1 Part 1.

1.1 Paste your warped canvas

Figure 1: The warped canvas.

2 Part 2.

2.1 Paste the function code *solve_homography(u,v)* & *warping()* (both forward & backward)

```python
    def solve_homography(u, v):
        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')
        A = np.zeros((2 * N, 9))
        for i in range(N):
            A[i * 2, :] = [
                u[i][0], u[i][1], 1, 0, 0, 0, -u[i][0] * v[i][0],
                -u[i][1] * v[i][0], -v[i][0]
            ]
            A[i * 2 + 1, :] = [
                0, 0, 0, u[i][0], u[i][1], 1, -u[i][0] * v[i][1],
                -u[i][1] * v[i][1], -v[i][1]
            ]
        U, S, Vh = np.linalg.svd(A)
        h = Vh.T[:, -1]
        H = h.reshape((3, 3))
        return H
```

```python
def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='b'):
    h_src, w_src, ch = src.shape
    h_dst, w_dst, ch = dst.shape
    H_inv = np.linalg.inv(H)

    U_x, U_y = np.meshgrid(range(xmin, xmax), range(ymin, ymax))
    U = np.concatenate(
        ([U_x.reshape(-1)], [U_y.reshape(-1)
                             ], [np.ones((xmax - xmin) * (ymax - ymin))]),
        axis=0)

    if direction == 'b':
        V = np.dot(H_inv, U)
        V = np.divide(V, V[2])
        V_x = V[0].reshape(ymax - ymin, xmax - xmin)
        V_y = V[1].reshape(ymax - ymin, xmax - xmin)

        mask_x = (0 <= V_x) & (V_x < w_src - 1)
        mask_y = (0 <= V_y) & (V_y < h_src - 1)
        mask = mask_x & mask_y

        masked_V_x = V_x[mask]
        masked_V_y = V_y[mask]

        # bilinear interpolation
        masked_V_x_int = masked_V_x.astype(int)
        masked_V_y_int = masked_V_y.astype(int)
        d_V_x = (masked_V_x - masked_V_x_int).reshape((-1, 1))
        d_V_y = (masked_V_y - masked_V_y_int).reshape((-1, 1))

        target = np.zeros(src.shape)
        target[masked_V_y_int, masked_V_x_int] += (1 - d_V_y) * (
            1 - d_V_x) * src[masked_V_y_int, masked_V_x_int]
        target[masked_V_y_int,
            masked_V_x_int] += d_V_y * (1 - d_V_x) * src[masked_V_y_int + 1,
                                                         masked_V_x_int]
        target[masked_V_y_int,
            masked_V_x_int] += (1 - d_V_y) * d_V_x * src[masked_V_y_int,
                                                         masked_V_x_int + 1]
        target[masked_V_y_int,
            masked_V_x_int] += d_V_y * d_V_x * src[masked_V_y_int + 1,
                                                   masked_V_x_int + 1]

        dst[ymin:ymax, xmin:xmax][mask] = target[masked_V_y_int,
                                                 masked_V_x_int]

    elif direction == 'f':
        V = np.dot(H, U)
        V = np.divide(V, V[2]).astype(int)
        V_x = V[0].reshape(ymax - ymin, xmax - xmin)
        V_y = V[1].reshape(ymax - ymin, xmax - xmin)

        mask_x = (0 <= V_x) & (V_x < w_dst)
        mask_y = (0 <= V_y) & (V_y < h_dst)
        mask = mask_x & mask_y

        masked_V_x = V_x[mask]
        masked_V_y = V_y[mask]

        dst[masked_V_y, masked_V_x, :] = src[mask]

    return dst
```

2.2 Briefly introduce the interpolation method you use

In the realm of image processing, two primary interpolation techniques stand out: nearest neighbor interpolation and bilinear interpolation. A succinct overview of the principles of the bilinear interpolation approach, which was implemented in my code, is provided below.

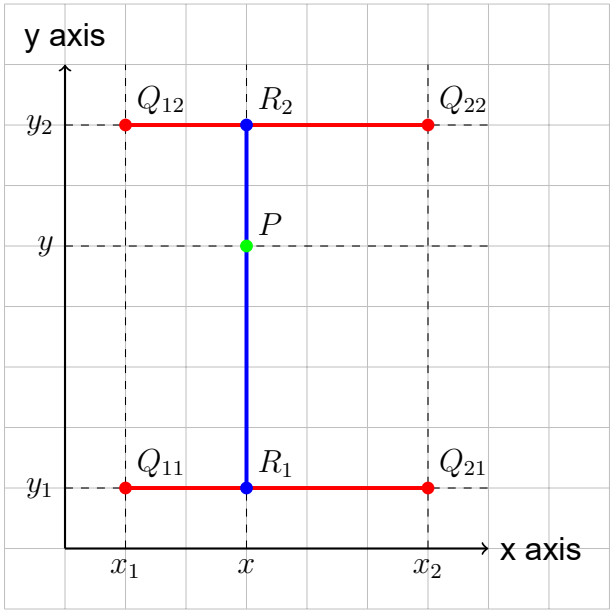

Figure 2: Interpolation diagram.

In the context of determining the value of an unknown function f at a given coordinate (x, y). Assuming we have knowledge of f at four specified points: $Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2), Q_{21} = (x_2, y_1)$, and $Q_{22} = (x_2, y_2)$.

By performing linear interpolation in the x-direction, this yields

$$f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$
$$f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

Then proceed by interpolating in the y-direction to obtain the desired estimate formula.

$$\begin{aligned}
f(x, y) =& \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \\
=& \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) + \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right) \\
=& \frac{1}{(x_2 - x_1)(y_2 - y_1)} \big(f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) \\
& + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1) \big) \\
=& \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}, \text{where} (x_2 - x_1)(y_2 - y_1) = 1 \\
=& \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}
\end{aligned}$$

As a result, by computing $f(x + dx, y + dy) = (1 - dx)(1 - dy) f(x, y) + x(1 - y) f(x + 1, y) + (1 - dx) y f(x, y + 1) + xy f(x + 1, y + 1)$ can get the result directly.

3 Part 3.

3.1 Paste the 2 warped images and the link you find

After scanning the QR code, you'll get the URL https://qrgo.page.link/jc2Y9, which subsequently redirects to the Computer Vision: from Recognition to Geometry website.

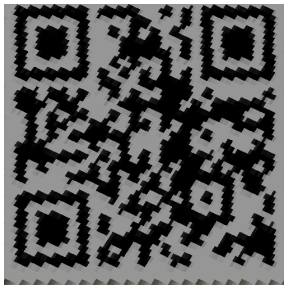

Figure 3: First warped image.

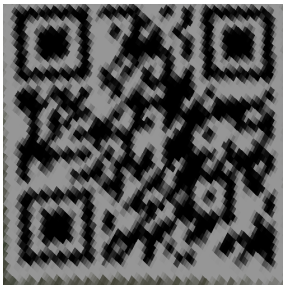

Figure 4: Second warped image.

3.2 Discuss the difference between 2 source images, are the warped results the same or different?

Figure 5: The first source image.

Figure 6: The second source canvas.

Looking at the two original images, it is obvious that the QR code on the roof in the first image (Figure 5) seems to be more square-shaped and free from distortion, whereas the QR code on the roof in the second image (Figure 6) exhibits bending and distortion. As a consequence, the QR code (Figure 3) produced from the first image displays a sharper image, whereas the QR code (Figure 3) derived from the second image seems blurrier. Nonetheless, both QR codes can be scanned to access the same website successfully.

3.3 If the results are the same, explain why. If the results are different, explain why?

The results are the same. A blurred QR code can be successfully scanned due to Reed-Solomon error correction. This technique adds redundancy to the QR code's data, allowing for error detection and correction. Even when the code is damaged or obscured, the algorithm reconstructs the original data, ensuring successful scanning. Thus, the combination of error detection, correction, and data reconstruction ensures that even a blurred QR code can be effectively interpreted by scanning devices.

4 Part 4.

4.1 Paste your stitched panorama

Figure 7: The stitched panorama.

4.2 Can all consecutive images be stitched into a panorama?

The consecutive images provided by TA can appropriately be stitched into a panorama as needed, although it's common for not all consecutive images to be suitable for panorama stitching.

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

It is obvious that if the number of detected features in the overlapping area between the two consecutive image frames is not sufficient, stitching failure will occur during the transformation, and this might cause misalignment or incomplete merging. Moreover, A simple concatenation of images with overlapping areas to form a panoramic image results in visible seams due to variations in the angle viewpoint of the camera and scene illumination, along with the spatial position errors of the images. As a result, the success or failure depends on the qualities of consecutive image frames.