COMP9517 Assignment 1

Fan Zhu z5245075

Task 1

In task 1 the input image is given as 3 images, each representing a channel in RGB. First the program read the input in '__main__' and splits the original input image (split_img()) into 3 separated images by dividing the shape of the image by 3 and rounded. Once the 3 separate images are generated, they are layered together in a new matrix to generate an initial colour image (figure 1.)

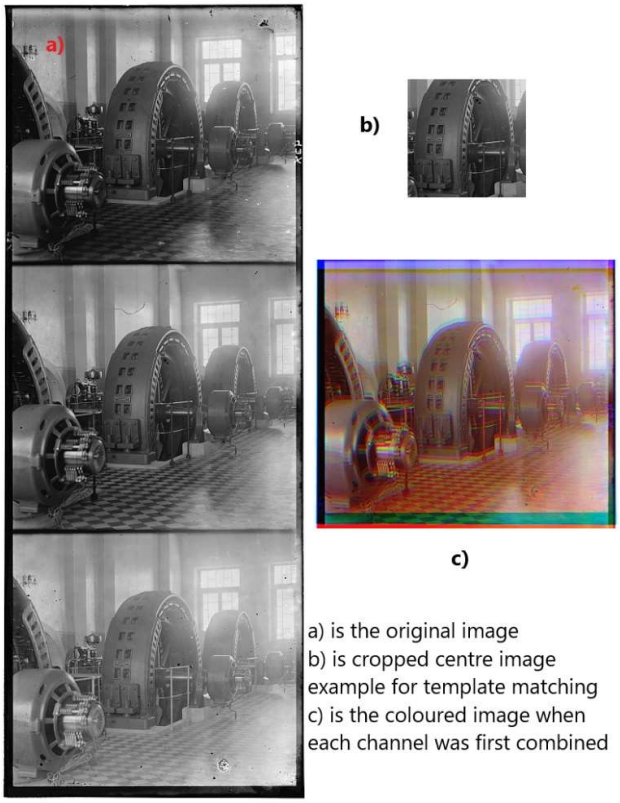

Figure 1.

We can see the image is mismatched as each layer are out of alignment with each other. The next step is to design the alignment algorithm (align() and img_alignment()) to match each channel. First the red channel is set to be template to be matched to, next each layer of image is given a score based on their translation against the template (quality_score()). Here inside the scoring function, it was decided to apply cropping of 150x150 pixels from the centre of the image and the template, so a smaller sized image without borders at the edges can be used for template matching. (Although this could be improved further by detecting and removing borders, but this was not the improvement added for Task 3).

The cropped images are checked against each other using cv2.matchTemplate() by moving through 20 pixels/20x20 (a suggested metric for low resolution images in assignment specifications). Normalized cross correlation (cv2.TM_CCORR_NORMED) was the equation used to calculate the score along with cv2.minMaxLoc() where the maximum value (max_val) from the output was used because we are using NCC. The search ends with the amount of shift in x and y when the largest score is calculated.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

Equation 1. Translate Transformation Matrix, week 3 lecture slides.

The x and y shift are inputted into the translate transformation matrix (equation. 1), this matrix is then used to convert the current image to match the template image. In this algorithm, the opencv inbuilt function cv2.warpAffine was implemented as this proved to be easier and quicker function to use to translate the desired image in the x and y direction.

Finally, after each template is matched the resulting image is generated. By placing the initial colour image and the resulting colour imaging next to each other we can clearly see the results of alignment. (figure 2.), here the blue layer is shifted (tx, ty) by (8, 10) and green by (5, 5).

Figure 2. 's2.jpg': left) original colour, right) after alignment

However, in some cases, the alignment did not produce great results. For input image 's1.jpg' the alignment made the original colour image worse (figure 3).

Figure 3. 's1.jpg': left) original, right) aligned

In the case of 's1.jpg', the blue was out of shift (x, y) by (1, 0) and the green resulted in (1, 5). This shift of 5 in the y direction caused the aligned result to be worse than the original. For the other

images s2 to s5, the algorithm performed well in alignment, but the first image s1 was the outlier in my program but this result is improved later, (will be discussed in Task 3).

Task 2

For Task 2, the idea is to implement the image pyramid where each layer is decreasing in resolution. Due to the nature of the method, initially it was decided to implement a recursive algorithm (pyramid() to call and pyramid_recur() is recursive). As the program goes down each step, inside pyramid_recur(), the functions cv2.pyrDown() is used, in this prebuilt function, a gaussian filter is applied followed by a reduction in resolution. The down sampled image (down_layer) is then passed into cv2.pyrUp() where an interpolation filter is applied followed by the increase of resolution. The resulting upsampled image (up_layer) is subtracted from the input image of the current layer to produce a residual image that is similar/same as a Laplacian filtered image.

However, there were issues with rescaling the image, since the alignment and measurement were taken at the lowest level with the lowest resolution, when upscaling the image, the residual Laplacian image from the previous layer could not be added, because this image has not been aligned. The first idea to solve this was just simply scale up the tx and ty (equation 1) and align the Laplacian images themselves at each step, when rescaling back to the original. This seemed costly and an even simpler solution would be just retrieving the tx, ty values, upscale the values at each layer (e.g. 2x larger for each layer) and transform the original image with the new retrieved translation matrix.

So by editing the initial alignment functions to create pyr_align() and pyr_alignment() just to return tx and ty values scored for the translation matrix, the concept ended up producing excellent results. Image ‘0054u’ ended up being perfectly aligned and ‘00911u’ having a small misalignment in the green layer but overall aligned well. Due to the resolution screen snippet was taken below (figure 4).

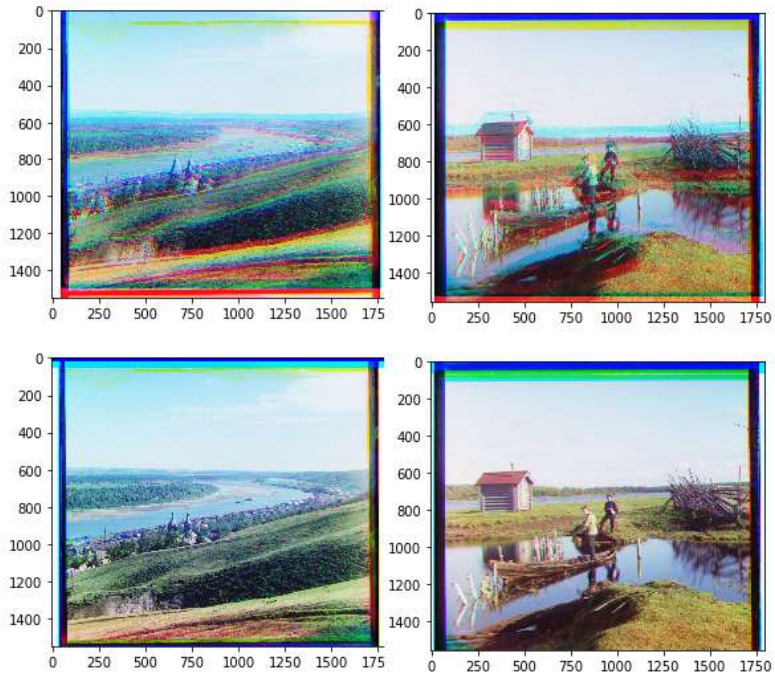

Figure 4. images after alignment with image pyramid left) '0054u.jpg', right) '00911u.jpg'

Task 3

After facing issues with several images, the improvement to algorithm design was to implement an edge detection filter such as Sobel, Laplacian or Canny to the cropped centre images used for template matching (figure 5). Here the function (edge_filter()) is applied in the alignment function, along with increasing search ranges to 30 pixels (30x30). After this addition, the results were outstanding. The images that had alignment issues were solved, since the numbers used in NCC were calculated using filtered images, that meant the pixels that contained higher numbers occurred at the edge pixels, thus the functions had an easier time calculating the values required.

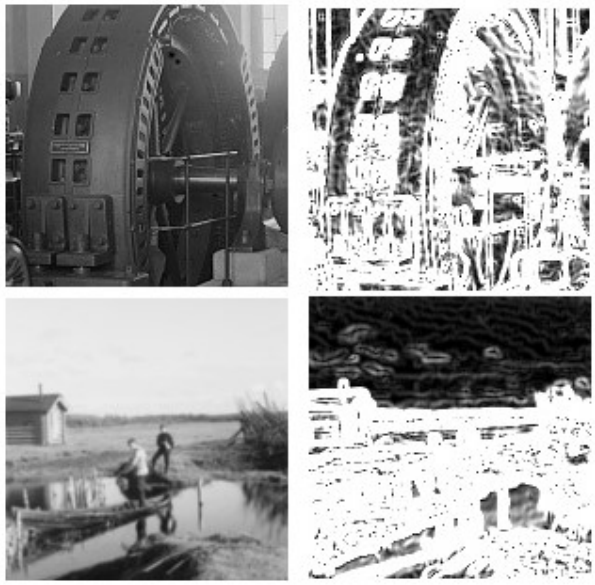

Figure 5. left) pre filter, right) filtered with Sobel

The figures below (figure 6 and figure 7) shows the improvement in each case, even with images like s2, we can see improvement from both the filter and increase in scan ranges to 30 pixels. Although the downside to increase search to 30 is the running times are indeed taking a fraction longer than a 20-pixel scan.

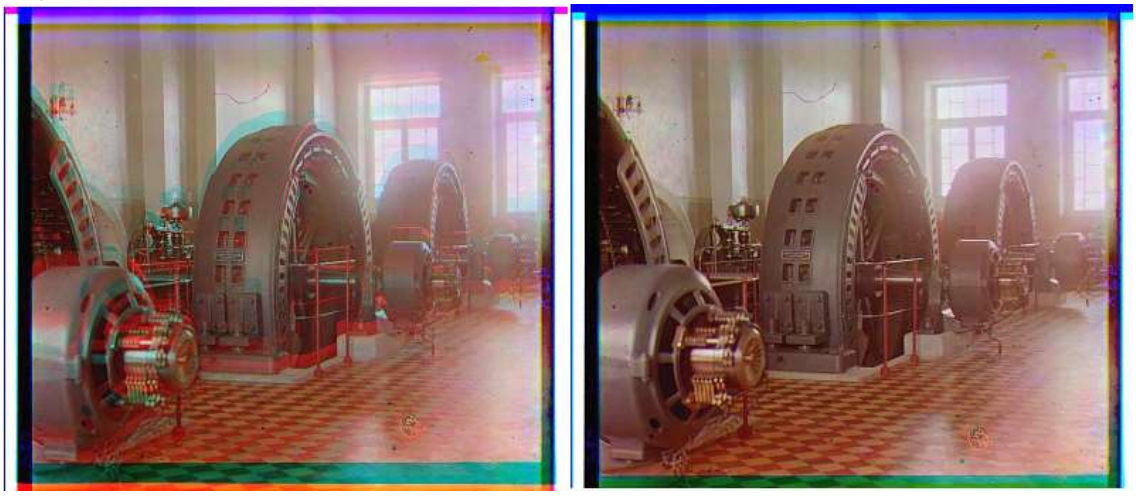

Figure 6. 's1.jpg', left) bad align, right) improved after filter application

Figure 7. 's2.jpg', left) 1st alignment, right) improved alignment

As we can see in figure 7, there are slight improvement in alignment, where in some areas (near the centre of the image), green layer can still be seen slightly out of alignment on the left compared to the right. Further examples are shown in figure 8 with additional images downloaded from the Library of Congress collection.

Figure 8. left) alignment done before use of edge filter, right) results after filter is used

In the end, it did not matter which edge detection filter was used, the results were an improvement from when not using an edge detecting filter. This meant the alignment is now perfect for every input rather than failing to work on a small number of outliers.

Additional improvements were tested, such as increasing contrast in gamma (brightness) but most pictures have a high brightness already so the actual functions were only used for testing in the end.