TP: Lucas Kanade Tracker

edmond.boyer@imag.fr, elise.arnaud@imag.fr

The objective of this practical work is to implement the Lucas Kanade method that enables the tracking of an object in an image sequence. This algorithm has been proposed in the 80's. It is still very used in image analysis and computer vision.

Definition

In the following, we will denote :

- i et j the position in line and column of a pixel in an image,
- $T(i, j)$ the image template of the object to track. The template is of size $n \times m$,
- $I^t(i, j)$ the image at instant t of the image sequence, of size $N \times M$,
- $I^t_o(i, j)$ the image extracted from $I^t(i, j)$ of size $n \times m$ and showing the tracked object.

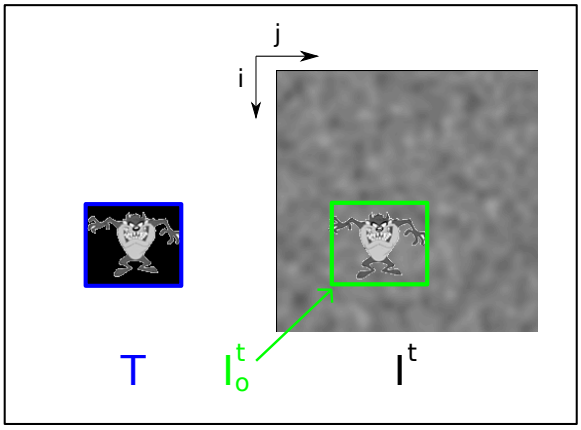

figure 1 - definitions

Principle

The Lucas Kanade algorithm is looking for the parameters Δi et Δj that minimise the following non linear expression:

$$\sum_{i,j} \big(T(i,j) - I_o^t(i+\Delta i, j+\Delta j)\big)^2$$

Under the assumption that the displacement is small (i.e. Δi et Δj below 4 pixels), this expression can be linearized to the first order, and become:

$$\sum_{i,j} \Big(T(i,j) - I_o^t(i,j) - \frac{\partial I_o^t(i,j)}{\partial i}\Delta i - \frac{\partial I_o^t(i,j)}{\partial j}\Delta j\Big)^2$$

This equation can be solved using a classical mean squared method (after few manipulations):

$$\begin{bmatrix} \Delta i \\ \Delta j \end{bmatrix} = \begin{bmatrix} \sum_{i,j} \left(\frac{\partial I_o^t(i,j)}{\partial i}\right)^2 & \sum_{i,j} \frac{\partial I_o^t(i,j)}{\partial i}\frac{\partial I_o^t(i,j)}{\partial j} \\ \sum_{i,j} \frac{\partial I_o^t(i,j)}{\partial i}\frac{\partial I_o^t(i,j)}{\partial j} & \sum_{i,j} \left(\frac{\partial I_o^t(i,j)}{\partial j}\right)^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i,j} \frac{\partial I_o^t(i,j)}{\partial i}(T(i,j) - I_o^t(i,j)) \\ \sum_{i,j} \frac{\partial I_o^t(i,j)}{\partial j}(T(i,j) - I_o^t(i,j)) \end{bmatrix} \tag{1}$$

The linearization step implies this equation to be solved iteratively. In practice, the image $T(i,j)$ has to be successively interpolated using the current displacement estimation Δ_i and Δ_j, and the equation has to be solved again until the estimated displacement is below a given threshold.

exercise 1

- Implement a function to extract a small image of size $n \times m$ from the image $I^t(i,j)$, and from pixel (i,j) (i.e. from i to $i+n-1$ and from j to $j+m-1$). This small image will be the image $I_o^t(i,j)$.
- Implement a function to estimate the gradient along i et j of an image.
- Implement a function that calculates the error between two images (i.e. $T(i,j) - I_o^t(i,j)$).
- Implement a function to calculate the inverse of a 2×2 matrix.

exercise 2

Implement a function to interpolate an image. This function takes as input parameters: an image $I(i,j)$, the translation along i denoted by Δ_i, the translation along j denoted by Δ_j.

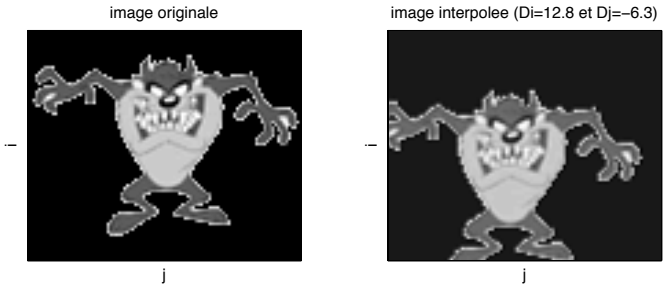

figure 2 - interpolation of an image by the translation Δ_i and Δ_j

As output, we should obtain the interpolated image $I_w(i,j)$. the value of a new pixel is a sum of the intensities of its four neighbours, weighted by the distance to the neighbours' centers.

The interpolated image is initialized with zeros. Let Δ^e be the integer part and Δ^f the decimal part of the displacement (example: $\Delta = 5.3$ then $\Delta^e = 5$ and $\Delta^f = 0.3$). In case of a negative displacement, the negative sign will be kept only on the integer part Δ^e (example: $\Delta = -1.7$ then $\Delta^e = -1$ and $\Delta^f = 0.7$). For each pixel (i,j) of the original image I, update image I_w in the following manner:

$$\begin{aligned}
I_w(i+\Delta_i^e, j+\Delta_j^e) &+= (1-\Delta_i^f)(1-\Delta_j^f)I(i,j) \\
I_w(i+\Delta_i^e+1, j+\Delta_j^e) &+= \Delta_i^f(1-\Delta_j^f)I(i,j) \\
I_w(i+\Delta_i^e, j+\Delta_j^e+1) &+= (1-\Delta_i^f)\Delta_j^f I(i,j) \\
I_w(i+\Delta_i^e+1, j+\Delta_j^e+1) &+= \Delta_i^f\Delta_j^f I(i,j)
\end{aligned}$$

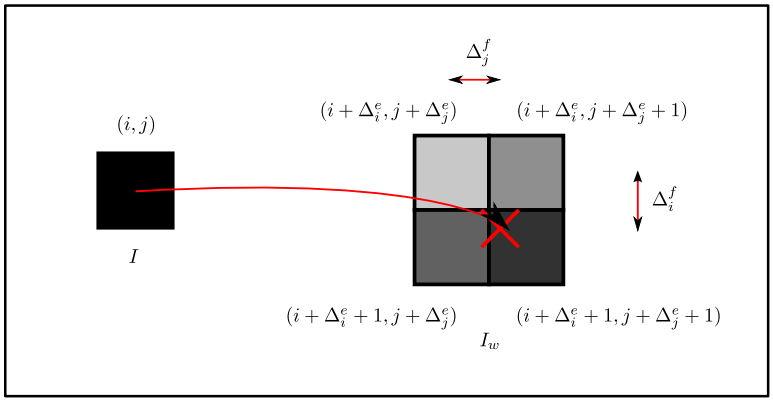

figure 3 - bilinear interpolation, the pixel value of image I is distributed on pixels of image I_w depending on the translations along i and j.

exercice 3

- Load the first image `taz001.pgm` of the image sequence data `synthetic` (image $I^1(i,j)$).
- Crop a small image of size 84×100 pixels at position $(i,j) = (50, 130)$ (image $I_o^1(i,j)$).
- Calculate the gradient along i and j of this image $I_o^1(i,j)$.
- Load image `taz.pgm` (image $T(i,j)$) that is your template.
- Run several iterations (300 maximum) of the following steps:

  Initialization: $\Delta_i = 0$, $\Delta_j = 0$, $T_w(i,j) = T(i,j)$

  1. Calculate the error between image $T_w(i,j)$ and image $I_o^1(i,j)$.
  2. Apply equation (1) to estimate the displacements along i (denoted δ_i) and along j (denoted δ_j).
  3. Update the displacements $\Delta_i = \Delta_i - \delta_i$ and $\Delta_j = \Delta_j - \delta_j$.
  4. Interpolate the image $T(i,j)$ with the displacements Δ_i and Δ_j and save it as image $T_w(i,j)$.

Advices:

- Check that the interpolated image $T_w(i,j)$ is moving in the right direction with respect to your estimation of Δ_i and Δ_j, and your image $I_o^1(i,j)$!
- You can reduce the number of iterations using a stopping criterion like "do a new iteration while $\sqrt{\delta_i^2 + \delta_j^2} > 10^{-7}$" (in that case, only 8 iterations are needed for the first example).
- The first displacement to estimate is around $x = -0.015$ $y = 0.038$ in pixel unit.

exercise 4

Apply the previous algorithm on every images of the image sequence. Let (D_i^t, D_j^t) be the position for the extraction of the first small image $I_o^t(i,j)$ from image $I^t(i,j)$ at instant t, repeat the following steps:

1. Extract the first small image $I_o^t(i,j)$ from $I^t(i,j)$ with (D_i^t, D_j^t) (use the integer parts)
2. Estimate the displacements Δ_i and Δ_j using the algorithm of exercise 3

3. update the positions $D_i^t = D_i^t - \Delta_i$ and $D_j^t = D_j^t - \Delta_j$
4. $t = t + 1$ and go to step 1

For each processed image, after convergence, save it with a frame around the tracked object.

suggestions for the report

In your report, you can:

- Show examples of image interpolation (it can be any image).
- Apply a translation (Δ_i, Δ_j) and its inverse $(-\Delta_i, -\Delta_j)$ successively on the same image to obtain the image $I_w(i, j)$, and compare it with the original image $I(i, j)$ (to compare, you can calculate $I_w(i, j) - I(i, j)$). This should show the limitation of the bilinear interpolation.
- For a given time t, display the error image $T(i, j) - I_o^t(i, j)$ for each iteration of the Lucas Kanade algorithm, and observe that the error decreases during the estimation.
- Show situations when the tracker is not efficient.