CS663 Assignment 1

Saksham Rathi, Kavya Gupta, Shravan Srinivasa Raghavan

August 2024

Contents

Question 1

In the first scenario, both the images differ in the pixel resolution but maintain the same aspect ratio (square pixels). We will use the motion model with the following features:

- Translation: To align the images based on the control points
- Rotation: To correct any angular misalignment between the images.
- Uniform Scaling: To account for the difference in pixel sizes (since both images have square pixels, the scaling factor will be uniform).

Such a transformation is sufficient in this case, because we don't need non-uniform scaling (both the pixels are square) and shearing.

In the second scenario, both the images have different pixel shapes (the second one is rectangular). We will have to use affine transformation in this case:

- Translation: To align the images based on the control points
- Rotation: To correct any angular misalignment between the images.
- Non-uniform Scaling: To account for the different scaling factors in the x and y directions.
- Shearing: To correct skewness.

This model can handle non-uniform scaling and shearing along with the features in the previous scenario, making it suitable aligning images with different pixel sizes.

Question 2

Relationship between the motion vectors

It can be understood as follows:

Given that the motion is purely translational:

- u_{12} represents the motion vector that aligns image I_2 with image I_1.
- u_{23} represents the motion vector that aligns image I_3 with image I_2.
- u_{13} represents the motion vector that aligns image I_3 with image I_1.

The relationship between these motion vectors can be described using vector addition:

$$u_{13} = u_{12} + u_{23}$$

This equation means that to align I_3 with I_1, you can first align I_3 with I_2 using u_{23}, and then align I_2 with I_1 using u_{12}.

Practical Consideration

In an ideal scenario (i.e., perfect conditions with no noise, distortion, or errors), this relationship would hold exactly in practice. However, in real-world applications, some distorting factor (most likely to be present in practical estimation) may cause this relationship to not hold.

1. **Noise:** Image data is often noisy due to sensor imperfections, lighting variations, or other environmental factors. This noise can lead to inaccuracies in the estimation of the motion vectors.
2. **Numerical Precision:** Computational limitations such as numerical precision and rounding errors can also introduce small deviations from the expected relationship.
3. **Pixels:** This motion between the images isn't completely continuous i.e. it's discrete, limited by the pixel size. Hence if the actual motion is having some fractional term (in terms of pixels), we have to round it up/down causing devation from the relationship.

Question 3

- Since we have access to the coordinates of both MATLAB's coordinate system and that of the image, we can compute the transformation matrix between the two coordinate systems. Since the graph is not rotated, the equations are fairly simple.
- The only possible relative motions are shifting of the orgin and non uniform scaling.

Therefore we have

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \mathcal{T} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \tag{1}$$

where $\mathcal{T} = \begin{pmatrix} c_1 & 0 & t_x \\ 0 & c_2 & t_y \\ 0 & 0 & 1 \end{pmatrix}$

The transformation represented by $\mathcal{T}$ can be solved for step by step by taking a few points from the image and mapping with the coordinates in MATLAB's graph. First step would be to solve for the origins of both the systems. This would give us the values of t_x and t_y. Then once that is done, we can compare the scales of both the systems. The scales are basically the values c_1 and c_2 and we are done.

Question 4

- There are 12 unknown parameters $A, B, C, D, E, F, a, b, c, d, e$ and f, therefore we would need at least $12/2 = 6$ control points.
- Let the pairs of physically corresponding control points be $\{(x_{11}, y_{11}), (x_{21}, y_{21})\}$, $\{(x_{12}, y_{12}), (x_{22}, y_{22})\}$, $\{(x_{13}, y_{13}), (x_{23}, y_{23})\}$, $\{(x_{14}, y_{14}), (x_{24}, y_{24})\}$. $\{(x_{15}, y_{15}), (x_{25}, y_{25})\}$ and $\{(x_{16}, y_{16}), (x_{26}, y_{26})\}$.

We are given that for any pair of corresponding points $\{(x_1, y_1), (x_2, y_2)\}$ the motion model is

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f \\ A & B & C & D & E & F \end{pmatrix} \cdot \begin{pmatrix} x_1^2 \\ y_1^2 \\ x_1 y_1 \\ x_1 \\ y_1 \\ 1 \end{pmatrix} \tag{2}$$

For each pair of control points that we have taken, there would be an equation similar to 2. The general format would look like this

$$\begin{pmatrix} x_{2k} \\ y_{2k} \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f \\ A & B & C & D & E & F \end{pmatrix} \cdot \begin{pmatrix} x_{1k}^2 \\ y_{1k}^2 \\ x_{1k} y_{1k} \\ x_{1k} \\ y_{1k} \\ 1 \end{pmatrix} \tag{3}$$

where $k \in \{1, 2, 3, 4, 5, 6\}$.

Combining all 6 eqautions we get,

$$\begin{pmatrix} x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} \\ y_{21} & y_{22} & y_{23} & y_{24} & y_{25} & y_{26} \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f \\ A & B & C & D & E & F \end{pmatrix} \cdot \begin{pmatrix} x_{11}^2 & x_{12}^2 & x_{13}^2 & x_{14}^2 & x_{15}^2 & x_{16}^2 \\ y_{11}^2 & y_{12}^2 & y_{13}^2 & y_{14}^2 & y_{15}^2 & y_{16}^2 \\ x_{11}y_{11} & x_{12}y_{12} & x_{13}y_{13} & x_{14}y_{14} & x_{15}y_{15} & x_{16}y_{16} \\ x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\ y_{11} & y_{12} & y_{13} & y_{14} & y_{15} & y_{16} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{4}$$

Let $X_2 = \begin{pmatrix} x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} \\ y_{21} & y_{22} & y_{23} & y_{24} & y_{25} & y_{26} \end{pmatrix}$, $X_1 = \begin{pmatrix} x_{11}^2 & x_{12}^2 & x_{13}^2 & x_{14}^2 & x_{15}^2 & x_{16}^2 \\ y_{11}^2 & y_{12}^2 & y_{13}^2 & y_{14}^2 & y_{15}^2 & y_{16}^2 \\ x_{11}y_{11} & x_{12}y_{12} & x_{13}y_{13} & x_{14}y_{14} & x_{15}y_{15} & x_{16}y_{16} \\ x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\ y_{11} & y_{12} & y_{13} & y_{14} & y_{15} & y_{16} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ and $\mathcal{M} = \begin{pmatrix} a & b & c & d & e & f \\ A & B & C & D & E & F \end{pmatrix}$. Equation 4 can be written compactly as

$$X_2 = \mathcal{M} \cdot X_1$$

$$\Rightarrow \boxed{\mathcal{M} = X_2 \cdot X_1^{-1}} \tag{5}$$

Therefore selecting control points such X_1 is invertible and solving equation 5 will provide the matrix $\mathcal{M}$ whose entries are the constants $A, B, C, D, E, F, a, b, c, d, e$ and f.

Question 5

(a) The code is in `Code/Q5_Code.m` and the rotated image is `Images/Q5/T2_rotated.jpg`.
(b) The code is in `Code/Q5_Code.m`.
(c) Here are the graphs:-

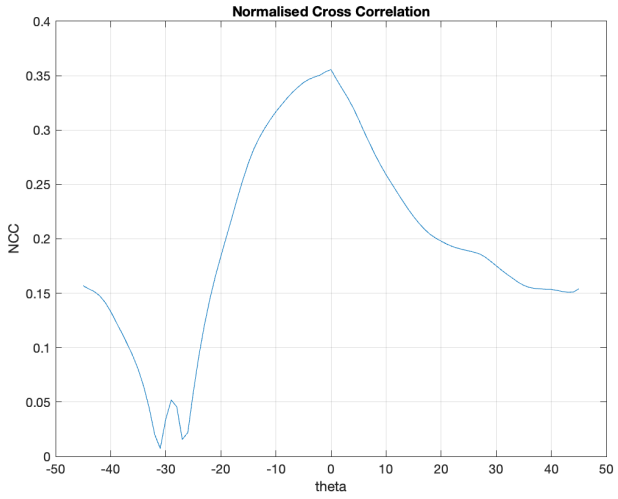

Figure 1: Normalised Cross Correlation (NCC) v/s θ

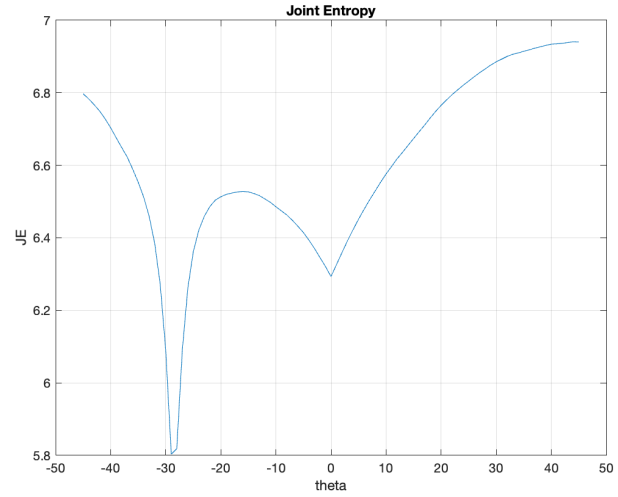

Figure 2: Joint Entropy (JE) v/s θ

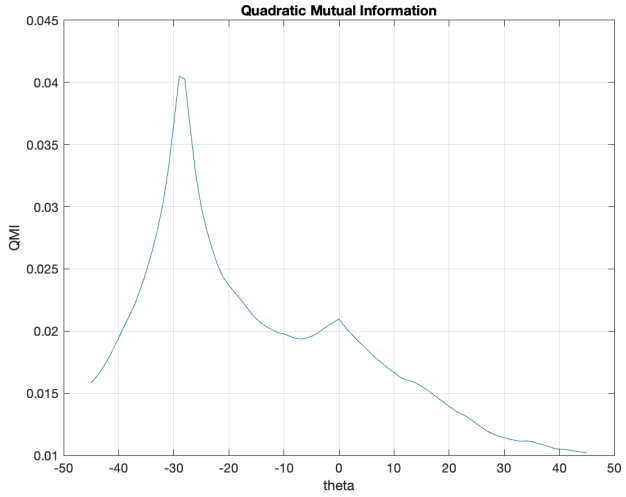

Figure 3: Quadratic Mutual Information (QMI) v/s θ

(d) Here are some observations:-

- **NCC:** More the absolute value of NCC, better the linear relation between the images. Global maxima exists at $\theta = 0°$, which is clearly not the correct rotation. There does exist a local maxima at around $\theta = -29°$ which we wanted. Hence it can be observed that NCC is not a good measure always to deal with image alignment.
- **JE:** Lesser the JE, better the alignment. We see that the global minima exists at $\theta = -29°$, close to our expected value (28.5° clockwise) upto precision of 1 degree, given the step size of theta is 1 degree too. Hence, JE checks out to be a good method for checking image alignment. We also observe a local minima at $\theta = 0°$.
- **QMI:** More the QMI, better the alignment. We see that the global maxima exists at $\theta = -29°$, close to our expected value (28.5° clockwise) upto precision of 1 degree, given the step size of theta is 1 degree too. Hence, QMI checks out to be a good method for checking image alignment. We also observe a local maxima at $\theta = 0°$.

Here are the optimal rotations:-

Measure	Optimal θ
NCC	$0°$
JE	$-29°$
QMI	$-29°$

(e) The code is in `Q5_Code.m` and the joint histogram for $\theta = -29°$:-

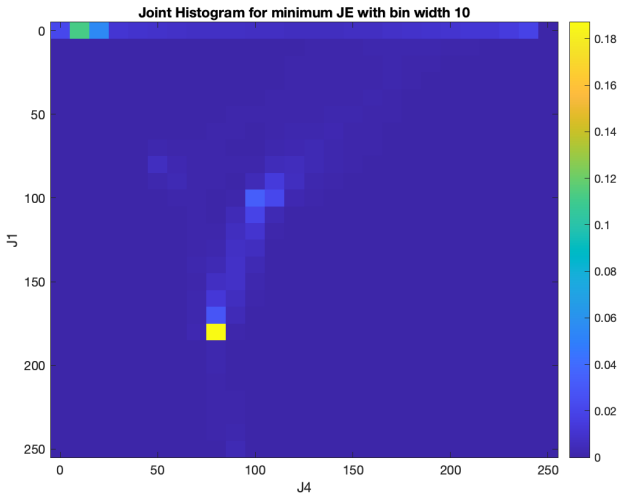

Figure 4: Joint Histogram for $\theta = -29°$

(f) When two discrete random variables I_1 and I_2 are statistically independent, then $p_{I_1 I_2}(i_1, i_2) = p_{I_1}(i_1) \times p_{I_2}(i_2)\ \forall\ i_1, i_2$ (in domain of the variables), i.e. joint PMF is equal to product of the marginal PMFs. In that case, QMI would be equal to 0. Hence the more the difference between $p_{I_1 I_2}(i_1, i_2)$ and $p_{I_1}(i_1) \times p_{I_2}(i_2)$, the more dependent the variables are. Hence the more aligned (correlated) the images are, QMI tends to be higher. Hence images are aligned when QMI is maximum.

Also unlike NCC, QMI can detect non-linear relation between variables.

Question 6

The code for this question is in "Code/a1_q6.m".
(a) The images were read using the imread function (and cast as double). Then we took 12 pairs of point as input from the user (through mouse click). The points were choosen to be as accurate and similar as possible. The values were saved in "selected_points.mat" to save effort for further runs. Here is a small section of the code:

```
n = 12; % Number of points
for i=1:n
    figure(1);
    imshow(im1/255);
    [x1(i), y1(i)] = ginput(1);
    figure(2);
    imshow(im2/255);
    [x2(i), y2(i)] = ginput(1);
end
save('selected_points.mat', 'x1', 'y1', 'x2', 'y2');
```

(b) We consider that the affine transformation matrix will be of the form:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ 0 & 0 & 1 \end{bmatrix}$$

We initialize matrix A with points from Img1 (n = 12 in our case):

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ y_{11} & x_{12} & y_{13} & \dots & y_{1n} \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

Similarly, we initialize matrix b with points from Img2:

$$\begin{bmatrix} x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ y_{21} & x_{22} & y_{23} & \dots & y_{2n} \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

The equation which we need to solve finally is (using least squares approach, since these are not square matrices):

$$TA = b$$

(c) In this part, we used nearest neighbour interpolation for image warping. We used reverse warping (which gives better results than forward warping). We initialized a matrix of size same as Img2 with zeroes. We then iterated over the points and calculated the their corresponding point using the inverse of matrix T (already obtained from previous part). Then, the values obtained were rounded, and also, we considered only those values which were in bounds. Here is the code:

```
warpedImg = zeros(size(im2), 'uint8');
for r = 1:size(im2, 1)
    for c = 1:size(im2, 2)
```

```
        originalCoords = invT * [c; r; 1];
        xOriginal = originalCoords(1);
        yOriginal = originalCoords(2);
        xNearest = round(xOriginal);
        yNearest = round(yOriginal);
        if xNearest >= 1 && xNearest <= cols && yNearest >= 1 && yNearest <= rows
            warpedImg(r, c, :) = im1(yNearest, xNearest, :);
        end
    end
end
```

Here are all the three images displaeyed:

Figure 5: Image warped using nearest neighbour interpolation

As we can see, the third image looks quite similar and aligned to the second image (except the black border, which comes because of out of bound values when multiplying by inv(T)).

(d) In this part, we used bilinear interpolation for image warping. The approach looks similar to the previous part, but instead of mapping the intensities to nearest integral points, we consider the weighted average of all 4 neighbouring points. And the weights are the areas of the four rectangles, here is the code:

```
    for r = 1:size(im2, 1)
    for c = 1:size(im2, 2)
        originalCoords = invT * [c; r; 1];
        xOriginal = originalCoords(1);
        yOriginal = originalCoords(2);
        x1 = floor(xOriginal);
        y1 = floor(yOriginal);
        x2 = x1 + 1;
        y2 = y1 + 1;
        dx = xOriginal - x1;
        dy = yOriginal - y1;
        pixelValue = zeros(1, channels);
        if x1 >= 1 && x1 <= cols && y1 >= 1 && y1 <= rows
            pixelValue = (1-dx)*(1-dy)*double(im1(y1, x1, :));
        end
        if x2 >= 1 && x2 <= cols && y1 >= 1 && y1 <= rows
            pixelValue = pixelValue + dx*(1-dy)*double(im1(y1, x2, :));
        end
        if x1 >= 1 && x1 <= cols && y2 >= 1 && y2 <= rows
```

```
            pixelValue = pixelValue + (1-dx)*dy*double(im1(y2, x1, :));
        end
        if x2 >= 1 && x2 <= cols && y2 >= 1 && y2 <= rows
            pixelValue = pixelValue + dx*dy*double(im1(y2, x2, :));
        end
        warpedImg(r, c, :) = uint8(pixelValue);
    end
end
```

Figure 6: Image warped using bilinear interpolation

Similar to the previous case, the image aligns really well.

(e) We already know that A is of the form:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ y_{11} & x_{12} & y_{13} & \dots & y_{1n} \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

and b is of the form:

$$\begin{bmatrix} x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ y_{21} & x_{22} & y_{23} & \dots & y_{2n} \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

It has also been given that the control points are perfectly colinear i.e. $y_{1i} = mx_{1i} + c$.

$$b = TA$$

$$bA^T = TAA^T$$

where A^T is the transpose of matrix A. We can find T if AA^T is invertible. $P = AA^T$ is given by:

$$\begin{bmatrix} \sum_{i=1}^{n} x_{1i}^2 & a\sum_{i=1}^{n} x_{1i}^2 + b\sum_{i=1}^{n} x_{1i} & \sum_{i=1}^{n} x_{1i} \\ a\sum_{i=1}^{n} x_{1i}^2 + b\sum_{i=1}^{n} x_{1i} & a^2\sum_{i=1}^{n} x_{1i}^2 + 2ab\sum_{i=1}^{n} x_{1i} + kb^2 & a\sum_{i=1}^{n} x_{1i} + nb \\ \sum_{i=1}^{n} x_{1i} & a\sum_{i=1}^{n} x_{1i} + nb & n \end{bmatrix}$$

If C_1, C_2, C_3 are the columns of the matrix P, then we can see from above that $C_2 - aC_1 - bC_3 = 0$. Hence the matrix is not invertible (rank-deficient). Hence, we will not be able to find T using this method. Therefore, affine transformation cannot be used if we choose co-linear points.