Lab 4: Tracking

CS4243, Semester 1, 2020

TA: Kai Xu
Due on: 15-11-2020 23:59

Objective

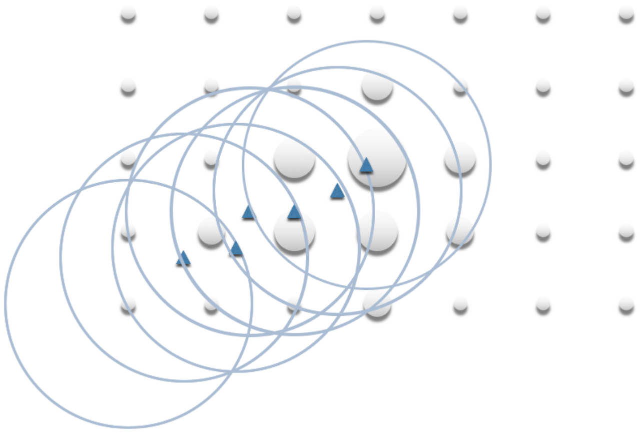
In this lab assignment, your task is to implement mean shift and Lucas–Kanade optical flow tracking algorithms. You will implement the following tasks:

1. Mean shift tracking. (30%)
   - Mean shift tracking pipeline. (25%)
   - Implement IoU metric for tracking, and report your IoU for mean shift.(5%)
2. Lucas-Kanade method for optical flow and tracking. (70%)
   - Basic Lucas-Kanade method (20 %)
   - Iterative Lucas-Kanade method.(20%)
   - Pyramid Lucas-Kanade method.(20%)
   - Object tracker pipeline and your IoU result.(10%)

Part 1: Mean Shift Tracking. (30%)

Let us first do a recap for mean shift clustering algorithm: if you recall mean shift in lab2, what we wanted to do was find modes in the density function of our data in the feature space. For the algorithm, we first define a window around it and compute the centroid of the window, then we shift the center of the window to the centroid, and repeat this procedure until the centroid stops moving.

For image, points are distributed equally everywhere, but now with each point we have an associated weight w.



The weight can be calculated by histogram back projection.

1.1 Histogram Back Projection

Back projection is a way of recording how well the pixels of a given image fit the distribution of pixels in a histogram model. In our scenario, we calculate the histogram model of the feature inside roi(region of interest) of human body and then use it to find the corresponding feature in an image.

We have completed this part for you, the value in the `dst` array represent the probability that a pixel belongs to a body area, based on the model histogram of roi that we used.

Therefore, in this part, the only thing you need to do is to tweak the parameters for histogram masking, i.e. the threshold and channel. These decide which part of the histogram of roi will be used as the reference.

☐ Tweak following parameters: `thresh_value0`, `thresh_value1`, `channel`.

1.2 Mean Shift for Tracking

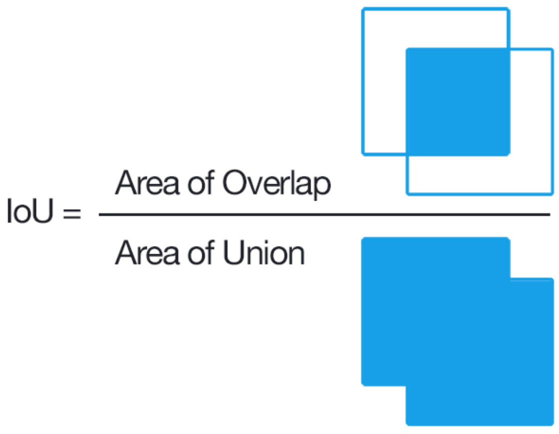
With the weight of each pixel, you can now calculate the weighted mean of the tracking window and perform iterative updates.

☐ Implement following function: `meanShift()`.

■ Prohibited function: `cv2.meanShift()`

1.3 Evaluation: Intersection over Union (IoU)

Intersection over Union (IoU) is the most commonly used evaluation metirc in the object detection area. The accuracy of the object detector is defined as follows: (credit to Adrian Rosebrock) :



☐ Implement following function: `IoU()`.

You will get full marks for IoU score > 0.65.

Part 2: Lucas-Kanade Optical Flow for Tracking. (70%)

2.1 Basic Lucas-Kanade Method

2.1.1 Keypoints Detection

In this section, we are going to implement basic Lucas-Kanade method for feature tracking. As we've discussed in the lecture, least-squares solution is "good" only when Hessian is well-conditioned, for image, i.e. corner regions. In order to do so, we use Harris corner detector to initialize the keypoints to track with Lucas-Kanade method. We have completed this part for you, you can run the cell for visualization.

2.1.2: Optical Flow Computation:

Given two consecutive frames, how can we find the flow vectors for the first frame which describe how objects move between frames? To start, we make a reasonable assumption called the **brightness constancy** assumption: the pixel intensity of a moving point stays the same between two consecutive frames with small time difference. In other words, picking any pixel of the moving can, its brightness stays approximately the same between frames–its movement should not affect its brightness after all.

Consider pixel intensity (a.k.a.brightness) $I(x, y, t)$ of a point (x, y) in the first frame t. Suppose that the point has moved to $(x + \Delta x, y + \Delta y)$ after Δt. According to the brightness constancy assumption, we can relate intensities of the point in the two frames using the following equation:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

This equation simply states that the point that we picked will have the same intensity even after it moves in space (Δx and Δy) and between frames (Δt). From this simple assumption, we can derive what is known as the **optical flow equation**. For a given point for any frame, the optical flow equation is given by:

$$I_x(\mathbf{p})v_x + I_y(\mathbf{p})v_y + I_t(\mathbf{p}) = 0$$

Here, I_x, I_y and I_t are partial derivatives of pixel intensity I. Meanwhile, v_x and v_y are **flow vectors** in the $x-$ and $y-$direction, respectively. These are the vectors we care about! If we can solve for these two values, we will be able to describe the motion of any object between frames.

One issue with the optical flow equation is that there are two unknowns that we want to solve for (v_x and v_y). This problem is known as the aperture problem. In other words, just looking an "aperture" at one pixel at a time, it is impossible to discern the true direction of motion of the object in question.

The Lucas–Kanade method solves this problem by adding another assumption: **spatial coherence**. That is, that the motion of the image contents between two frames is approximately constant within a neighborhood of the point p under consideration.

Consider a neighborhood of p, $N(p) = \{p_1, ..., p_n\}$ (e.g. 3x3 window around p). Adding the spatial coherence assumption to the optical flow equation, we see that the following should be satisfied:

For every $p_i \in N(p)$,

$$I_x(p_i)v_x + I_y(p_i)v_y = -I_t(p_i)$$

These equations can be written in matrix form $Av = b$, where

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \quad v = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \quad b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \vdots \\ -I_t(p_n) \end{bmatrix}$$

We can now solve for the flow vectors (now represented as v) by solving the following least-squares problem: $\|Av - b\|_2$. You may use `np.linalg.lstsq` to solve this equation.

☐ Implement following function: `lucas_kanade()`.

2.1.2 Keypoints Tracking

Now we can use Lucas-Kanade method to track keypoints across multiple frames. The idea is simple: compute flow vectors at keypoints in i-th frame, and add the flow vectors to the points to keep track of the points in $i+1$-th frame. We have provided the function 'track_features' for you.

Instead of keeping these 'bad' tracks, we would want to somehow declare some points are 'lost' and just discard them. One simple way to is to compare the patches around tracked points in two subsequent frames. If the patch around a point is not similar to the patch around the corresponding point in the next frame, then we declare the point to be lost. Here, we are going to use mean squared error between two normalized patches as the criterion for lost tracks. You should first normalize (here normalization means standardization, i.e. $\frac{X-\mu}{\sigma}$) each patch and then compute MSE between them.

☐ Implement following function: `compute_error()`.

0.1 2.2 Pyramidal Lucas-Kanade Feature Tracker

2.2.1 Iterative Lucas-Kanade Method

Below is the step-by-step description of Iterative Lucas-Kanade Method:

Let $p = \begin{bmatrix} p_x & p_y \end{bmatrix}^T$ be a point on frame I. g be the flow vector guessed from previous pyramid level(initialized as zero vector). The goal is to find flow vector $v = \begin{bmatrix} v_x & v_y \end{bmatrix}^T$ such that $p + v$ is the corresponding point of p on the next frame J.

- Initialize flow vector:

$$v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Compute spatial gradient matrix:

$$G = \sum_{x=p_x-w}^{p_x+w} \sum_{y=p_y-w}^{p_y+w} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y^2(x,y) \end{bmatrix}$$

- for $k = 1$ to K

```
    - Compute temporal difference: $\delta I_k(x,y) = I(x,y) - J(x+g_x+v_x, y+g_y+v_y)$
    - Compute image mismatch vector:
```

$$b_k = \sum_{x=p_x-w}^{p_x+w} \sum_{y=p_y-w}^{p_y+w} \begin{bmatrix} \delta I_k(x,y) I_x(x,y) \\ \delta I_k(x,y) I_y(x,y) \end{bmatrix}$$

```
    - Compute optical flow: $v^k = G^{-1} b_k$
    - Update flow vector for next iteration: $v := v + v^k$
- Return $v$
```

☐ Implement following function: `iterative_lucas_kanade()`.

2.2.2 Pyramidal Lucas-Kanade Method

Following is the description of pyramidal Lucas-Kanade algorithm:
Let p be a point on image I and s be the scale of pyramid representation.

```
- Build pyramid representations of $I$ and $J$: $\{I^L\}_{L=0,\ldots,L_m}$ and $\{J^L\}_{L=0,\ldots,L_m}$
- Initialize pyramidal guess $g^{L_m} = \begin{bmatrix} g_x^{L_m} & g_y^{L_m} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$
- for $L = L_m$ to 0 with step of -1
    - Compute location of $p$ on $I^L$: $p^L = p/s^L$
    - Let $d^L$ be the optical flow vector at level $L$:
          $d^L := IterativeLucasKanade(I^L, J^L, p^L, g^L)$
    - Guess for next level $L-1$: $g^{L-1} = s(g^L + d^L)$
- Return $d = g^0 + d^0$
```

☐ Implement following function: `pyramid_lucas_kanade()`.

■ Prohibited function: `cv2.calcOpticalFlowPyrLK()`

2.4 Lucas-Kanade Object Tracking

Now let us build a simple object tracker using Lucas-Kanade method you have implemented. In order to test the object tracker, we provide you a short face-tracking sequence. Each frame in the sequence is annotated with the ground-truth location (as bounding box) of face. We've already implement the tracking pipeline for you, run it and show your IoU score. You will get full marks for IoU score > 0.55.

Hand in

Files to be submitted are `lab4.py` and `lab4.ipynb`. Please zip them into a file named `AXXX1_AXXX2_AXXX3.zip`, where AXXX is the student number of the group members. Each group should submit only once. Groups with missing files or incorrectly formatted code that does not run will be penalized. The submission deadline is `15/11/2020,23:59`. Q&A sessions for Lab 2 will be held on `10/11/2020,15:00-17:00 13/11/2020,09:00-11:00`.