

Assignment 3

Computer Vision 15–385, Spring 12

Due Date: Tuesday 03/20/2012

Total Points: 85

In this assignment you will be implementing an affine Lucas-Kanade tracker. Your code will be able to track patches over frames in videos provided changes in the patch’s appearance due to viewpoint and illumination variations aren’t too severe. Two videos are included for testing the tracker, one of a car and another from a helicopter approaching a runway.

In this assignment, to initiate the tracker you need to define a template by drawing a bounding box around the object to be tracked on the first frame of the video. For each of the subsequent frames the tracker will update an affine transform that warps the current frame so that the template in the first frame is aligned with the warped current frame.

We have provided a wrapper function that reads frames from a video, provides an interface for marking out the bounding box of the template on the first frame and displays the output for each frame. Your job is to add the actual tracking functionality to the code. You will need to write two functions (along with any helper functions you need). The first function will run only once and initialize the tracker for the template patch on the first frame. The other function will run on every frame and update the affine transform that aligns the current frame with the template.

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 $x' = [u' \ v']^T$.

$$x' = W(x; p) \quad (1)$$

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

$$W(x; q) \circ W(x; p) = W(W(x; p), q) \quad (2)$$

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

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

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

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

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

$$W(x; q) \circ W(x; p) = W(W(x; p), q) = W(W(p)x, q) = W(q)W(p)x \quad (5)$$

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 simpler to consider an affine warp as a set of 6 parameters in a vector p and it will sometimes be easier 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 a sequence of images 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 .

$$L = \sum_x [T(x) - I(W(x; p))]^2 \quad (6)$$

The minimization is performed using an iterative procedure that making a small change (Δp) 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

$$L = \sum_x [T(W(x; \Delta p)) - I(W(x; p))]^2 \quad (7)$$

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

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

Table 1: Summary of Variables

Symbol	Vector/Matrix Size	Description
u	1×1	Image horizontal coordinate
v	1×1	Image vertical coordinate
x	2×1 or 1×1	pixel coordinates: (u, v) or unrolled
I	$m \times 1$	Image unrolled into a vector (m pixels)
T	$m \times 1$	Template unrolled into a vector (m pixels)
$W(p)$	3×3	Affine warp matrix
p	6×1	parameters of affine warp
$\frac{\partial T}{\partial u}$	$m \times 1$	partial derivative of image wrt u
$\frac{\partial T}{\partial v}$	$m \times 1$	partial derivative of image wrt v
∇T	$m \times 2$	image gradient $\nabla T(x) = \begin{bmatrix} \frac{\partial T(x)}{\partial u} & \frac{\partial T(x)}{\partial v} \end{bmatrix}$
$\frac{\partial W}{\partial p}$	2×6	Jacobian of affine warp wrt its parameters
J	$m \times 6$	Jacobian of error function L wrt p
H	6×6	Pseudo Hessian of L wrt p

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.

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

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

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

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

$$\Delta p = H^{-1} J^T [I_{warped} - T] \quad (11)$$

where J is the Jacobian of $T(W(x; \Delta p))$, $J = \nabla T \frac{\partial W}{\partial p}$, H is the approximated Hessian $H = J^T J$ and I_{warped} is $I(W(x; p))$. Note that for a given template, the Jacobian J and Hessian H are independent of p . This means they only need to be computed once and then they can be reused during the entire tracking sequence.

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.

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

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

2 Submitting Your Assignment

Your submission for this assignment will comprise 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 `theory.pdf`. Each of the MATLAB functions you write as described in Section 4 along with any extra helper 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 in a plaintext file or a pdf named `experiments.txt` or `experiments.pdf`. If you attempt the extra credit section include a folder named `extra` with relevant data you created, results generated and a short description of what you did.

A directory has been created on for uploading all your course related files. The directory can be found at `afs/cs.cmu.edu/academic/class/15385-s12-users/andrewid` (for graduate students, the folder is `15685-s12-users`). Inside your submission directory, create a subfolder named `p2` for this assignment. Do not add any extra layers of indirection to your directory structure. Your final upload should have the files arranged in this layout:

- `afs/cs.cmu.edu/academic/class/15385-s12-users/andrewid`
 - `p3`
 - * `theory.txt` or `theory.pdf`
 - * `experiments.txt` or `experiments.pdf`
 - * `matlab`
 - `initAffineLKTracker.m`
 - `affineLKTracker.m`
 - `affineLKDemo.m` (already provided)
 - `warpImageMasked.m` (already provided)
 - * `extra` (*optional*)
 - `initAffineLKTrackerPyramid.m`
 - `affineLKTrackerPyramid.m`
 - `initAffineLKTrackerRobust.m`
 - `affineLKTrackerRobust.m`

You may need to run `aklog cs.cmu.edu` when you log in to be able read and write from your submission directory. Your files are due by 23:59:59 on the submission day. Please check to

make sure you have write permissions to your submission folder ahead of time so that problems (if any) can be fixed. We will be using timestamps to determine submission times and for late day counting, so do not modify your files after the submission deadline unless you wish to use a late day.

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 J in terms of the warp parameters $p = [p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ p_6]'$.

Question 2: Computational complexity

(10 points)

Find the computational complexity (Big O notation) for the initialization step (Precomputing J and H^{-1}) and for each runtime iteration (Equation 11) or the Lucas Kanade Tracker. 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 .

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.

Initializing the Tracker

(30 points)

Write the function `initAffineLKTracker()` that initializes the LK tracker by precomputing 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 is the same size as `img`. The mask is *true* at pixels that lie inside the user defined bounding box and are hence part of the tracking template and is *false* everywhere else.

The function should output a MATLAB structure `affineLKContext` that contains the Jacobian of the affine warp with respect to the 6 affine warp parameters and the inverse of the approximated Hessian matrix (J 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, run

your code on `initTest.pgm` using the entire image as the template region and compare your J and H^{-1} matrices to those in `initTest.mat`. Each column of the Jacobian can be reshaped into an image which should make the visualization more intuitive.

The Main Tracker

(30 points)

Write the function `affineTracker()` that does the actual template tracking.

```
function Wout = affineTracker(img, tmp, mask, Win, 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 for the previous frame (`Win`) and the precomputed J and H^{-1} matrices `context`.

The function should output the 3×3 matrix `Wout` 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 to apply affine warps to images.

5 Experiments

Tracking a Car

(5 points)

Test your tracker on the short car video sequence (`data/car1/`) by running the wrapper function `affineLKDemo`. What sort 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 video¹ (`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.

6 Extra Credit

Option 1: Adding Illumination Robustness The LK tracker as it is formulated now breaks down when there is a change in illumination because the sum of squared distances error it tries to minimize is sensitive to illumination changes. There are a couple of things you could try to do to fix this. Perhaps the simplest is to scale the brightness of pixels in each frame so that the average brightness of pixels in the tracked region stays the same as the average

¹This dataset is courtesy Prof. Sanjiv Singh and Sebastian Scherer at the Field Robotics Center who work on autonomous helicopters

brightness of pixels in the template. Test your new tracker on the car video sequence again (data/car/).

Use the same function names for the initializer and tracker with 'Robust' appended. Include both versions of the code in your submission.

Option 2: LK Tracking on an Image Pyramid If the target being tracked moves a lot between frames, the LK tracker can break down. One way to mitigate this problem is to run the LK tracker on an image pyramid instead of a single image. The Pyramid tracker starts by performing tracking on a higher level (smaller image) to get a course alignment estimate, projecting this down into the lower level and repeating until a fine aligning warp has been found at the lowest level of the pyramid (the original sized image). In addition to being more robust, the pyramid version of the tracker is much faster because it needs to run fewer gradient descent iterations on the full scale image. Implement a Pyramid tracker.

Use the same function names as before but append 'Pyramid' to the function names while submitting your code. Include both versions of the code in your submission.