
Out: 04 / 15 / 2019
Due: 05 / 02 / 2019 (deadline: midnight)

Late submissions: Late submissions result in 10% deduction for each day. The assignment will no longer be accepted 3 days after the deadline.

Office hours:

		Mon	Tue	Wed	Thu
Guido Gerig	Office 10.094				
Andrew Dempsey	ad4338@nyu.edu	10-12am			
Anshul Sharma	as10950@nyu.edu		10-12am		
Bhavana Ramakrishna	br1525@nyu.edu			10-12am	

Location: cubicle spaces in 2 Metrotech, 10.098 A, B, D, E, H

Linear image transformations

The purpose of this assignment is to learn about linear geometric image transformations. Please read following instructions carefully.

A) Affine Image Transformation

Consider the 2D linear transformation from source image (x,y) to target (x',y') as: $x' = a11\,x + a12\,y + a13$

$$y' = a21\,x + a22\,y + a23$$

and its formulation in homogeneous coordinates as:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \quad \text{or simplified } \bar{x}' = [A]\bar{x}\,.$$

Recall the discussions on reasoning why to replace a forward by a backward transforms. This implies that given the parameters for a transformation [*A*], you actually perform the inverse process from the target raster (x',y') back to the source image: $\bar{x} = [A^{-1}]\bar{x}'$. Doing so, you scan the empty target image buffer (x',y') pixel-by-pixel and calculate the corresponding pixel location and associated intensity in the target image (x,y).

Affine transformation with resampling

Implement a program that transforms an image given an affine transformation (6 parameters), which includes rotation, translation, scaling and shear. Please note that the transformation is generally applied from the target image backwards to the source image, i.e. you step through each pixel of the new target image, determine the position in the source image, and take/calculate the intensity at this pixel to be transferred to the target image. Two types of interpolations need to be implemented:

1) Nearest neighbor (NN): Take the intensity from the pixel which is closest to the non-grid position of the transformation. Please note that this can be achieved by rounding a non-grid (x,y)-ccordinate to the next integer coordinates, which are pixels that exist in your source image.
2) Implement bilinear interpolation from the 4 neighbors of the non-grid coordinate, following the discussion in the course and the solution from your theoretical HW3.

Take an input image of your choice, choose a transformation, and apply it via backward transform. Best for debugging is not to use huge images but to limit/crop to a reasonable size for efficient computation.

1) Define an affine transformation with 6 parameters via a combination of translation, rotation, and scaling, and derive the combined transformation matrix [*A*]. Note that the translations in x and y and scaling should be relatively small, otherwise you would move the image out of the visible window.
2) Invert the matrix [*A*] to be used for the backtransform.
3) Implement the backtransform by defining an empty buffer for the target raster image of the same size as the source image. Then loop for each pixel of the target image (x',y'), calculate the corresponding (x,y) with applying $\bar{x} = [A^{-1}]\bar{x}'$.
4) Apply both, NN and bilinear interpolation to your image of choice.(*)
5) For the affine transformation, demonstrate the differences between nn and bilinear, best by large zooming of an image subregion.

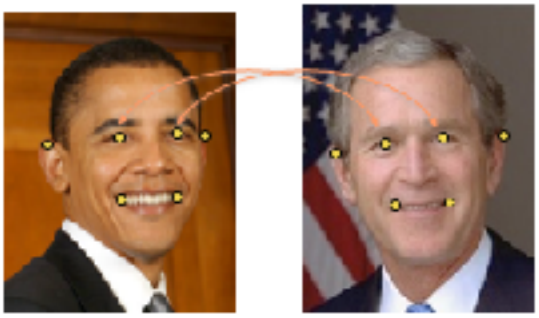
Provide a short description of your results and observed differences between NN and bilinear.

(*) Hint: By transforming an image, the transformation may move parts outside of the original image size. It is suggested to keep the target image the same size, and to paint pixels that have no partner pixel in the source image as black (i.e. starting with an empty target image).

Original, and rotation with NN and bilinear interpolations.

B) Calculation of affine (linear) transform from landmarks}

As discussed in the course, a transformation can be determined based on a set of corresponding landmarks in a source and a target image. A minimum of 3 points with (x,y) coordinates is required, but more landmarks result in a more stable solution by solving an overconstrained linear equation system (**).



$$\bar{c}_1, \bar{c}'_1$$
$$\bar{c}_2, \bar{c}'_2$$
$$\bar{c}_3, \bar{c}'_3$$
$$\bar{c}_4, \bar{c}'_4$$
$$\bar{c}_5, \bar{c}'_5$$
$$\bar{c}_6, \bar{c}'_6$$

Sets of corresponding landmarks to calculate an affine transformation between a pair of images.

1) Use or implement a module that gives you pixel positions by clicking at locations with the mouse.
2) Use this module to create sets of corresponding pixel positions in a source and a target image.
3) Set up the linear equation system and implement a solution to solve for the affine transformation between the source and target image:
   3a) For only 3 pairs of (x,y) landmarks, you can directly calculate the transformation via six equations for six unknowns of the affine transformation.
   3b) Using more than 3 pairs of landmarks, you create an overconstrained equation system that has to be solved via least squares solution. A standard solution is using a pseudoinverse or SVD to solve estimate the inverse, a discussion is found in the lecture slides (**).
4) Apply the transformation and check for the correctness of the result by displaying source, target and resulting images side by side. You can even create another result by blending the result and source together (e.g. adding the images) to have some visual check of geometry differences.

As test images, you can use own pictures (e.g. of frontal view of faces of humans or animals or whatever you like). Would such pictures not be available, you can search the web (e.g. http://www.face-rec.org/databases/). You can be creative about the choice of images and best use your own.

(**) Please see additional materials in regard to solving a linear equation system for solving overconstrained systems which are systems that have more equations than unknowns so that the rectangular Matrix A cannot be inverted. E.g, for Matlab using Matlab help: for solving the system ***A.x=b*** in Matlab, there are the choices of ***x=A/b*** (fastest), or ***x=pinv(A)*b***, or ***x=inv(A'*A)*A'*b).*** For python users **Ax=B** , or **x=np.linalg.inv(A)*B**