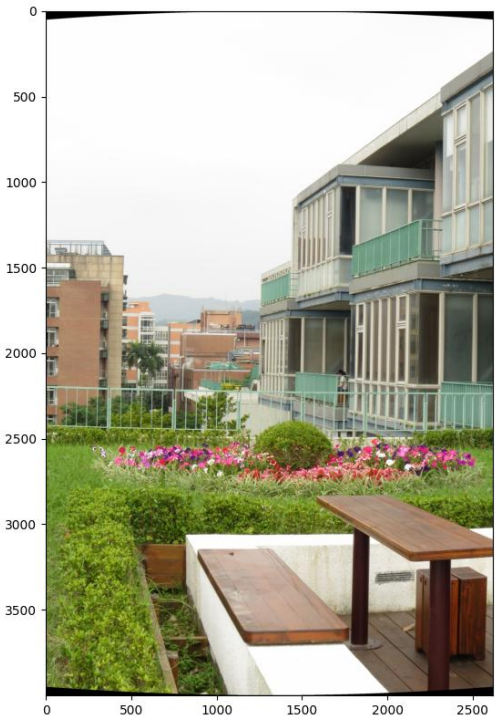
$$\mathrm{F(pixel)} = \mathrm{F(mm)} \times \frac{\mathrm{ImageWidth(pixel)}}{\mathrm{SensorWidth(mm)}}$$

$$\text{For our result photos, } \mathrm{F(pixel)} = 10.1 \times \frac{2664}{4.5}$$

- **projection**

$$x' = f \tan^{-1} \frac{x}{f}, \qquad y' = f \frac{y}{\sqrt{x^2 + f^2}}$$





Features - Multi-Scal Oriented Patches

detector

- **Scale picture**

$$
\begin{aligned}
&I(x,y) = 0.2125 \times r(x,y) + 0.7154 \times g(x,y) + 0.0721 \times b(x,y) \\
&P_0(x,y) = I(x,y) \\
&P_l'(x,y) = P_l(x,y) * g_{\sigma_p}(x,y) \\
&P_{l+1}(x,y) = P_l'(sx,sy)
\end{aligned}
$$

- **Harris corner matrix**

$$
H_l(x,y) = \nabla_{\sigma_d} P_l(x,y) \nabla_{\sigma_d} P_l(x,y)^T * g_{\sigma_i}(x,y)
$$

$$
\nabla_\sigma P(x,y) = \nabla P(x,y) * g_\sigma(x.y)
$$

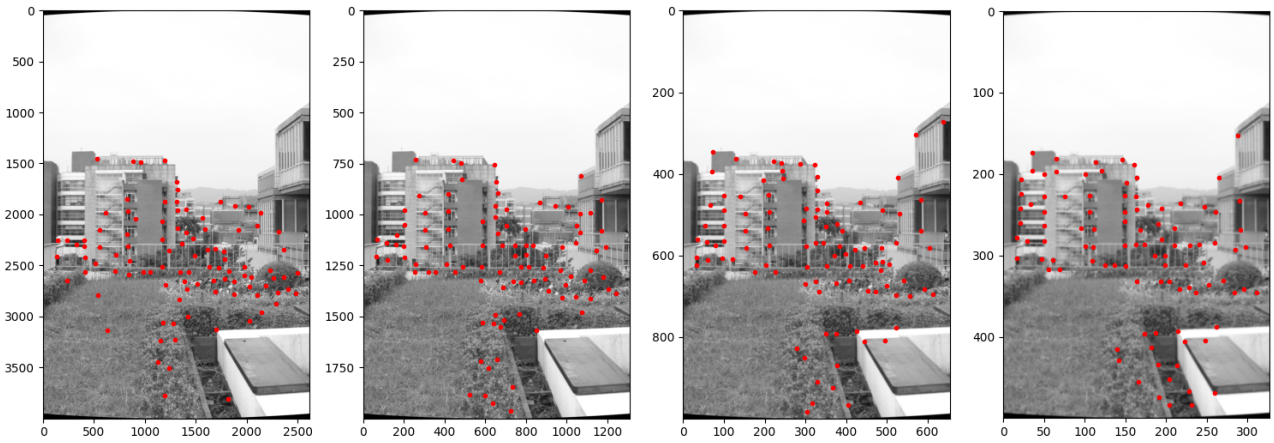
$$
\frac{dP(x,y)}{dx} = P(x,y) * k_x(x,y), \quad k_x(x,y) = \begin{bmatrix} -\frac{1}{3} & 0 & \frac{1}{3} \\ -\frac{1}{3} & 0 & \frac{1}{3} \\ -\frac{1}{3} & 0 & \frac{1}{3} \end{bmatrix}
$$

$$
\frac{dP(x,y)}{dy} = P(x,y) * k_y(x,y), \quad k_y(x,y) = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} \end{bmatrix}
$$

- **Corner detection function**

$$
f_{HM}(x,y) = \frac{det\ H_l(x,y)}{tr\ \mathrm{H}_l(x,y)}
$$

- **Implementation**
  - **scale picture** :
    $s = 2, \sigma_p = 1.0$
    $\text{number of scaled photos } = 4$
  - **Harris corner matrix** :
    $\sigma_d = 1.0, \sigma_i = 1.5$
  - **Select feature** :
    $|f_{HM}(x, y)| > 10$
    $r = max(1, \frac{min(w,h)}{40})$, only retain maximums in a meighborhood of radius r
    $\text{number of keypoints'} \leq 1000$ (for each scaled photo)
- **Example** (number of keypoints ≤ 100)

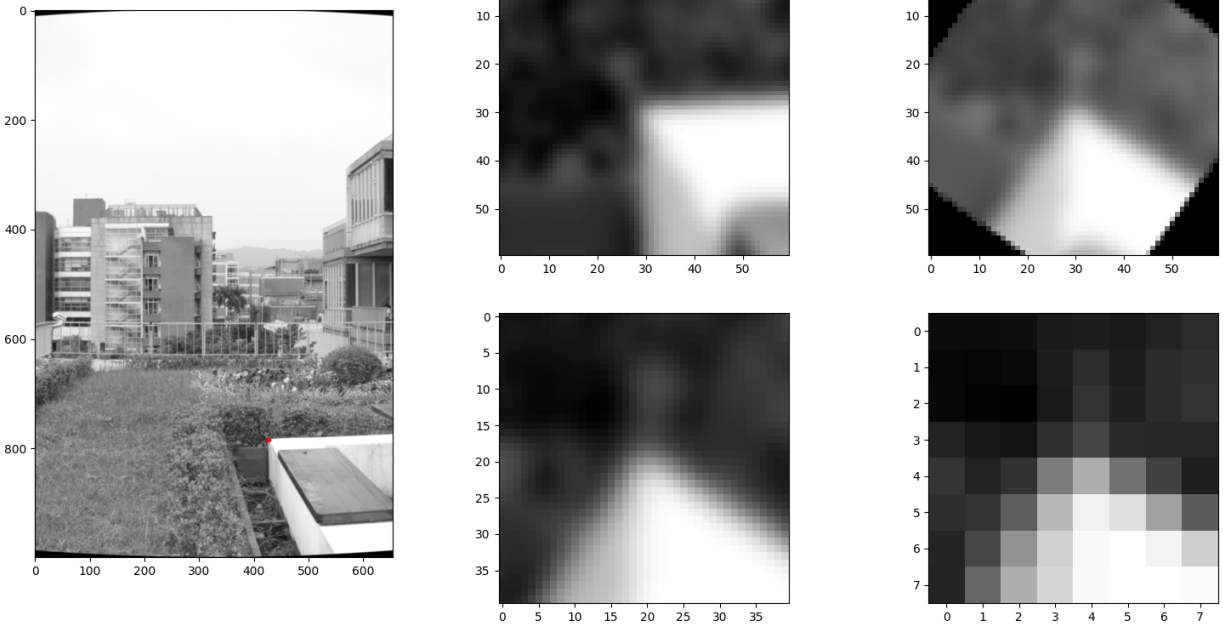


Descriptor

- **orientation assignment** : from blurred gradient

$$
\begin{aligned}
&u_l(x,y) = \nabla_{\sigma_o} P_l(x,y) \\
&[cos\theta, sin\theta] = u/|u| \\
&\theta' = arctan(\frac{sin\theta}{cos\theta}), \quad \theta' \in (-\frac{\pi}{2}, \frac{\pi}{2})
\end{aligned}
$$

- **cut and rotate** :
  1. Get 60×60 picture certering at keypoint from blur image. (Make sure that I can get 40×40 picture after rotate.)
     $blur = P_l(x,y) * g_{2\times\sigma_p}(x,y)$
  2. rotate the picture by $-\theta'$, note that :
     $\theta' = f(\theta), \quad f(\theta) = f(\theta + \pi)$
     θ and $\theta + \pi$ have the same θ' value; however, rotating them by the same angle is easier to distinguish them. Because the gradient orientations are opposite.
  3. Get 40×40 picture certering at keypoint, and scale by $s = 5, i.e.$ get 8×8 output image $i.e.$ 64 dimension vector.
- **Example**
  - **left** : Image at scale 4 with a keypoint.
  - **middle top** : 60×60 image from blur image.
  - **right top** : rotate by $-\theta'$
  - **middle below** : 40×40 image.
  - **right below** : output image.



Pairing Features from Two Photo

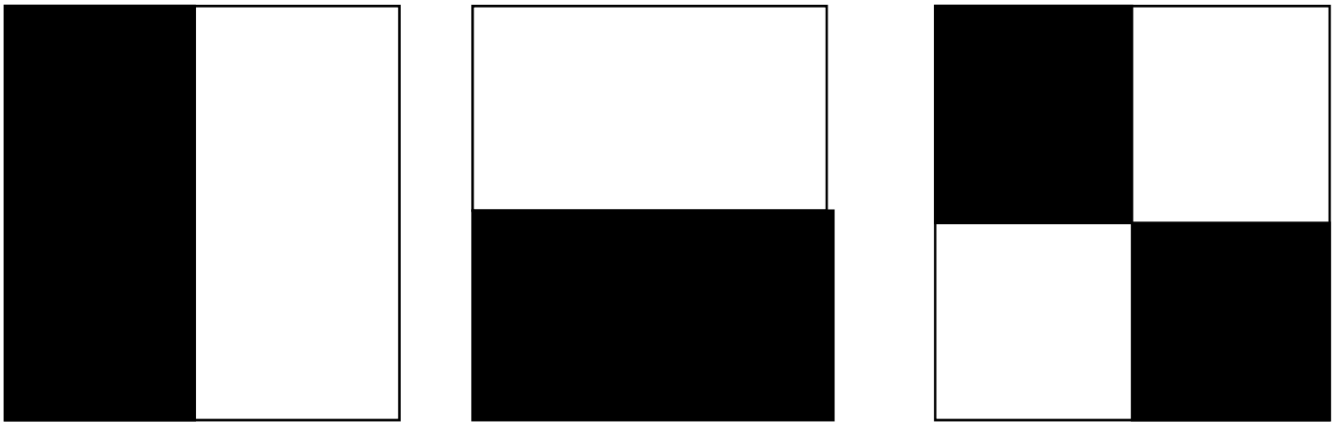
After getting the 64-dimension features from the photos, we match the feature from two neighborred picture.

We try the **bruteforce method** first.
To pair a feature in a photo, we normalize its 64-dimension vector, and then just simply go throug all the feature (also be normalized beforehand) from the other photo (which is next to the photo we want to pair) and calculate their "squared distance" in 64-dimension vector ($\Sigma(A[x,y]-B[x,y])^2$), and then match it with the nearest one.

Wavelet-based Hashing ans Bins (Not adoptted)

We also try using Wavelet-based hashing to hash the 64 dimention vector of a feature into a 3- dimention vector hash with three binary mask. And then use these 3-dimention hash to categorize all the features into $10 \cdot 10 \cdot 10$ bins.

The three masked we tried:



After the distribute features into these bins, we pairing the feature with a similar way in bruteforce method. But this time, we only test the distance with the features in the adjacent 3 bins in the three dimentions, which means the neighboring 27 bins.

Theoretically, this method makes the paring process 1000/27 times more fast, and it indeed accelerate our pairing a lot. However, in our test, this method have a very low accuracy (to pairing the nearest point) then the bryteforce method. It has only 50% chance to pair the nearest feature correctly, even when we change the searching range to the adjacent 5 bins in each dimention (neightboring 125 bins), which has only 1000/125 accelaration.

Result of using the two methods to pairing photo A, B. (The color of dots in photo B, means the algorithm pairing it to the one with the same color in photo A.)

Brute-force has higher accuracy, and more feature pairs found.

So after the trying, we decide to use the original bruteforce method with some simple GPU acceleration (Python cupy).

Calculate the Offset from Feature Pairs (RANSAC)

After paring the feature, we gets pairwise offsets. We use the RANSAC algorithm, to find the final offset with the two photos. In this process, we repeatedly pick an offset of a random feature pair, an give it a score by counting the number of the other offsets whose has a distance lower then the threshold (we set this threshold as sqrt(10) pixels, and repeat the picking process 500 times).

Panorama

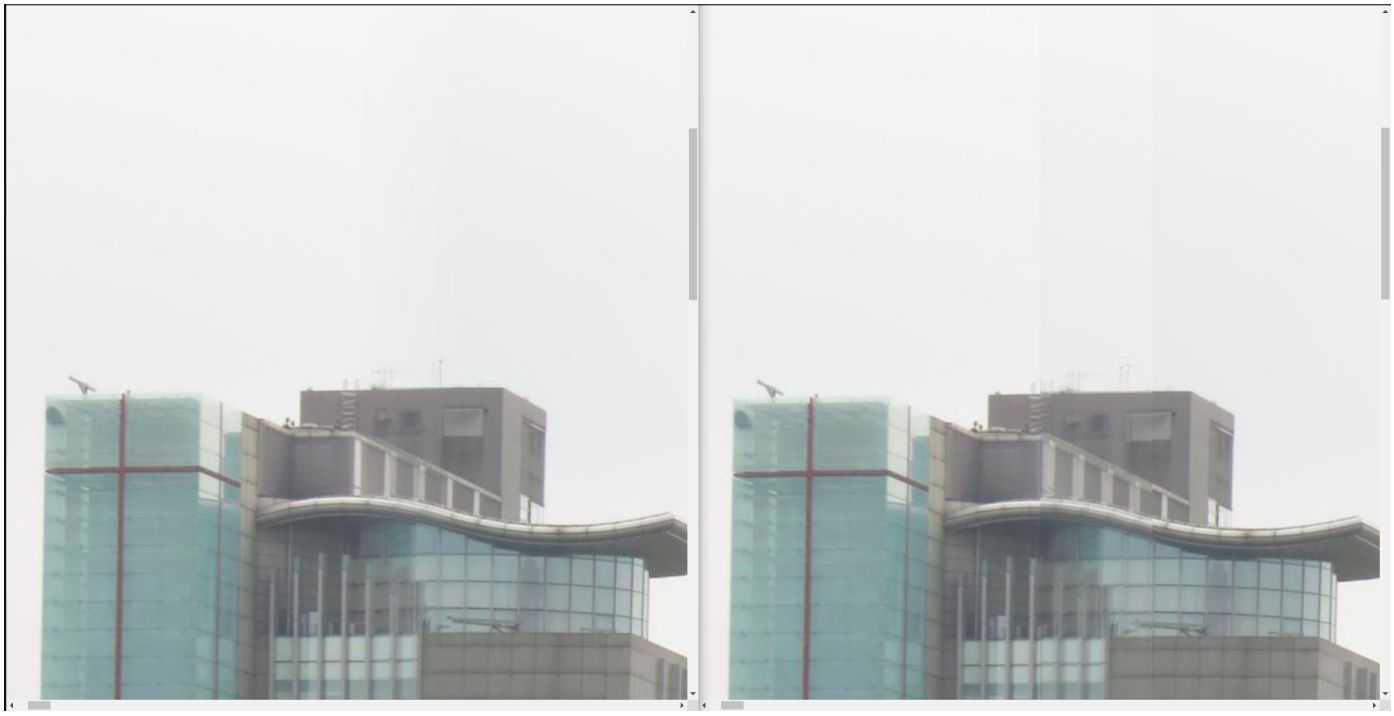
After getting the offset of all pictures, we put them together.
For the overlapping part, we first try to just calculate the average RGB value of the pixels. But it does not perform well. So we try the below improvement.

Slice the Uneccessary Overlapping

After we get the panorama, we find the overlapping making too many goasting(鬼影). So we slice the margin of photos, leaving it with an only 100-pixel overlapping with other photos. (Before our slicing, there is usually a 1000-pixel overlapping between each two photos.)

linear Blending

The slicing make our picture more cleary in most part, but also let the unalignment more apeearent in the margin. To deal with this problem we use the linear-blending in the margin, make the margin more smooth.



(The left photo is processed with linear-blending, it has no awkward lines in the sky. The right photo is processed with simply getting average of overlapping part, which has two appearent margin in the sky, and has more ghosting in the middle.)

Panorama from example photo:

Our photo:

Result