Lecture 6 - Alignment

Gidon Rosalki

2025-11-23

1 Alignment

We have been discussing finding the transformation between two images, i.e. the difference in translation, rotation, zoom. There are also affine transformations, and homography transformations. Our assumption is that it is a static scene, with no 3D effects like motion parallax. This is good for video stabilisation, denoising, video mosaicing, and so on.

We discussed the different types of transformations at length in last week's tutorial. In short, we had translation, scaling / zoom, rotation, affine, and projective / homography. Most affine transformations cannot be generated by changing the camera parameters.

1.1 Rolling shutter

In a film camera, an entire frame is recorded at the same time. This was similarly true with the first digital cameras (used CCD). In newer digital cameras, like all mobile phones, instead of CCD, they use CMOS, where each line is recorded at a different time. In a 24fps video, there may be 1000 lines in the camera sensor, so it records 24,000 lines per second for a $1K \times 1K$ image. There's a very good video by Smarter Every Day explaining rolling shutter if it is unclear. Overall, closer objects to the image move relatively faster, resulting in a greater slant.

To reduce the effect of rolling shutter, one may use a very short exposure time, since then there will be less change to the image between the top line, and the bottom line. However, in this course, we are going to pretend that there is no rolling shutter.

1.2 Computing global translation

1.2.1 Point correspondences

Assume that we can find two corresponding point pairs, between two images. We can then compute the translation between these two points, and thus find a way to match them together. The problem with this is that there may be many wrong correspondences, or even *no* correspondences. Consider an image with many repeating sections, like an image of a circle. There are no points, and many repeating sections.

1.2.2 Direct methods

We are assuming constant brightness, and no rolling shutter.

Given images I_1, I_2, we can find teh translation (u, v), that will minimise the SSD (Sum of Squared Differences):

$$E(u,v) = \sum_x \sum_y \left(I_1(x,y) - I_2(x+u, y+v)\right)^2$$

the implementation is to use the average **per pixel** error, only over an **area of overlap** (meaning, not measuring the difference over where the images do not overlap). However, we also must divide by the number of pixels in the area of overlap, so that the score does not change unduly by decreasing the area of overlap.

Since

$$(a-b)^2 = a^2 - 2ab + b^2$$

we can write

$$E(u,v) = \sum_x \sum_y I_1^2 - 2\sum_x \sum_y I_1(x,y) \cdot I_2(x+u, y+v) + \sum_x \sum_y I_2^2$$

Since the first, and last terms are both almost constant, minimising the SSD is simply maximising the cross correlation

$$CC(u,v) = \sum_x \sum_y I_1(x,y) \cdot I_2(x+u, y+v)$$

However, consider comparing between an image, and a lighter version of it. Given this method, there will be a higher correlation between it and a lightened version of itself image, than between a duplicate, since multiplying by a higher value pixel results in a higher value. To avoid this, we create NCC, which is invariant to global addition, and multiplication of intensity $I_2 = a \cdot I_1 + b$.

$$NCC\left(u,v\right) = \frac{\sum\left(I_1\left(x,y\right) - \hat{I}_1\right) \cdot \left(I_2\left(x+u, y+c\right) - \hat{I}_2\right)}{\sqrt{\sum\left(I_1\left(x,y\right) - \hat{I}_1\right)^2}\sqrt{\sum\left(I_2\left(x,y\right) - \hat{I}_2\right)^2}}$$

Where in short, we subtract the average grey level, and divide by the variance.

There is also multiresolution search, where we make use of pyramids. We start with a very small version of both images, and find the highest matching area. We can then go down a level to a higher resolution version, and search within the same area, and repeat this until we reach the full size images. This way, we do not have so many computations of comparisons, since we only search a small window every time.

Normalised Cross Correlation is an excellent method to find objects in pictures, and to track objects in video. Multiresolution search (pyramids) is used in object search, and is not needed when tracking from one frame to another.

There are some limitations of correlation search. There is only discrete accuracy, checking every possible translation in integer pixel values, there is no sub pixel accuracy. Furthermore, the Complexity increases exponentially with numbers of parameters: A translation (u,v) has complexity of n^2, rotation (u,v,α) of n^3, zoom (u,v,α,s) has n^4, and affine is n^6.

1.2.3 Continuous approximation

This is the Lucas-Kanade (LK) approximation. A local Taylor approximation in 1D is given as

$$f\left(x+u\right) \approx f\left(x\right) + f'\left(x\right) \cdot u + \ldots$$

This may be expanded into 2D for images:

$$f\left(x+u, y+v\right) \approx f\left(x,y\right) + \frac{\partial f}{\partial x} \cdot u + \frac{\partial f}{\partial y} \cdot v + \ldots$$

1.2.4 Alignment by error minimisation

When I_2 is shifted relative to I_1, we want to find the translation (u,v) by minimising SSD:

$$E\left(u,v\right) = \sum_x \sum_y \left(I_1\left(x,y\right) - I_2\left(x+u, y+v\right)\right)^2$$

To simplifu, we may look at a single pixel, and use the Taylor approximation

$$\begin{aligned} E\left(u,v\right) &= \left(I_1\left(x,y\right) - I_2\left(x+u, y+v\right)\right)^2 \\ &\approx \left(I_2\left(x,y\right) + \frac{\partial I_2}{\partial x} \cdot x + \frac{\partial I_2}{\partial y} \cdot v - I_1\left(x,y\right)\right) \\ &= \left(I_x \cdot y + I_y \cdot v + I_t\right)^2 \end{aligned}$$

Where

$$I_x = \frac{\partial I_2}{\partial x} \tag{1}$$

$$I_y = \frac{\partial I_2}{\partial y} \tag{2}$$

$$I_t = I_2 - I_1 \tag{3}$$

So, writing it in the simple form, we get the error function

$$E\left(u,v\right) = \left(I_x \cdot u + I_y \cdot v + I_t\right)^2$$

Where I_x is the x derivative of I_2, I_y is the y derivative of I_2, and I_t is the image difference $I_2 - I_1$. We are looking to find (u,v) which minimise the error function. The same (u,v) will approximately minimise the Taylor approximation, but it is only accurate for very small values of (u,v), on the order of a single pixel, since it is only the first order Taylor approximation. For larger values of (u,v), we have the iterative approach.

1.2.5 Iterative approach

We compute the image derviatives I_x, I_y, and set u, v both to 0. Compute once

$$A = \begin{bmatrix} \sum I_x \cdot I_x & \sum I_x \cdot I_y \\ \sum I_y \cdot I_x & \sum I_y \cdot I_y \end{bmatrix}$$

We iterate until convergence ($I_t \approx 0$), by

1. Compute $b = \begin{bmatrix} \sum I_x \cdot I_t \\ \sum I_y \cdot I_t \end{bmatrix}$
2. Solve the equations to compute the residual motion $A \cdot \begin{bmatrix} du \\ dv \end{bmatrix} = -b$
3. Update motion u, v with residual motion $u+ = du,\ v+ = dv$
4. Warp I_2 towards I_1 with total motion (u, v)

The power of iterations allows us to compute the derivatives only once on I_1. There are two stages in each iteration, motion estimation (solving equations), and warping I_2 (usually backwards). This works even with poor motion estimation, as long as it reduces the overall error. Warping of one image towards the other is done from the original image using total motion, and not from previous image using residual motion. Repetitive warping blurs.

1.2.6 Multiresolution

Lucas-Kanade assumes that corresponding pixels in the two images have same derivative. It works OK even if derivatives are similar. However, this fails for very large motions. So, reducing the resolution blurs the images, creating similar derivatives, even for larger motions.

If we compare feature points to LK, we find that in both cases we go over the image to compute partial derivatives. As a result, there is a very similar complexity. When uniquely identifiable features points can be found, then they are better, as they can be used to compute homographies. However, in blurry images, feature points may be difficult to find, and so LK may be preferable. Overall, LK is more accurate in translation.

1.3 Optical flow

Optical flow is when there is an individual motion for each pixel (like when there is parallax, or independently moving objects).

We have some key assumptions (shocking nobody):

- Colour consistency, there is no change in colour (this is changed to brightness in greyscale images)
- Small motion: The points do not move very far

Examining small windows around a pixel may not provide accurate motion. Consider a line that passes beyond the window. We would be unable to differentiate between the line translating to the left, and translating up and to the left. This is called the **aperture problem**.

1.3.1 Correlation based optical flow

For each small region in one image, we search for the best correlation in the second image. This assumes that we do not move too much, and thus do not need to search the entire image. If there is a large region, then there is a higher probability of recognising accurate motion, but there will also be poor localisation (poor knowledge of what moved). A smaller region results in better localisation, but poorer knowledge of motion. We use pyramids to reduce the search area.

1.3.2 Correlation based pyramids for optical flow

To perform this we create two Gaussian pyramids from the two input images, compute the optical flow using 5×5 regions on the smallest pyramid level, smooth the optical flow and use it as an initial guess for higher resolutions, and continue to the next level, searching close to the guess from the higher resolution.

1.3.3 Gradient based optical flow

Here, we compute (u, v) using Lucas-Kanade between two corresponding regions.

- Large region: Accurate motion. Poor localization.
- Small region: Good localization. Poor motion.

We use pyramids to reduce search area.

1.3.4 Pyramids and iterative refinement

- Create two Gaussian pyramids from the two input images
- Iterative Lukas-Kanade on smallest images
    1. Estimate velocity at each pixel by solving Lucas- Kanade equations in its neighborhood
    2. Warp I2 towards I1 using the estimated flow field
    3. Repeat until convergence
- Continue to next pyramid level.