Gideon Mooijen, 10686290
Lucas van Berkel, 10747958

2 Interpolation

Interpolation is creation of new data-points, based on estimations. Assume a picture is being enlarged. All pixels have a certain colour, and the new position is calculated through *the transformation algorithm*. This will result in an incomplete picture: the result will have more pixels than the input. To 'fill in the blanks', nearest-neighbour interpolation can be used. The empty spaces will be replaced by the nearest neighbour. The floor function rounds down a number, so the transformation algorithm can be applied. So: new positions that could not be calculated (x must be an integer), are now possible to be calculated, by replacing it for the lowest known data-point. If we want to fit a line $f_1(x) = ax + b$, we need to solve the following equations.

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

$$b = (1+k)F(k) - kF(k+1) \tag{2}$$

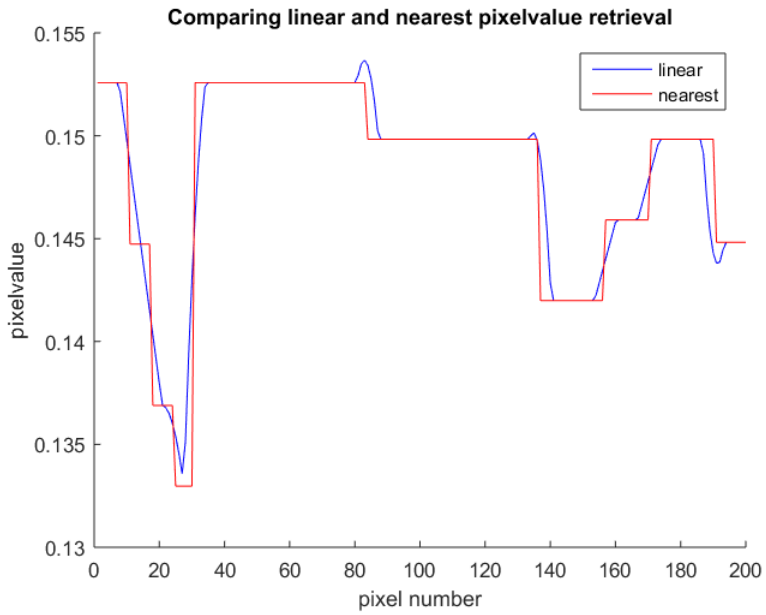

Figure 1: Comparing linear and nearest pixelValue retrieval

3 Rotation

The formula for rotating counterclockwise around the origin in 2D is given by:

$$\begin{bmatrix} cos\phi & sin\phi \\ -sin\phi & cos\phi \end{bmatrix} \tag{3}$$

This can be simulated by rotating around the z-axis in 3D. Then, the equation is:

$$\begin{bmatrix} cos\phi & sin\phi & 0 \\ -sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The 2x2-matrix can be used to rotate a $(x, y)^T$ to a point $(x', y')^T$ around the origin. The 3x3-matrix can be used when an extra value is added. $(x, y)^T$ will be rewritten as $(x, y, 1)^T$ and results in $(x', y', 1)^T$. If the last value is disregarded, the outcome is the same. When we want to rotate about an arbitrary point c, we need to perform a translation as well. In that case, we need to obtain the coordinates of the point c, perform that translation first, then rotate, and then reverse the translation. You can find the corresponding matrix below:

$$\begin{bmatrix} cos\phi & sin\phi & -a \\ -sin\phi & cos\phi & -b \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Note: a and b are the translation variables and dependant on point c. This is again, a counterclockwise rotation. With c itself happens nothing, because it is translated to the exact origin when the first translation is executed, so the rotation has no effect. Afterwards, it is 'put back' to its original location.

Figure 2: Before, during and after rotations

4 Affine Transformations

For affine transformations, we need obtain homogenous coordinates. To solve the two formulas as a matrix equation, we need to introduce a new coordinate. We rewrite $(x, y)^T$ as $(x, y, 1)^T$. Since we have 6 parameters, we need 3 pairs of points. The introduction of the third coordinates allows solving this system.

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{6}$$

If we replace a, b, d and e from our matrix in (3), we obtain:

$$\begin{bmatrix} cos(\phi) & sin(\phi) & c \\ -sin(\phi) & cos(\phi) & f \end{bmatrix} \tag{7}$$

This is a transformation matrix that contain both a rotation that is dependant on ϕ, and a translation dependant on c (x-direction) and f (y-direction). The best points to be considered are the corners of the frame, because they can easily be identified. We need exactly three points. With less, the system can not be solved (less equations than variables). With more, our system would be overdetermined. Then we can still find a solution with the least-squares-method, but it is not desirable.

Figure 3: Rotating 45 degrees using affine transformation

5 Re-projecting Images

Written in terms of m_{ij}:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \tag{8}$$

Seemingly, there are 9 unknowns in this matrix, but in fact, there are 8. This is due to the fact that to transform a quadriliteral to another quadriliteral in 2D, we need 4 corners, so 8 datapoints, as each point has two values (x, y). So: the unknowns can be written as the vector:

$$\begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \\ 1 \end{bmatrix} \tag{9}$$

We can solve this through the following equation:

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -x_1v_1 & -x_1 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -y_1u_1 & -y_1v_1 & -y_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -x_2v_2 & -x_2 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -y_2u_2 & -y_2v_2 & -y_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -x_3v_3 & -x_3 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -y_3u_3 & -y_3v_3 & -y_3 \\ u_4 & v_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -x_4v_4 & -x_4 \\ 0 & 0 & 0 & u_4 & v_4 & 1 & -y_4u_4 & -y_4v_4 & -y_4 \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{10}$$

If we use $\|\mathbf{m}\| = 1$, we can solve the system exactly. This corresponds to asking $\mathbf{m}$ to be in the nullspace of the matrix, which is 1-dimensional. The projection transformation is sufficiently determined by fewer than 9 parameters, because it is still dependant on only the rotation and translation.

Figure 4: Projecting flyer

7 Estimating a Camera's Projection Matrix

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 & -x_2 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 & -y_2 \\ . & . & . & . & & . & . & . & . & . & . & . \\ X_n & Y_n & Y_n & 1 & 0 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{bmatrix} \tag{11}$$

If you have many more correspondences it can still represent the same equations, only if they fit the projection. If you have many more correspondences and they do not fit the projection, you can use the 'least-squares-method' to obtain the outcome that best fits the correspondences. In that case, there will not be an exact solution for $A\mathbf{m} = 0$. Note: we are looking for the non-trivial solution where $\mathbf{x} \neq 0$. The last column of V in the SVD of A corresponds to the null space of A (or: kernel of A). This last column contains one value, and because they appear on V in descending order, $y = (0, 0, ..., 1)^T$ returns the lowest value. This results in the best approximation of the null-space of A (or: kernel of A) and thus, is the best minimalization possible.

8 Projecting Cubes into a Picture (Pseudo 3D)

Following is the recreation of the given image using our code. We need at least three calibration points per image.

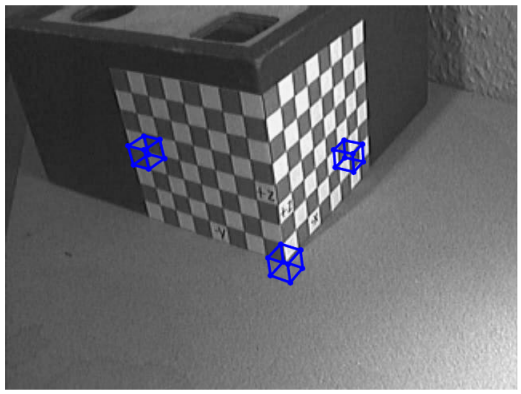

Figure 5: Part A

The following figure is the outcome of part B. The small errors you might observe are most likely the outcome of the ginputfunction (not succesfully clicking in the desired location).

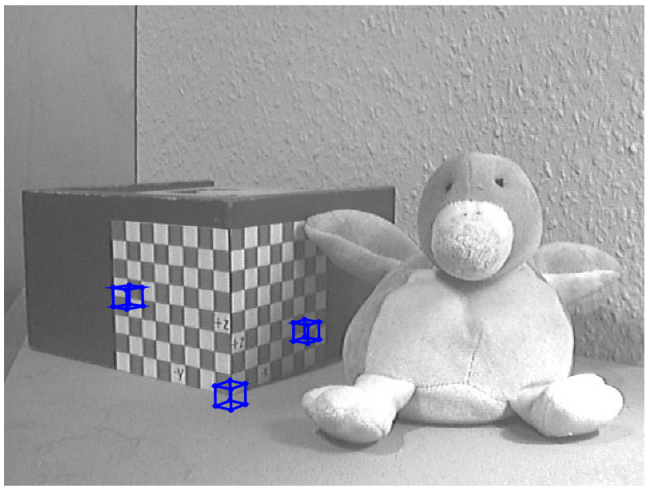

Figure 6: Part B

9 How to run

There is a different .m script for each part of the exercise. These should be executed autonomously to test the functionality of our code. It is pretty straightforward, as each part can simply be opened and run to check the outcome.