Lab 1 - theory

Beeldverwerken
April 18, 2018

Patrick Spaans (11268476)
Bram Otten (10992456)

How to run MATLAB code

We've made a live script, `Spaans_Otten_Lab1.mlx`. That calls a bunch of functions, which are in the accompanying `.m` files. If it doesn't work, there's a `Spaans_Otten_Lab1_backup.m` and a PDF.

2.1

To get the outcome of $F(x)$ for the nearest neighbour of some non-integer, we need to get the F of the integer nearest to that non-integer: $F(\lfloor x + .5 \rfloor)$

2.2

To fit $f_1(x) = ax + b$ with known integers k and $k+1$ and known outcomes $F(k)$ and $F(k+1)$, we can use the difference between those in- and outputs.

$$F(k) = ak + b$$
$$F(k+1) = a(k+1) + b$$

2.3

The equations of 2.2 in matrix form and their solution:

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

2.4

We know $F(k,l)$, $F(k,l+1)$, $F(k+1,l)$, $F(k+1,l+1)$ and want an $F(x,y)$ inbetween those points. To interpolate from the four surrounding points with known outcomes, we first interpolate for just the two points in both single dimensions, filling in something known for the other dimension. We then interpolate between two already 'uni-dimensionally-interpolated' points for that final answer:

$$\begin{aligned}
f(x,l) &= (-F(k,l) + F(k+1,l))x + (k+1)F(k,l) - kF(k+1,l) \\
&= (1-(x-k))\,F(k,l) + (x-k)\,F(k+1,l) \\
&= (1-\alpha)\,F(k,l) + \alpha\,F(k+1,l) && \text{(with } \alpha = (x-k)\text{)} \\
f(k,y) &= (-F(k,l) + F(k,l+1))y + (k,l+1)F(l) - kF(k,l+1) \\
&= (1-(y-l))\,F(k,l) + (y-l)\,F(k,l+1) \\
&= (1-\beta)\,F(k,l) + \beta\,F(k,l+1) && \text{(with } \beta = (y-l)\text{)} \\
f(x,y) &= (1-\beta)\,f(x,l) + \beta\,f(x,l+1) && \text{inserting } f(x,l) \text{ etc.} \\
&= (1-\beta)\,((1-\alpha)\,F(k,l) + \alpha\,F(k+1,l)) + \\
&\quad \beta\,((1-\alpha)\,F(k,l+1) + \alpha\,F(k+1,l+1))
\end{aligned}$$

3.1

The following matrix **R** gives a counter-clockwise rotation of ϕ radions about the origin:

$$\mathbf{R} = \begin{pmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{pmatrix}$$

3.2

We can use this matrix **R** to figure out where $\begin{pmatrix} x \\ y \end{pmatrix}$ in ends up after a ϕ rotation around the origin:

$$\begin{pmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

3.3

To rotate about an arbitrary point **c**, we first translate to make **c** the origin temporarily, then rotate, and then we do the inverse of the translation to **c** to return to the original position.

$$\begin{pmatrix} 1 & 0 & -c_1 \\ 0 & 1 & -c_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & c_1 \\ 0 & 1 & c_2 \\ 0 & 0 & 1 \end{pmatrix}$$

3.4

c is the center of the rotation, which the rotation matrix considers $(x, y) = (0, 0)$ and thus:

$$\mathbf{Rc} = \begin{pmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \mathbf{c}$$

The center of a rotation is therefore at the same location before and after a rotation.

4.1

$$x' = ax + by + c \text{ and } y' = dx + ey + f$$

In these equations, x' is translated by c, while y' is translated by f. This right now impossible to represent in the matrix equation, since there is no $\mathbf{R}$ such that $\mathbf{R}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$. c and f can't be added as a singular values to x and y.

4.2

We need a trick, represent the $\mathbf{x}$ vector in one more dimension, to deal with the constants c and f.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + c \\ dx + ey + f \end{pmatrix}$$

4.3

Replacing a, b, d, e with the elements of a 2D rotation matrix, the matrix multiplication shown above becomes a rigid body motion: a rotation followed by a translation.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & c \\ \sin\phi & \cos\phi & f \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

4.4

We can figure out the three required point-pairs of the in- and output image because we know the origin and probably something else like the size of the image.

So if we know the height and width of the output image, we can figure out three output coordinates (0, 0), (0, width), (height, 0).

4.5

A property of an affine transformation is that grid lines remain parallel: the initially orthonogal x and y axes (and therefore the edges of the parallelogram) remain orthogonal.

This means that once we have the coordinates of three corners of the parallelogram, we don't need the coordinate of the last corner, since it can be inferred. (It's in the span of the basis we've already defined.) So more than 3 coordinates isn't necessary.

With less than 3 corner coordinates, we can't infer the coordinates of the other corners, which means that we don't have enough information to solve all the parameters.

5.1

The specified function can be written in terms of m_{ij} by replacing a with m_{11}, b with m_{12}, ..., m with m_{33}:

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

5.2

This works out to the following equations for each point correspondence i:

$$\begin{aligned} m_{11}x_i + m_{12}y_i + m_{13} &= \lambda_i x_i' \\ m_{21}x_i + m_{22}y_i + m_{23} &= \lambda_i y_i' \\ m_{31}x_i + m_{32}y_i + m_{33} &= \lambda_i \end{aligned}$$

fill in for just two per correspondence:

$$\begin{aligned} m_{11}x_i + m_{12}y_i + m_{13} &= (m_{31}x_i + m_{32}y_i + m_{33})\, x_i' \\ m_{21}x_i + m_{22}y_i + m_{23} &= (m_{31}x_i + m_{32}y_i + m_{33})\, y_i' \end{aligned}$$

5.3

For transforming any arbitrary quadrilateral into any other arbitrary quadrilateral, we need 8 parameters. We know the quadrilateral's four corner angles must add up to 360 degrees, but not much else; we can't infer the fourth corner from the other three like we can for a parallelogram. So we need the coordinates of all of those corners, or a transformation matrix containing as much information.

5.4

The parameter m_{33} isn't really an unknown parameter, since it's a constant that doesn't change the direction of the vectors, only the placement of the points. It has an influence on λ only (it acts as a constant that's always added to it). The whole vector is later divided by that λ, so it doesn't really matter what its value is. By setting it to 1 it can normalize the matrix without impacting the rotation, and it will no longer be an unknown.

5.5

With 8 unknowns, only 4 correspondence points are needed, since 4 correspondence points can be used to create 8 equations, which can solve all the unknowns.

5.6

Collecting all nine unknowns in a vector, we get:

$$\mathbf{m} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{21} & m_{22} & m_{23} & m_{31} & m_{32} & m_{33} \end{pmatrix}^T$$

5.7

With X_i corresponding to x_i in the original image and Y_i corresponding to y_i, the correspondence equations from before (again, with X for x' and Y for y' as the points to project to) can be rewritten into:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 & X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 & Y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 & X_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 & -y_2Y_2 & Y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3X_3 & X_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3Y_3 & -y_3Y_3 & Y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 & X_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4Y_4 & -y_4Y_4 & Y_4 \end{pmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \\ m_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

5.8

In order to avoid the trivial solution for $\mathbf{Am} = \mathbf{0}$, where $\mathbf{m}$ is just a zero vector, we have to add a constraint to a where it has to be the norm of 1. By removing m_{33} as an unknown and giving it the value of 1 (as in question 5.4), we can allow a norm of 1 while also preventing the trivial solution.

5.9

While linear transformations in 3D require nine parameters, this projective transformation requires less. We may drop a few handy properties like parallel lines remaing parallel when projecting, but we're still working in 2D and straight lines remain straight. There's no third dimension 'hidden' in images that show perspective, and there being nine unknowns here and in 3D doesn't necessarily mean there's much of a relation. The constraints we have to deal with in projective transformations in 2D are just different from those for linear transformations in 3D.

5.10

A null space function is very useful for this problem. $\mathbf{m}$ should be `null(A)` in MATLAB. The returned basis for the null space is orthonormal, and definitely not trivial.

7.1

For each correspondence points, the following equations can be obtained:

$$\begin{aligned}\lambda x_i &= m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}\\ \lambda y_i &= m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}\\ \lambda &= m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}\end{aligned}$$

By eliminating λ, we get:

$$\begin{aligned}m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14} - m_{31}X_i x_i - m_{32}Y_i x_i - m_{33}Z_i x_i - m_{34}x_i = 0\\ m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24} - m_{31}X_i y_i - m_{32}Y_i y_i - m_{33}Z_i y_i - m_{34}y_i = 0\end{aligned}$$

The unknowns are all 12 values of $\mathbf{m}$. With 6 known points, we get 12 rules which can be split up to form $\mathbf{Am} = \mathbf{0}$, where $\mathrm{m} = \{m_{m11}, m_{m12}, \ldots m_{33}, m_{34}\}'$. $\mathbf{A}$ consists of a row for each rule, where the row multiplied by the $\mathbf{m}$ vector returns 0 like in the original rule.

$$A = \begin{pmatrix}
X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -X_1x_1 & -Y_1x_1 & -Z_1x_1 & -x_1\\
0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -X_1y_1 & -Y_1y_1 & -Z_1y_1 & -y_1\\
X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -X_2x_2 & -Y_2x_2 & -Z_2x_2 & -x_2\\
0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -X_2y_2 & -Y_2y_2 & -Z_2y_2 & -y_2\\
X_3 & Y_3 & Z_3 & 1 & 0 & 0 & 0 & 0 & -X_3x_3 & -Y_3x_3 & -Z_3x_3 & -x_3\\
0 & 0 & 0 & 0 & X_3 & Y_3 & Z_3 & 1 & -X_3y_3 & -Y_3y_3 & -Z_3y_3 & -y_3\\
X_4 & Y_4 & Z_4 & 1 & 0 & 0 & 0 & 0 & -X_4x_4 & -Y_4x_4 & -Z_4x_4 & -x_4\\
0 & 0 & 0 & 0 & X_4 & Y_4 & Z_4 & 1 & -X_4y_4 & -Y_4y_4 & -Z_4y_4 & -y_4\\
X_5 & Y_5 & Z_5 & 1 & 0 & 0 & 0 & 0 & -X_5x_5 & -Y_5x_5 & -Z_5x_5 & -x_5\\
0 & 0 & 0 & 0 & X_5 & Y_5 & Z_5 & 1 & -X_5y_5 & -Y_5y_5 & -Z_5y_5 & -y_5\\
X_6 & Y_6 & Z_6 & 1 & 0 & 0 & 0 & 0 & -X_6x_6 & -Y_6x_6 & -Z_6x_6 & -x_6\\
0 & 0 & 0 & 0 & X_6 & Y_6 & Z_6 & 1 & -X_6y_6 & -Y_6y_6 & -Z_6y_6 & -y_6
\end{pmatrix}$$

7.2

More correspondence points can just be added to the matrix as more rows, but could result in there not being a null space of $\mathbf{A}$, since points are likely slightly off and thus inconsistent. (In order to still get a value for $\mathbf{m}$ to be used in representing the resulting equations, the null space value of $\mathbf{A}$ will have to be approximated by trying to minimize $\mathbf{Am}$.)

7.3

$\mathbf{Am} = \mathbf{0}$ will most likely not be exactly correct with any non-trivial **m**, as correspondences are in turn likely to be in conflict with each other. The best solution will be to minimize **Am** (with, still, a non-trivial **m**).

7.4

(In this question **m** is **x**, and most not `\mathbf{}`'ed letters are still vectors or matrices.)

The procedure works as follows:

The goal is to minimize the norm of Ax, without trivial solutions. This can be done by adding a constraint that says that the norm of x must be 1.

A can be written as its SVD decomposition, which is UDV^T. Since U is orthogonal, it doesn't influence the norm andcan be removed. V (and thus V^T) is also orthogonal, so it is allowed to add V^T to the constraint. This gives us:

"Minimize the norm of $DV^T x$, with as constraint that the norm of $V^T x$ must be 1."

By adding a rule $y = V^T x$, the problem can be simplified to:

"Minimize the norm of Dy, with as constraint that the norm of y must be 1."

Since D is a diagonal matrix, to get the smallest norm of Dy, we take only the smallest value of D through y. Since the values of diagonal matrices are listed in descending order, this means that we only want the last value of D. To get this value, we want y to be equal to $[0, 0, ...0, 1]^T$.

The last thing to be done is getting the value of x (which contains the solutions to the unknown). Since $y = V^T x$, $x = Vy$, which means that $x = V(0, 0, ..., 0, 1)^T$. Since the only value that isn't 0 is the last value, the only relevant values of V are those that are multiplied by that value, which are all the values in the last column of V. This means that the best possible solution is the last column of V multiplied by 1, which is just the last column of V.