BV1

Nik Brouw, Tim Stolp

April 2019

2 Theory

2.1

Nearest Neighbour for point x: $F(floor(x+0.5))$

2.2

Solve for "a" and "b":

a

"a" is the slope of the plot.

$$a = \frac{\delta y}{\delta x}$$

$$a = \frac{F(k+1) - F(k)}{k - k + 1}$$

$$a = F(k+1) - F(k)$$

b

From $y = ax + b$ we derive $b = y - ax$. Take a point (x,y) and fill in the formula.

$$b = F(k) - k(F(k+1) - F(k))$$

$$b = F(k) - kF(k+1) + kF(k)$$

$$b = (1+k)F(k) - kF(k+1)$$

2.3

These two equations can be written in matrix form:

$$\begin{bmatrix} k & 1 \\ k+1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} F(k) \\ F(k+1) \end{bmatrix}$$

We can solve (a,b) by doing:

$$\begin{bmatrix} k & 1 \\ k+1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} F(k) \\ F(k+1) \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

Which ends up as:

$$\begin{bmatrix} -1 & 1 \\ k+1 & -k \end{bmatrix} \begin{bmatrix} F(k) \\ F(k+1) \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

Solving gives

$$a = F(k+1) - F(k)$$
$$b = (k+1)F(k) - kF(k+1)$$

2.4

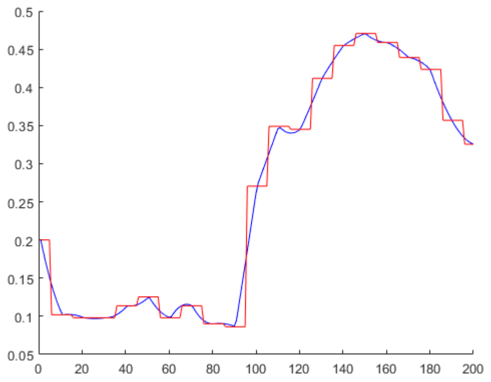
If we apply the previous function twice for x and once for y, we can substitute and as a result receive $f_1(x,y)$. In the following formulae, $a = x-k$ and $b = y-l$

$$f_1(x,l) = (1-a)F(k,l) + aF(k+1,l)$$
$$f_1(x,l+1) = (1-a)F(k,l+1) + aF(k+1,l+1)$$
$$f_1(x,y) = (1-b)F(x,l) + bF(x,l+1)$$

Substituting the first and second formula into the third gives:

$$F_1(x,y) = (1-b)(1-a)F(k,l)+(1-b)aF(k+1,l)+(1-a)bF(k,l+1)+abF(k+1,l+1)$$

2 Matlab



The blue line is bi-linear interpolation and the red line is nearest neighbour interpolation. The bi-linear interpolation seems to be much smoother and thus more accurate. In comparison, nearest neighbours takes big jumps from one pixel value to another.

3 Theory

3.1

R = $\begin{bmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{bmatrix}$

3.2

To transform a point we use the dot product between the rotation matrix and the point.

$$\begin{bmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

3.3

Through matrix multiplication $R(x,y)^T$, after translating $(x,y)^T$ to the origin, and afterwards translating the image back to the original position, as shown in the following formula:

$$R_\phi(x-t)+t$$

where x describes the $(x,y)^T$ coordinate, and t describes the origin vector.

3.4

lets say center is $(1,2)^T$

$$\begin{bmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{bmatrix} (\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \end{bmatrix}) + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\begin{bmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Rotating the center around itself results in the center staying at its place.

3 Matlab

The run time of bi-linear interpolation is approximately 0.049377 seconds, the run time of nearest neighbour interpolation is approximately 0.079630 seconds. From the images you can see that the nearest neighbour method produces a lower quality image, but it is nearly twice as fast. To verify the quality difference the distance is calculated between the pixel values of the original picture, and a picture that has been rotated 30 degrees and then rotated back 30 degrees and then cropped out of the black box that was there to preserve image information. The distance is calculated by subtracting the images matrices from each other and raising each number to the power of 2.
Squared error for nearest neighbour = 602.034823
Squared error for bilinear = 368.384824
The black box around the rotated image does influence the resulting squared errors a little bit because the image is not cropped out perfectly, but when comparing the two interpolation methods it does not matter because both methods get the same error margin from the black box, which cancels each other out.

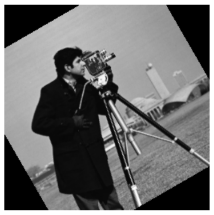
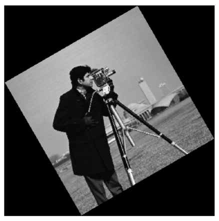
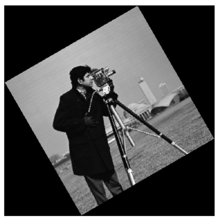


Figure 1: 30 Degree rotation while maintaining image size.



(a) Bi-linear interpolation



(b) Nearest neighbour interpolation

Figure 2: 30 Degree rotation while maintaining image information.

4 Theory

4.1

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

The dimensions of the matrices are not compatible and c and f have no effect on the outcome.

4.2

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

By adding a 1 as third dimension of the vector we can fix the compatibly of the 2 matrices and also incorporate c and f.

4.3

$$\begin{bmatrix} cos\phi & -sin\phi & c \\ sin\phi & cos\phi & f \end{bmatrix}$$

This matrix performs a rotation and then a translation. $(c, f)^T$ makes up the translation vector and ϕ is the rotation amount in radials.

4.4

To solve the equation we can any three points and the points which they map to. The easiest way to do this is to take the three points that map to the corners of the output image, in an output image with dimensions MxN these points have coordinates $(1, 1)^T$, $(M, 1)^T$ and $(1, N)^T$.

4.5

In order to solve the unknowns of a linear system we need at least as many equations as unknowns. In this case we have 6 unknowns so we need 6 equations, each point provides us with 2 equations, one for x and one for y, so we minimally need 3 points. Taking less points will result in some unknowns having infinitely many possible solutions. Taking more points will not add more information since those points would be able to be made from the already chosen points. Besides that taking a wrong fourth point could lead to a contradiction making the linear system unsolvable.

4 Matlab

To rotate an image 45 degrees using the affine matrix the matrix first has to be solved. This is done by finding 3 points which map to 3 output points in the resulting image. This is done by taking the corners of the output image and rotating them backwards using a rotation matrix.

Taking the points $(1,1)^T$, $(M,1)^T$ and $(1,N)^T$ and rotating them gives us:
x1 = 128.000000, y1 = -51.605122
x2 = 308.312229, y2 = 128.707107
x3 = -52.312229, y3 = 128.707107
M = 256.000000, N = 256.000000

With these parameters the function myAffine.m is called which rotates the image.

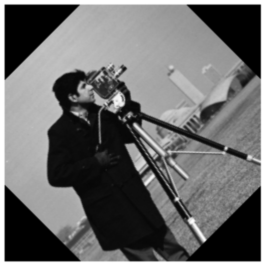


Figure 3: 45 Degree rotation using affine matrix.

5 Theory

5.1

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + m_{13} \\ m_{21}x + m_{22}y + m_{23} \\ m_{31}x + m_{32}y + m_{33} \end{bmatrix} = \begin{bmatrix} \lambda x' \\ \lambda y' \\ \lambda \end{bmatrix}$$

5.2

$m_{11}x + m_{12}y + m_{13} = \lambda x'$
$m_{21}x + m_{22}y + m_{23} = \lambda y'$
$m_{31}x + m_{32}y + m_{33} = \lambda$

Substitute λ into the other equations.

$m_{11}x + m_{12}y + m_{13} = m_{31}xx' + m_{32}yx' + m_{33}x'$
$m_{21}x + m_{22}y + m_{23} = m_{31}xy' + m_{32}yy' + m_{33}y'$

Giving us 2 equations for each point correspondence.

5.3

It needs at least 8 parameters.

5.4

You can divide the matrix M by m_{33} resulting in m_{33} always being 1 giving us 8 unknowns instead of 9. This can be done because multiplying M by a constant does not have an effect on the final answer because we're not interested in scalars.

5.5

To solve 8 unknowns we need a minimum of 8 equations. 4 points give us 8 equations. So we minimally need 4 points.
In 3D M is a 3x4 matrix giving us 12 unknowns, dividing everything by m_{34} gives us 11 unknowns, so we need a minimum of 6 points.

5.6

$$\begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}$$

5.7

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' & -x' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' & -y' \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

5.8

To avoid $\mathbf{m} = 0$ we say $||\mathbf{m}||$ must be 1.

5.9

Because we are projecting to 2D space instead of 3D.

5.10

In the formula $D\mathbf{m} = 0$ m is solved by taking $\mathbf{m} \in kernal(D)$. This is done by using the $null(D)$ function in matlab.

5 Matlab

projMatrix is nearly the same as the one above, the order is different but the result is the same.

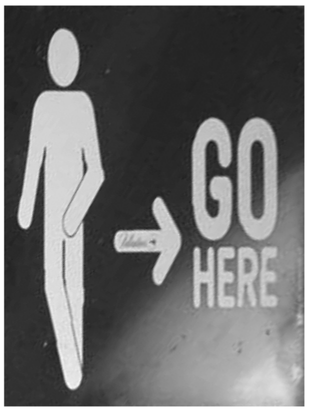


Figure 4: Object projected on a 400x300 pixel image.

7 Theory

7.1

First we get the matrix formula.

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{24} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix}$$

giving us the formulas

$m_{11}X + m_{12}Y + m_{13}Z + m_{14} = \lambda x$
$m_{21}X + m_{22}Y + m_{23}Z + m_{24} = \lambda y$
$m_{31}X + m_{32}Y + m_{33}Z + m_{34} = \lambda$

substituting λ into the other equations gives us

$m_{11}X + m_{12}Y + m_{13}Z + m_{14} = m_{31}Xx + m_{32}Yx + m_{33}Zx + m_{34}x$
$m_{21}X + m_{22}Y + m_{23}Z + m_{24} = m_{31}Xy + m_{32}Yy + m_{33}Zy + m_{34}y$

putting these formulas into a matrix multiplication

$$\begin{bmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -x \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -xZ & -x \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{24} \end{bmatrix}$$

Left side being a part of A for each point correspondence. Giving us the whole matrix A with n = amount of point correspondences.

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{bmatrix}$$

7.2

Adding more correspondences will increase the amount of rows but not columns so the matrix multiplication still works. Adding more correspondences will also increase the accuracy of the estimation of the m vector.

7.3

No there will still not be an exact solution, but there will be a more accurate estimation.

7.4

We want to minimize $||A\mathbf{x}||$ while keeping $||\mathbf{x}|| = 1$ to prevent $\mathbf{x}$ being all 0s. To do this we take the SVD of A giving us $||UDV^T\mathbf{x}||$. U has no effect on the length so we can leave it out giving us $||DV^T\mathbf{x}||$. V^T also has no effect on the length so we can change the restriction to $||V^T\mathbf{x}|| = 1$. Then we say $V^T\mathbf{x} = y$ giving us $||Dy||$. D is the matrix containing all eigenvalues ordered from big to small on the diagonal, so to minimize $||Dy||$ while keeping $||y|| = 1$ we choose $y = (0, 0, ..., 0, 1)^T$ to get only that smallest eigenvalue. Going back we have $y = V^T\mathbf{x}$ we want $\mathbf{x}$ so $\mathbf{x} = Vy$. Filling in y gives us $\mathbf{x} = V(0, 0, ..., 0, 1)^T$ and the solution to that gives us the last column of V.

8 Matlab

A minimum of 18 calibration points are needed to perform these projections.

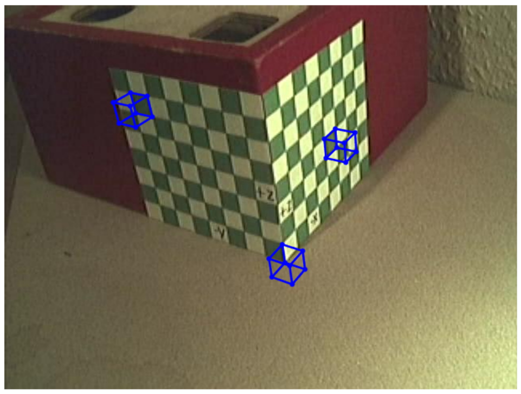


Figure 5: Recreation of Figure 9 in the homework assignment

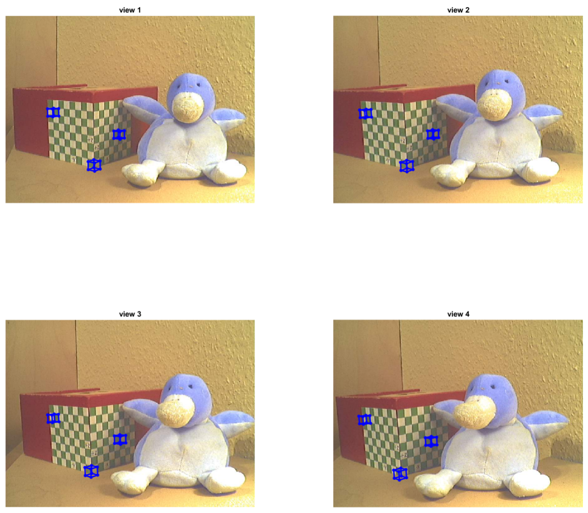


Figure 6: Recreation of the cubes in Figure 9 of the homework assignment from different viewing angles