EA-1 Lab Project: Linear Transformations

This part of the Lab Project will focus on Linear Transformations. You will learn how to apply Linear Transformations in order to apply fun warping effects on images. Furthermore, you will test how warping can affect face recognition, based on the student image database we constructed in the Lab Introduction. We remind here that all material from the three parts of the Lab Project should be submitted together. There will be a single due date for the Lab Project during the last week of classes.

1 Digital Images in Matlab

Intuitively, a gray scale image in its digital form can be viewed as a matrix of *pixels*. Let's denote an image by I, which has R rows and C columns (i.e., it is a $R \times C$ matrix). Each pixel $I(u, v)$ is an element in this matrix where u indexes its row while v indexes its column. Therefore, $I(u, v)$ records the *pixels value* or *intensity* at location (u, v). To make things easier, we choose this coordinate system to retrieve pixels, and this matches with Matlab's (row,column) indexing. For example, the top-left corner pixel of the image is $I(1, 1)$, which is the first element on the first row in the matrix; the bottom-right corner pixel is $I(R, C)$, which is the last element on the last row in the matrix. For more information on image manipulation functions, please refer to the Introduction of the Lab Project.

2 Image Transformations as Linear Transformations

2.1 Warping a Triangular Region

You are given:

- The coordinates of three vertices of a triangle (u_1, v_1), (u_2, v_2) and (u_3, v_3). They define a triangular region in the original image I_1.
- The coordinates of the new locations of these three vertices (u'_1, v'_1), (u'_2, v'_2) and (u'_3, v'_3). They define the targeted distorted (or transformed) region in the transformed image I_2.

We need to solve:

As illustrated in Figure 1, we need to find the linear transformation A that maps the distorted image I_2 to the original image I_1, i.e., for any pixel location (u', v') in the triangular region in I_2 we need to determine its corresponding location (u, v) in I_1, so that we can copy the pixel value $I_1(u, v)$ to $I_2(u', v')$.

Solution: Assume the transformation has the form,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} u' \\ v' \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}, \tag{1}$$

which is called an *affine transform*. It has 6 parameters which need to be determined. Due to the addition of the vector $\begin{bmatrix} e & f \end{bmatrix}^T$, it is not linear, but we can make it a linear transform by simply

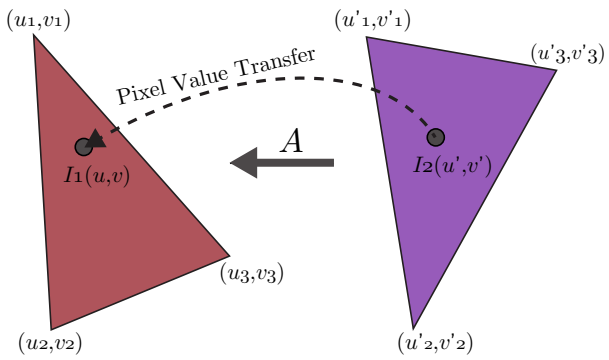

Figure 1: Solving the transformation based on the three pairs of correspondences of the vertex corrdinates, i.e., (u_i, v_i) in the original triangle I_1 corresponds to (u_i', v_i') in the new triangle I_2, where $i = \{1, 2, 3\}$. We need to determine the matrix that transforms (u_i', v_i') to (u_i, v_i). Once this is done, for any pixel in the triangle in $I_2(u', v')$, we can easily find its corresponding pixel in $I_1(u, v)$.

augmenting the 2 dimensional input vector $\begin{bmatrix} u' & v' \end{bmatrix}^T$ to be a 3 dimensional vector $\begin{bmatrix} u' & v' & 1 \end{bmatrix}^T$. Therefore we can obtain the following linear transformation,

$$\begin{bmatrix} a & b & e \\ c & d & f \end{bmatrix} \begin{bmatrix} u' \\ v \\ 1' \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix} \rightarrow A \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}, \text{ where } A = \begin{bmatrix} a & b & e \\ c & d & f \end{bmatrix}. \tag{2}$$

The transformation matrix A in equation (2) has 6 parameters. Now we need to solve for the parameters, based on three pairs of vertex correspondences. The correspondences lead to the following set of equations and solutions:

$$A \begin{bmatrix} u_1' & u_2' & u_3' \\ v_1' & v_2' & v_3' \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \rightarrow A = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \left(\begin{bmatrix} u_1' & u_2' & u_3' \\ v_1' & v_2' & v_3' \\ 1 & 1 & 1 \end{bmatrix} \right)^{-1}. \tag{3}$$

Once the transform (or the *affine warp*) is obtained, for each pixel (u', v') in the distorted image, we can locate its corresponding pixel (u, v) in the original image, and copy the pixel value at (u, v) in I_1 to the pixel location of (u', v') in I_2. One example is shown in Figure 2.

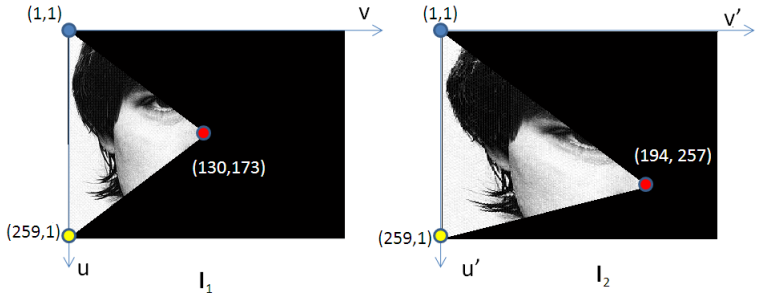

Figure 2: An example of warping a triangular image region by only offsetting one vertex. The three vertices of the original triangle are $(1, 1)$, $(259, 1)$, and $(130, 173)$. Their corresponding vertices in the new triangle are $(1, 1)$, $(259, 1)$ and $(194, 257)$.

2.2 Image Warping

There are many different ways to warp an entire image. Here, we will try a simple case. Suppose the image is divided into 4 triangles as shown in Figure 3. One can specify the distortion by moving the center point P to a new position P_{new}, which will distort the 4 triangles as shown in Figure 3. For example, triangle ACP is transformed to ACP_{new}. Then one can compute 4 different transforms to warp the 4 triangles, respectively. This partitions the complex and nonlinear transformation of the entire image into four simpler and linear transformations.

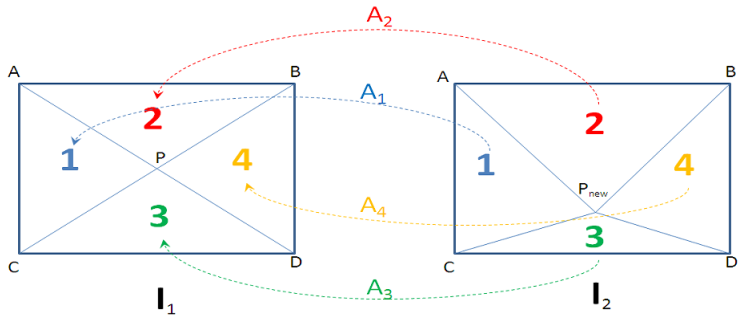

Figure 3: A simple image warping. Offsetting the image center creates the distortion of 4 triangles.

3 Problem Description

3.1 Goal of this Project

The goal of this part of the Lab Project is to create an algorithm for warping/unwarping of images based on the technique discussed in Section 2, i.e., moving the central point of the image at different locations. Furthermore, the warping/unwarping functions will be tested on the student images and the database created at the Introduction of the Lab Project to study how face recongition can be affected due to the images being distorted.

We provide you with the following:

- Function `transformStudentImages(x,studentNumber,numTransform)` which performs a set of `numTransform` warping realizations of the image in `x` at a set of points which are selected at random based on the specific `studentNumber`. The function returns as outputs the matrix `transformedStudentImages` which contains the warped images vectorized in its columns and the matrix `warpPoints` whose columns represent the points that the central point of the image has been moved to, at each warping realization.

- Script `LabProject.m` which contains the main script you would need to run for this Lab Project. Note that this is an extension of the same file provided to you at the Introduction part of the Lab Project. Except for selecting the student number that corresponds to you, you **should not** make any other changes to this script. The script has been designed to run up to different sections once you have written the appropriate functions, as described in Section 4.

- Finally, the function `identifyImages()` which uses the results of the previous steps of the algorithm to summarize the findings in a nice figure.

Note: Functions `transformStudentImages()` and `identifyImages()` are provided in `.p` file format. This format is essentially the same as a MATLAB `.m` file but cannot be accessed for editing or viewing.

The main steps of the algorithm are:

1. Warp the student image moving its central point to `numTransform` different locations selected at random.
2. Unwarp each warped image by moving the `warpPoint` back to the center of the image. The image might be significantly distorted if the performed warping was severe.
3. Identify each warped image with a student in the `correctedDatabase` created in the Introduction part of the Lab Project.
4. Identify each unwarped image with a student in the `correctedDatabase` created in the Introduction part of the Lab Project.
5. Present the identification results before and after unwarping the images.

3.2 To Do Items

1. Copy and paste the new provided files in the same folder where you have the files for the Introduction part of the Lab Project. If you wish, for the file `LabProject.m`, you might only copy and paste the new part of the code under the `LAB PROJECT - PART 1 - TRANSFORMATIONS`. If, instead, you choose to copy and replace the whole file, you will again need to set the `studentNumber` variable to the number that corresponds to your student image from the folder `Student_Images`.
2. Try to run the provided script `LabProject.m`. You will notice that you get a set of `warnings`. This is due to the fact that some functions are missing in order for the code to run properly.
3. Implement and test the functions discussed in Section 4 one-by-one and test the script `LabProject.m` until all the warnings dissappear. Your whole script should only run completely (warning/error-free) once you have implemented all the functions correctly.
4. Once you have finished writing all the functions test the whole script `LabProject.m`. You should get a big cumulative figure as an output. The figure will show 4 rows of images. The first one depicts 8 transformed images of the same student, the second one represents the face recognition results when the transformed images are used as input, the third one represents the corresponding unwarped images while the last one shows the face recognition results when the unwarped images are used as input. An example is presented in Figure 4.

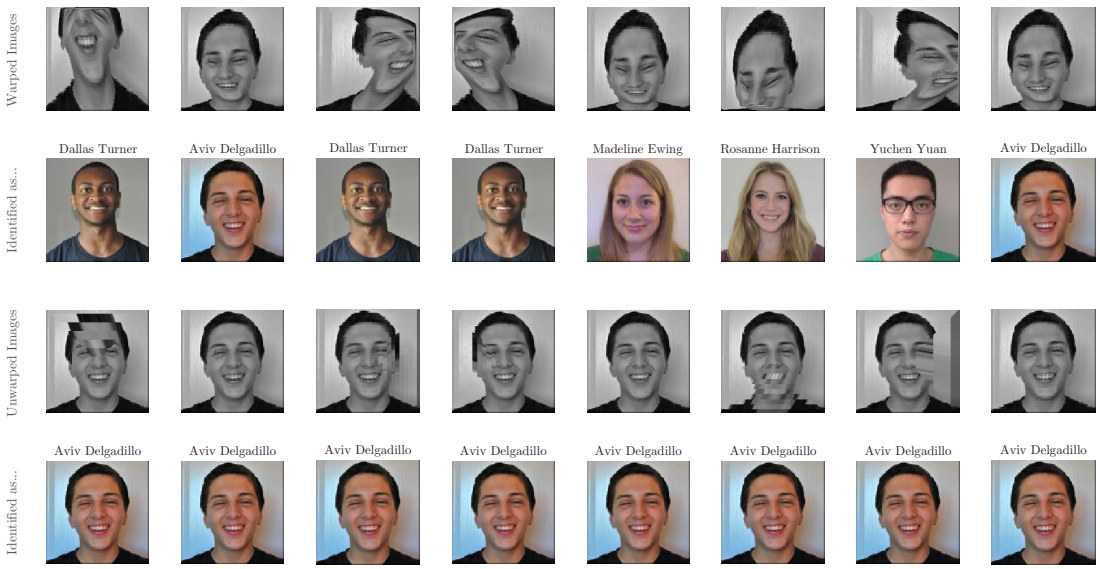

Figure 4: Example of final face recognition results before and after unwarping.

4 Set of Functions to be Implemented

You will create a set of functions to perform the warping operation as well as see how warping affects face recognition.

4.1 Creating the Warping Mask

- `function y = evaluateLine(x,point1,point2)`

  This function acceps a value `x` and the two-element column vectors `point1` and `point2` denoting the coordinates of two points in the $\mathbb{R}^2$ space. In this function you should evaluate the `y` coordinate at coordinate `x` for a line that passes through the two points, `point1` and `point2`.

  A systematic way to do this would be to solve the linear system of equations,

$$A\mathbf{x} = \mathbf{b} \rightarrow \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix},$$

  where (x_1, y_1) and (x_2, y_2) are the coordinates of the two points that the line passes through and the equation of the line is represented by $y = ax + b$.

- `function [mask ,rows, columns] = createMask(R,C,warpPoint)`

  This functions accepts the number of rows, `R`, and the number of columns, `C`, of an image and a two element vector `warpPoint` denoting the coordinates of a point that would divide the image in 4 triangles when combined with the 4 corners of the image.

  This function should label each pixel in the image with 1, 2, 3 or 4 depending on which of the four triangles it belongs to and should return this result in the `mask` output matrix. At the same time, it should return a list of all the row and column coordinates, for all the pixels in the image, in the corresponding output vectors `rows` and `columns`.

  In this function, you should:

1. Check whether the `warpPoint` is within the image's bounds, otherwise report an error. **Note**: For the warping to work you should consider points lying at the outer limits of the image (i.e., first and last rows as well as first and last columns of the image) as **out of bounds**.
2. Create 4 two-element column vectors `UL`, `LL`, `LR` and `UR` which store the coordinates of the four corners of the image, respectively, Upper-Left, Lower-Left, Lower-Right and Upper-Right.
3. Loop through all the image pixels and check which triangle they belong to. You should use the `evaluateLine` function created above to determine whether each point falls below or above a set of lines that form the triangle and make the decision accordingly. Label the pixels based on the presented example in Figure 3.
4. While you loop through the image pixels, store each and every row and column coordinate of each pixel in the output vectors `rows` and `columns`. At the end of your function the elements of each of these vectors should be equal to the number of pixels in the image.

Note: Be very careful with which side of the lines you consider as positive and which as negative. Note the axis in Figure 2 (the y coordinate is positive in the downward direction).

Before you proceed, test your function using:

```
[mask ,rows, columns] = createMask(100,150,[20, 50]); figure; imshow(mask,[]);
```

The resulting figure should look like the one of Figure 5. If the shading of each region is different, it is ok. It just means that you assigned different labels (numbers) to each region.

Figure 5: Example resulting mask when using the commands `[mask ,rows, columns] = createMask(100,150,[20, 50]); figure; imshow(mask,[]);`.

4.2 Image Warping

- `function A = solveWarp(TStart,TEnd)`

  This function accepts two 2×3 matrices `TStart` and `TEnd` holding the three vertices of two triangles for which we want to determine the transform `A` which transforms the one to the other. This function should:

  1. Augment the `TEnd` matrix with a row of ones as in equation (3).
  2. Solve for the linear transformation `A` using equation (3).

- `function [startCoor, endCoor] = ... transformCoordinates(A, rEndCoor, cEndCoor, R, C)`

  This function accepts a transformation matrix `A`, a column vector of row coordinates `rEndCoor`, a column vector of column coordinates `cEndCoor` and the number of rows `R` and columns `C` of the image. It returns two vectors of linear (1-D) coordinates storing the `startCoor` coordinates in the initial image and the `endCoor` coordinates in the transformed image.

  This function should:

  1. Create a matrix `endCoor` which has the `rEndCoor` as the first row, `cEndCoor` as the second row and a third row of all ones. Note that this corresponds to the set of all points represented by the input coordinates in matrix form, as in equation (3).
  2. Apply the transformation to get the new coordinates `startCoor = round(A*endCoor)`. The rounding is necessary so that we avoid non-integer values for the pixel locations.
  3. Prune away values in the `startCoor` matrix that fall outside the bounds of the image by setting them to 1, R or C, respectively.
  4. Transform the coordinates from 2-D to 1-D (remember that a matrix can be indexed as $A(i, j)$ but also as $A(i)$. For our purposes, 1-D indexing makes everything simpler. In order to transform the coordinates from 2-D to 1-D you can do the following:
     - `startCoor = sub2ind([R,C], startCoor(1,:), startCoor(2,:));`
     - `endCoor = sub2ind([R,C], endCoor(1,:), endCoor(2,:));`

- `function unwarpedImage = imageWarp(img, warpStartPoint, warpEndPoint)`

  This function accepts the image `img` as an input as well as the initial and final positions of the point that is being moved to perform image warping. Both `warpStartPoint` and `warpEndPoint` are two-element column vectors representing the two points.

  This function should:

  1. Get the size of the input image `img`.
  2. Create the same 4 two-element column vectors which store the coordinates of the four corners of the image (as in the function `createMask()`).
  3. Create 4 matrices representing the set of vertices that correspond to the triangles that describe the image before and after warping. The matrices should be of size 2×3 containing the vectors for each vertex on a triangle (follow the notation of equation (3)).You should construct 4 matrices for the triangles before the warping and 4 matrices for the triangles after the warping. These matrices will be fed to the function `solveWarp` in order to determine the transformation matrix for each pair of transformed triangles.
  4. Create the warping mask using the function `createMask()` with the `warpEndPoint` as input.
  5. Find the transformation (warping) matrix for each one of the four pairs of transformed triangles using the function `solveWarp`. Store each transformation matrix in a separate variable.
  6. Transform each pixel location in the resulting image to its corresponding pixel location of the initial image using the transformation matrices you calculated above. This step is a little tricky. You can follow the **Hints** below:
     - Create a `for-loop` to loop through the 4 different areas (triangles) in the image.

- For each triangle set the transformation matrix `A` to be the appropriate one from the ones calculated above. The mask that was constructed using the `createMask()` function will give you this information through the label it has stored per pixel.
- Isolate only the rows and columns that correspond to the current triangle. Use logical indexing on the warping mask as well as on the vectors `rows` and `columns` that were provided by the `createMask()` function.
- Use the function `transformCoordinates` to transform the coordinates within the current triangle from the resulting image to the ones of the initial image.
- Get the pixel values of the initial image and place them to their transformed locations in the resulting image. Note that using the 1-D coordinates calculated by the function `transformCoordinates` makes the assignment of pixel values from the one image to the other much easier.

Note: The above steps are just a suggestion for faster implementation. If you find this implementation hard to understand you can just loop through all the pixels in the image and perform the corresponding transform, based on the labels stored in the warping mask.

Before you proceed, check your function using:

```
x = im2double(imread('jobs_small.jpg'));
wrapStartPoint = [ceil((size(x,1)+0.5)/2); ceil((size(x,2)+0.5)/2)]; % Center
wrapEndPoint   = [50,100]; % Random end point.
y = imageWarp(x,wrapStartPoint,wrapEndPoint);
z = imageWarp(y,wrapEndPoint,wrapStartPoint);
figure; subplot(1,3,1); imshow(x,[]); title('Original Image');
subplot(1,3,2); imshow(y,[]); title('Warped Image');
subplot(1,3,3); imshow(z,[]); title('Unwarped Image');
```

The resulting figure should look like the one of Figure 6. From the above script you can see that the function `imageWrap()` can perform both warping or unwarping by interchanging the role of the `warpStartPoint` and `warpEndPoint`.

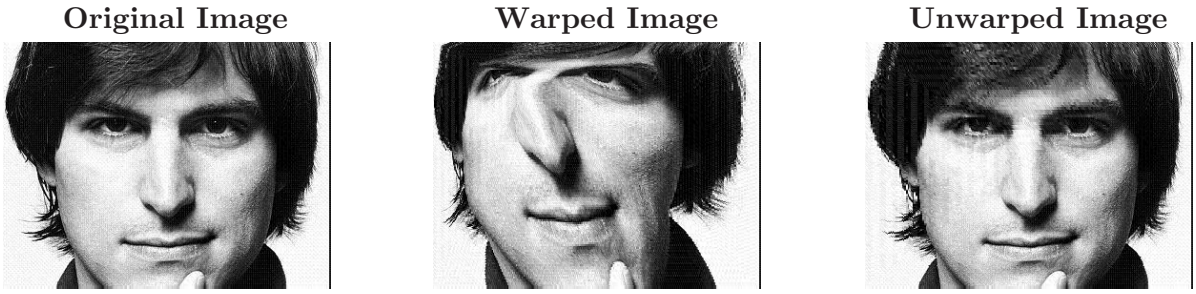

Figure 6: Example of warping and unwarping using the provided image `jobs_small.jpg`.

4.3 Combining Image Warping and Face Recognition

- `function unwarpedStudentImages = ... unwarpTransformedImagesToCenter(transformedStudentImages,warpPoints,imgSize)`

  This function accepts the matrix `transformedStudentImages` and `warpPoints`, provided by the function `transformStudentImage()`, as inputs as well as the 2 element row vector `imgSize` holding the imageSize. It returns the matrix `unwarpedStudentImages` with all the images unwarped.

  This function should:

  1. Initialize the `unwarpedStudentImages` matrix to a matrix of all zeros of the same size as `transformedStudentImages`.
  2. Loop through all the columns of the matrix `transformedStudentImages`.
  3. Create an image of size `imgSize` from each column. You can use the `createMatrix()` function from the Introduction of the Lab Project.
  4. Unwarp the image using the function `imageWarp()` selecting as start point the corresponding point in the `warpPoints` matrix and as ending point the center of the image. The center can be calculated using the expression in the warping/unwarping example above.
  5. Vectorize the unwarped image and place it in the corresponding column of the resulting matrix `unwarpedStudentImages`.

- `function minErrorPos = associateImagesWithDatabase(images, correctDatabase)`

  This function accepts as inputs a matrix `images` with vectorized images in its columns and a database of images `correctDatabase`, which should be the one you created in the Introduction of the Lab Project. This function should loop through all the images in the `images` matrix and find the best match (column) for this image in the database. You can use the function `findMinimumErrorPosition()` you created in the Introduction of the Lab Project. The output should be a vector holding the minimum error position (column) for each image in the matrix `images`.