Homework 4
M1522.001000 Computer Vision (2019 Spring)
Due: Thursday May 16 2:00PM

The goal of this homework is to implement an affine Lucas-Kanade tracker. Your tracker will be able to track patches over frames in videos provided changes in the patch's appearance due to the viewpoint and movement of an object.

1 Preliminaries

An image transformation or warp is an operation that acts on pixel coordinates and moves pixel values from one place to another in an image. Translation, rotation and scaling are all examples of warps. We will use the symbol W to denote warps. A warp function W has a set of parameters p associated with it and maps a pixel with coordinates $x = [u\ v]^T$ to $x' = [u'\ v']^T$.

$$x' = W(x; p) \tag{1}$$

Warps can be composed, that is, after applying a warp $W(x; p)$ to an image, another warp $W(x; q)$ can be applied to the warped image. The resultant (combined) warp is

$$W(x; q) \circ W(x; p) = W(W(x; p), q) \tag{2}$$

An affine transform is a warp that can include any combination of translation, anisotropic scaling and rotations. An affine warp can be parameterized in terms of 6 parameters $p = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6]'$. One of the convenient things about an affine transformation is that it is linear, its action on a point with coordinates $x = [u\ v]^T$ can be described as a matrix operation

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = W(p) \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3}$$

where $W(p)$ is a 3×3 matrix such that

$$W(p) = \begin{bmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Note that for convenience, when we want to refer to the warp as a function we will use $W(x; p)$ and when we want to refer to the matrix for an affine warp we will use $W(p)$. Since affine warps can be implemented as matrix multiplications, composing two affine warps reduces to multiplying their corresponding matrices

$$W(x; q) \circ W(x; p) = W(W(x; p), q) = W(W(p)x, q) = W(q)W(p)x \tag{5}$$

An affine transform can also be inverted, that is for a given affine warp $W(p)$ we can find another affine warp that reverses $W(p)$ and recovers the original image. The inverse

warp of $W(p)$ is simply the matrix inverse of $W(p)$, $W(p)^{-1}$. In this assignment it will sometimes be simplerto consider an affine warp as a set of 6 parameters in a vector p and it will sometimes be eaiser to work with the matrix version $W(p)$. Fortunately, switching between these two forms is easy (Equation 4).

A Lucas Kanade tracker maintains a warp $W(x;p)$ which aligns an image I_t to a template T . We denote pixel locations by x, so $I(x)$ is the pixel value at location x in image I. For the purposes of this derivation, I and T are treated as column vectors (think of them as unrolled image matrices). $W(x;p)$ is the point obtained by warping x with a transform that has parameters p. W can be any transformation that is continuous in its parameters p. Examples of valid warp classes for W include translations (2 parameters), affine transforms (6 parameters) and full projective transforms (8 parameters). The Lucas Kanade tracker minimizes the pixel-wise sum of square difference between the warped image $I(W(p))$ and the template T .

$$L = \sum [T(x) - I(W(x;p))]^2 \tag{6}$$

The minimization is performed using an iterative procedure that making a small change $(\Delta p)$to p at each iteration. It is computationally more efficient to do the minimization by finding the Δp that helps align the template to the image, then applying the inverse warp to the image. Hence at each step, we want to find the Δp to minimize

$$L \approx \sum_x [T(W(x;\Delta p)) - I(W(x;p))]^2 \tag{7}$$

For tracking a patch template, the summation is performed only over the pixels lying inside the template region. We can expand $T(W(x;\triangle p))$ in terms of its first order linear approximation to get

$$L \approx \sum_x \left[T(x) + \nabla_{\Delta p}(T(W(x;p)))\Delta p - I(W(x;p)) \right]^2 \tag{8}$$

The middle term can be split using the chain rule into two parts, one for the image gradients in the image gradients in the template (∇T) and another for the Jacobian of the warp with respect to its parameters.

$$L \approx \sum_x \left[T(x) + \nabla T(x) \frac{\partial W}{\partial p} \Delta p - I(W(x;p)) \right]^2 \tag{9}$$

Where $\nabla T(x) = [\frac{\partial T(x)}{\partial u} \ \ \frac{\partial T(x)}{\partial v}]$. To minimize we need to take the derivative of L and set it to zero.

$$\frac{\partial L}{\partial \Delta p} = 2 \sum_x \left[\nabla T(x) \frac{\partial W}{\partial p} \right]^T \left[T(x) + \nabla T(x) \frac{\partial W}{\partial p} \Delta p - I(W(x;p)) \right] \tag{10}$$

Setting to zero, switching from summation to vector notation and solving for Δp we get

$$\Delta p = H^{-1} G^T [I_{warped} - T] \tag{11}$$

where G is the gradient of $T(W(x;\Delta p))(=\nabla T\frac{\partial W}{\partial p})$, H is the approximated Hessian $H = G^T G$ and I_{warped} is $I(W(x;p))$.

Once Δp has been solved for, it needs to be inverted and composed with p to get the new warp parameters for the next iteration.

$$W(x;p) \leftarrow W(x;p) \circ W(x;\Delta p)^{-1} \tag{12}$$

The next iteration solves Equation 11 starting with the new value of p. Possible termination criteria include the absolute value of Δ p falling below some value or running for some fixed number of iterations

2 Submitting Your Assignment

Your submission for this assignment consists of answers to a few theory questions, the code for your MATLAB implementation and a short writeup describing any thresholds and parameters used, interesting observations you made or things you did differently while implementing the assignment. The answers to the theory questions in Section 3 should be in a plaintext file or a pdf named `theory.txt` or `.pdf`. Each of the MATLAB functions you wrote should be in a folder named `matlab`, upload only `.m` files to this folder and make sure all the files needed for your code to run (except data) are included. The writeup describing your experiments (refer to Section 5) should be **typed** with **English**. You need to zip the following items, name it as `(your student ID)-(your name)-HW4 .zip` or `.tar.gz` and send it to TA's email (ta.cv@vision.snu.ac.kr) with title as `"(your student ID)-(your name)-HW4"`. Your zip file should be sent before the beginning of the class of the due date. Later than that, you will use one late day.

Your submitted zip file should include the files arranged in this layout:

- `theory.txt` (or `.pdf`): answers to the theory questions in Section 3
- `experiments.txt` or (or `.pdf`): the writeup describing your experiments
- Folder `data`
  - Folder `car`, Flolder `landing`
  - `initTest.mat`
- Folder `matlab`
  - `affineLKDemo.m`, `warpImage.m` (*already given*).
  - `warpImageMasked.m` `initAffineLKTracker.m`, `affineTrackerMasked.m`,
  - `affineLKDemoUpdate.m`
  - You can also include any extra helper functions.

Refer to Section 4 and Section 5 for the details of the above files.

3 Theory Questions

Question 1: Calculating the Jacobian (5 points)

Assuming the affine warp model defined in Equation 4, derive the expression for the Jacobian Matrix of the warp ($= \frac{\partial W}{\partial p}$) in terms of the u, v in Equation 3.

Question 2: Calculating the Gradient (5 points)

Derive the expression for the Gradient G defined in Equation 11 in terms of the u, v.

Question 3: Computational complexity of initialization (5 points)

Explain computational complexity of G and H and Express your answers in terms of n, m and p where n is the number of pixels in the template T, m is the number of pixels in an input image I and p is the number of parameters used to describe the warp W.

Question 4: Computational complexity of iteration (5 points)

Find the computational complexity (Big O notation) for each iteration of the Lucas Kanade Tracker. Parameter settings are the same as in Question 3.

Question 5: Practical Issue I (5 points)

A true derivative cannot be computed in practise on pixel-discretized images. Devise a method to overcome this limitation.

Question 6: Practical Issue II (5 points)

Lucas-Kanade tracker formulates search as an optimisation problem which maximizes the similarity to the template. This function (L), however, can be non-convex over the image. Devise a method to find a good starting point near solution.

4 Programming

The included script file `affineLKDemo.m` handles reading in images, template region marking, making tracker function calls and displaying output onto the screen. The function prototypes provided are guidelines, you can add new arguments if you feel they are needed. Just make sure that your code runs with the script or your modified version of the script, that your script generates the outputs we are looking for (a frame sequence with the bounding box of the target being tracked on each frame) and that its easy to change the input data directory to your script. Two videos are included for testing the tracker, one of a car and another from a helicopter approaching a runway.

We provide following two data folders in `hw4.zip`.

- Folder `car (data/car/)` : It includes short car video sequence: The video sequence is the data for 261 car driving images, and the size of each image is (480×720).

- Folder `landing (data/landing/)` : It includes landing video sequence : This video was taken from a helicopter during a runway approach. It include 120 video frame, and the size of each is (480×720).

Part 1 Warping the Template Region (5 points)

Write the function `warpImageMasked()` that warping the pixels lying inside the template region.

```
function [img_warped] = warpImageMasked(img, mask, W)
```

The function will input a grayscale image `(img)` along with a logical mask image `(mask)` and a warping matrix `(W)`. The `mask` is true at pixels that lie inside the template region and are hence part of the tracking template and is false everywhere else. The function should output an image `(img_warped)` warped template area.

Part 2 Initializing the Tracker (15 points)

Wrtie the function `initAffineLKTracker()` that initializes the LK tracker by computing important matrices needed to track a template patch.

```
function [affineLKContext] = initAffineLKTracker(img, msk)
```

The function will input a greyscale image `(img)` along with a logical mask image `(msk)` that has the same size as `img`. The function should output a MATLAB structure affineLKContext that contains the Gradient and the inverse of the approximated Hessian matrix (G and H^{-1} in Equation 11). When unrolling images into vectors, follow the standard MATLAB convention of going column by column (ie. use `img(:)`). To make sure your implementation is correct, load `initTest.mat` and run your function with the image in `initTest.mat` and compare your result with reference in `initTest.mat`.

Part 3 The Main Tracker (20 points)

Write the function `affineTrackerMasked` that does the actual template tracking.

```
function Wout = affineTrackerMasked(img, tmp, mask, W, context)
```

The function will input a greyscale image of the current frame (img), the template image `(tmp)`, the logical mask `(msk)` that marks out the template region in `tmp`, The affine warp matrix (W) for the previous frame and the precomputed G and H^{-1} matrices. The function should output a 3×3 matrix W_{out} that contains the new affine warp matrix updated so that it aligns the current frame with the template. You can either use a fixed number of gradient descent iterations or formulate a stopping criteria for the algorithm. You can use the included image warping function `warpImage` or your implementation in Part 1 to apply warp to entire image or only template region.

Part 4 Periodic Template Updates (5 points)

Refer to `affineLKDemo.m` and write a MATLAB script file `affineLKDemoUpdate.m` which does not update the template image during some image frames or entire tracking. The update cycle should be parameterized.

5 Experiments

Tracking a Car (5 points)

Test your tracker on the short car video sequence (`data/car/`) by running the wrapper function `affineLKDemo`. What kinds of templates work well for tracking? At what point does the tracker break down? Why does this happen?

Tracking Runway Markings (5 points)

Try running your tracker on the landing video (`data/landing/`). This video was taken from a helicopter during a runway approach. With a model of the landing zone markings (the lengths of the line segments etc) the output of the tracker can be used to estimate the camera position and hence the helicopter's position with respect to a landing zone and can be used in an automated landing system.

Template Update (5 points)

Check how the update cycle of template image affects tracking performance. What are the pros and cons of frequent template image updates.