VFX Homework2 Report

Team 22 R11944042徐翊凌 R11922014陳建豪

Code Explanation

Warping

We perform cylindrical projection on input images at first, so we only need to calculate the translation relation when matching images.

We share the same camera with r11944008 and r11922002, so we directly take the focal length (1921).

We first calculate the width of the projected image I' (height remains the same). We take the focal length as radius r, and the angle can be calculated by $\theta = 2 \cdot \tan^{-1}(\frac{\text{image width}/2}{\text{focal length}})$, and thus the resulting width is $r\theta$.

After we have the size of I', we iterate through every pixel on I' and determine which pixel on the original image I should be projected to this pixel. According to the slide, we have

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

so we can easily know that $x = \tan\frac{x'}{s} \cdot f$. As the slide suggests, we set $s = f$. After we

$$y = \frac{y'}{s} \cdot \sqrt{x^2 + f^2}$$

know the pixel position on I, we simply set $I'(x', y') = I(x, y)$.

Feature Detection

We adopt the Harris Corner Detection, and the following are the briefly explanation of our implementation.

1. Apply Gaussian blur on the image and obtain G_σ^x
2. Compute derivatives on x-direction and y-direction of blurred image
   - $I_x = G_\sigma^x * I$
   - $I_y = G_\sigma^y * I$
3. Compute products of derivatives at every pixel
   - $I_{x^2} = I_x * I_x$
   - $I_{y^2} = I_y * I_y$
   - $I_{xy} = I_x * I_y$

4. Compute the sums of the products of derivatives at each pixel
    - $S_{x^2} = G_{\sigma'} * I_{x^2}$
    - $S_{y^2} = G_{\sigma'} * I_{y^2}$
    - $S_{xy} = G_{\sigma'} * I_{xy}$
5. Define the matrix at each pixel
    - $M(x, y) = \begin{bmatrix} S_{x^2}(x, y) & S_{xy}(x, y) \\ S_{xy}(x, y) & S_{y^2}(x, y) \end{bmatrix}$
6. Compute the response of the detector at each pixel
    - $R = \det M - k(\mathrm{Tr}\, M)^2$

Feature Description

We adopt the SIFT's description method, and we will explain the essence in brief text, or the section would be too lengthy.

To generate descriptor for a keypoint, we first select the pixels around the keypoint (the window size is 16x16). Afterwards, we slide a 4x4 subwindow on the selected 16x16 window and calculate the rotated row index, rotated column index, gradient magnitude and the gradient orientation. Afterwards, we utilize these values and generate the 8 bin histogram. Eventually, by thresholding and normalizing the above information, we can obtain the final descriptor.

Feature Matching

To find the closest keypoint of a given keypoint, we simply calculate the euclidean distance between the given keypoint and the keypoints on the another image and select the one with the shortest distance, since all images are projected on the same cylindrical surface beforehand.

Image Matching

We adopt the RANSAC algorithm and reuse the result of the feature matching to find the best shifting offset of the two image, and the following is the explanation of our implementation:

1. Randomly select a keypoint match pair (k_1, k_2) and calculate the offset between the two keypoints in the x and y directions, dx and dy.
2. Shift all other keypoints on the first image by (dx, dy).
3. For every keypoint match pair, calculate the euclidean distance between shifted keypoints on the first image and the keypoints on the second image.
4. Count the inliers, whose euclidean distance after the shiftinf is less than the given threshold.
5. Record the offset with the most inliers.

Blending

To deal with the overlapping part, we implement linear blending to decide weighting value of the two overlapping pixel. The pixel value of the overlapping area would be detemined as $w_1 \cdot p_1 + (1 - w_1) \cdot p_2$, where w_1 is 1 on the left end of the overlapping area and constantly decrease to 0 until the right end of the overlapping area.

Bonus

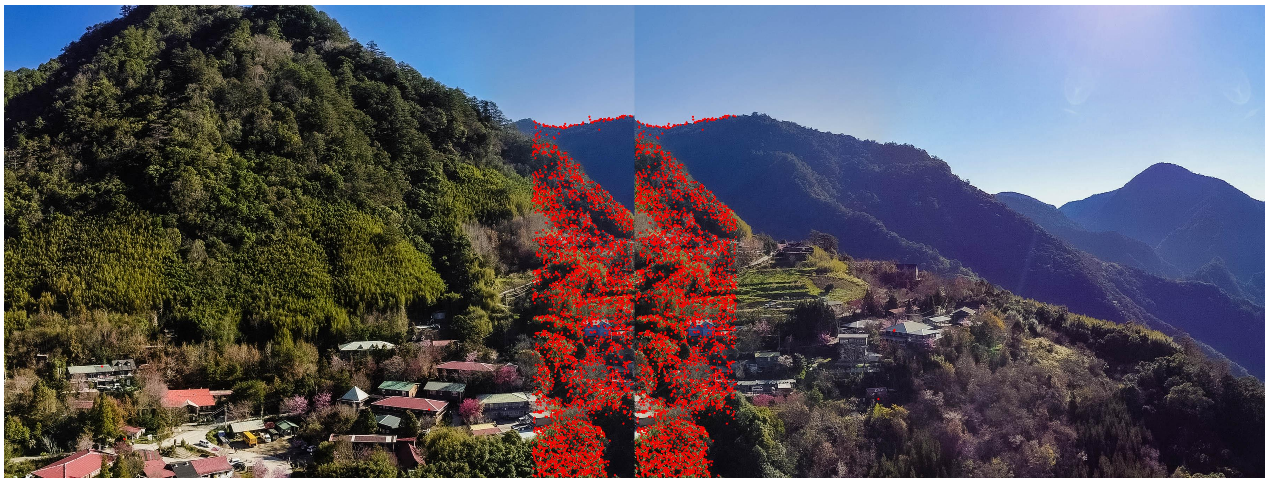
Alignment

We find the 4 vertices (upper left, upper right, lower left, lower right) of the paranoma and apply opencv's perspective warp to map the picture to a rectangle.

Poisson Blending

- We try to implement the Poisson Blending to make the panorama more naturally, here's our reference.
- The essence of the Poisson Blending is the equation: $|N_p|f_p - \sum_{q\in N_p\cap\Omega} f_q - \sum_{q\in N_p\cap\partial\Omega} f_q = \sum_{q\in N_p} v_{pq}$
  - N_p stands for the 4-connected neighbors of p
  - Ω stands for the overlapping area
  - $\partial\Omega$ stands for the boundary of the overlapping area
  - What we try to solve is $\{f_p|p\in\Omega\}$
  - To better solve the equation, the f_q in term $\sum_{q\in N_p\cap\partial\Omega} f_q$ will be replaced by the actual pixel value on the destination image, f_q^*, so the equation becomes $|N_p|f_p - \sum_{q\in N_p\cap\Omega} f_q = \sum_{q\in N_p\cap\partial\Omega} f_q^* + \sum_{q\in N_p} v_{pq}$
- The equation above can be represented as a linear system $Ax = b$
  - First, build a dict to map the index to the pixel
  - For each row of A (take row i as example)
    - $A[i][i] = |N_p|$
    - $A[i][j] = -1$ if the corresponding pixel of j is a neighbor of the corresponding pixel of i
  - For each entry of b (take row i as example)
    - Find 4-connected neighbors of the corresponding pixel of i
    - If a neighbor belongs to the boundary, add the entry with $f_q^* + v_{pq}$, otherwise add v_{pq}, where $v_{pq} = f_p^* - f_q^*$
  - Finally, just solve the system with minimum square error and fill the pixel value back to the overlapping area
- However the implementation seems to have some bugs and we can't solve it

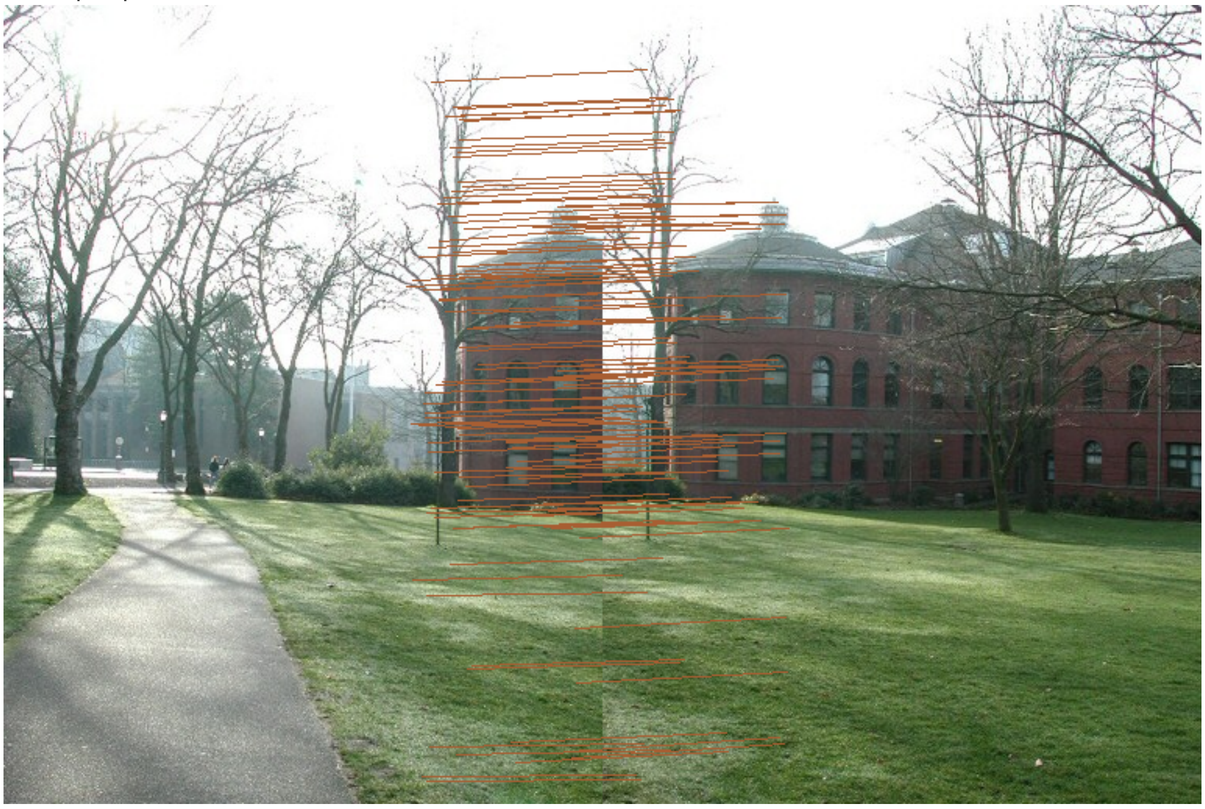
Feature Detection (Harris corner detection Algorithm)



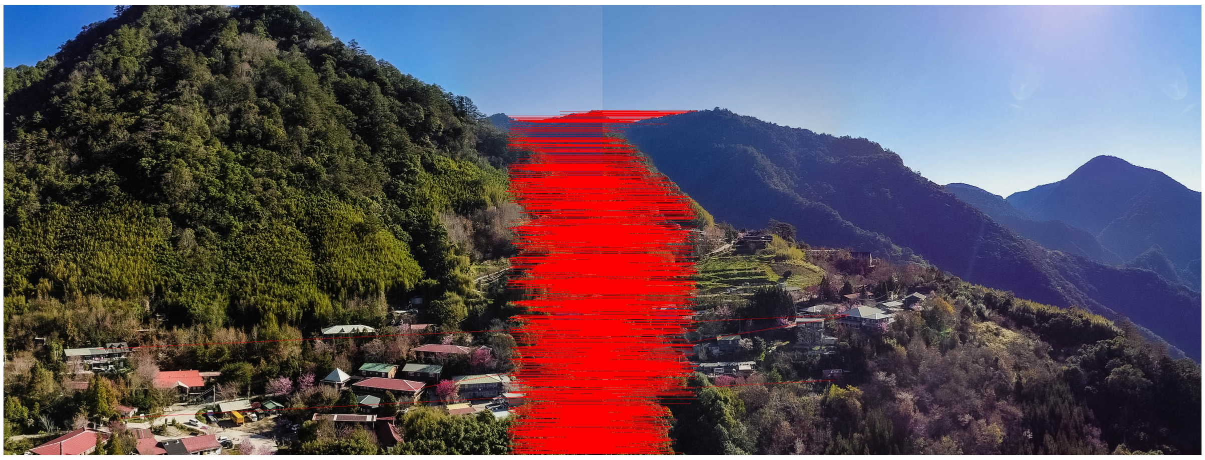
- Our artifact
- On the overlapping part, the features that we find in two images is almost same

Feature matching

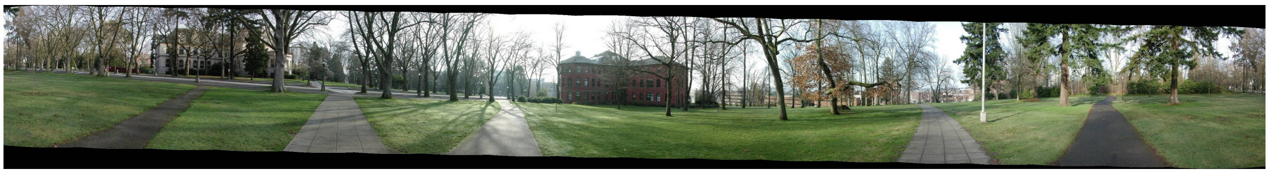
- Example picture



- Artifact



Result image



- Fig. 1 the panorama with linear blending
- Fig. 2 our artifact with linear blending and alignment