Fundamentals of Computer Vision

Project 3

Tracking Objects in Videos

Instructions

1. **Integrity and collaboration:** Students are encouraged to work in groups but each student must submit their own work. If you work as a group, include the names of your collaborators in your write-up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.

2. **Start early!** Running the code on the videos can take a lot of time, making debugging very slow.

3. **Verify your implmenentation as you proceed!** Otherwise you risk having a huge mess of malfunctioning code that can go wrong anywhere.

4. **Questions:** If you have any question, please look at Discussion on canvas first. Other students may have encountered the same problem, and might have been solved already. If not, post your question on the discussion board. TA will respond as soon as possible.

5. **Write-up:** Your write-up should mainly consist of two parts, your answers to theory questions and your insights as you attempt the programming questions. Specific items to be included in the write-up are mentioned in each question.

6. **Code:** Stick to the function prototypes mentioned in the handout. This makes verifying code easier for the TA. If you do want to change a function prototype or add an extra parameter, please talk to the TA.

7. **Submission:** Your submission for this assignment should be a zip file, <First_FmilyName.zip>. composed of your write-up, your Matlab implementations (including helper functions). Please make sure to remove the data/ folder and any other files that are not required. Ensure that your submission is of a reasonable size. You may want to use video compression if your videos are huge.

   Your final upload should have the files arranged in this layout:

- <First_FmilyName>.zip
  - <First_FmilyName>/
    - <First_FmilyName>.pdf
    - matlab/
      - LucasKanade.m.m
      - affineMBTracker.m
      - initAffineMBTracker.m
      - lk_demo.m
      - mb_demo.m
    - results/
      - car.mp4

Figure 1: Sample images from the video sequence provided

Overview

One incredibly important aspect of human and animal vision is the ability to follow objects and people in our view. Whether it is a tiger chasing its prey, or you trying to catch a basketball, tracking is so integral to our everyday lives that we forget how much we rely on it. In this assignment, you will be implementing an algorithm that will track an object in a video.

You will first implement the Lucas-Kanade tracker, and then a more computationally efficient version called the Matthew-Baker (or inverse compositional) method [1]. This method is one of the most commonly used methods in computer vision due to its simplicity and wide applicability. We have provided two video sequences: a car on a road, and a helicopter approaching a runway.

To initialize the tracker you need to define a template by drawing a bounding box around the object to be tracked in 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.

Preliminaries

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

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

An affine transform is a warp that can include any combination of translation, anisotropic scaling and rotations. An affine warp can be parametrized in terms of 6 parameters $\mathbf{p} =$

$[p_1\ p_2\ p_3\ p_4\ p_5\ p_6]^T$. One of the convenient things about an affine transformation is that it is linear; its action on a point with coordinates $\mathbf{x} = [u\ \ v]^T$ can be described as a matrix operation

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \mathbf{W}(\mathbf{p}) \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2}$$

Where $\mathbf{W}(\mathbf{p})$ is a 3×3 matrix such that

$$\mathbf{W}(\mathbf{p}) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Note that for convenience when we want to refer to the warp as a function we will use $\mathbf{W}(\mathbf{x};\mathbf{p})$ and when we want to refer to the matrix for an affine warp we will use $\mathbf{W}(\mathbf{p})$. Table 1 contains a summary of the variables used in the next two sections. It will be useful to keep these in mind.

Table 1: Summary of Variables

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

Lucas-Kanade: Forward Additive Alignment

A Lucas Kanade tracker maintains a warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ which aligns a sequence of images $\mathbf{I}_t$ to a template $\mathbf{T}$. We denote pixel locations by $\mathbf{x}$, so $\mathbf{I}(\mathbf{x})$ is the pixel value at location $\mathbf{x}$ in image $\mathbf{I}$. For the purposes of this derivation, $\mathbf{I}$ and $\mathbf{T}$ are treated as column vectors (think of them as unrolled image matrices). $\mathbf{W}(\mathbf{x};\mathbf{p})$ is the point obtained by warping $\mathbf{x}$ with a transform that has parameters $\mathbf{p}$. $\mathbf{W}$ can be any transformation that is continuous in its parameters $\mathbf{p}$. Examples of valid warp classes for $\mathbf{W}$ 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 $\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))$ and the template $\mathbf{T}$.

In order to align an image or patch to a reference template, we seek to find the parameter vector $\mathbf{p}$ that minimizes L, where:

$$L = \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{x}\right) - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right]^2 \tag{4}$$

In general this is a difficult non-linear optimization, but if we assume we already have a close estimate $\mathbf{p}$ of the correct warp, then we can assume that a small linear change $\Delta\mathbf{p}$ is enough to get the best alignment. This is the forward additive form of the warp. The objective can then be written as:

$$L = \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{x}\right) - I\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}\right)\right)\right]^2 \tag{5}$$

Expanding this to the first order with Taylor Series gives us:

$$L \approx \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{x}\right) - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}\right)\right) - \nabla\mathbf{I}(\mathbf{x})\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p}\right]^2 \tag{6}$$

Here $\nabla\mathbf{I}(\mathbf{x}) = \left[\frac{\partial\mathbf{I}(\mathbf{x})}{\partial u}\frac{\partial\mathbf{I}(\mathbf{x})}{\partial v}\right]$, which is the vector containing the horizontal and vertical gradient at pixel location $\mathbf{x}$. Rearranging the Taylor expansion, it can be rewritten as a typical least squares approximation $\Delta\mathbf{p}^* = \underset{\Delta p}{\operatorname{argmin}}||A\Delta p - b||^2$

$$\Delta\mathbf{p}^* = \underset{\Delta\mathbf{p}}{\operatorname{argmin}} \sum_{\mathbf{x}} \left[\nabla\mathbf{I}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p} - \left\{\mathbf{T}\left(\mathbf{x}\right) - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right\}\right]^2 \tag{7}$$

This can be solved with $\Delta\mathbf{p}^* = (A^TA)^{-1}A^Tb$ where:

$$(A^TA) = \mathbf{H} = \sum_{\mathbf{x}} \left[\nabla\mathbf{I}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^T \left[\nabla\mathbf{I}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right] \tag{8}$$

$$A = \sum_{\mathbf{x}} \left[\nabla\mathbf{I}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right] \tag{9}$$

$$b = \mathbf{T}\left(\mathbf{x}\right) - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}\right)\right) \tag{10}$$

Once $\Delta\mathbf{p}$ is computed, the best estimate warp can be updated $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$, and the whole procedure can be repeated again, stopping when $\Delta\mathbf{p}$ is less than some threshold.

Matthew-Baker: Inverse Compositional Alignment

While Lucas-Kanade alignment works very well, it is computationally expensive. The inverse compositional method is similar, but requires less computation, as the Hessian and Jacobian only need to be computed once. One caveat is that the warp needs to be invertible. Since affine warps are invertible, we can use this method.

In the previous section, we combined two warps by simply adding one parameter vector to another parameter vector, and produce a new warp $\mathbf{W}(\mathbf{x}, \mathbf{p}+\mathbf{p}')$. Another way of combining warps is through composition of warps. After applying a warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ to an image, another warp $\mathbf{W}(\mathbf{x};\mathbf{q})$ can be applied to the warped image. The resultant (combined) warp is

$$\mathbf{W}(\mathbf{x};\mathbf{q}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}) = \mathbf{W}(\mathbf{W}(\mathbf{x};\mathbf{p}),\mathbf{q}) \tag{11}$$

Since affine warps can be implemented as matrix multiplications, composing two affine warps reduces to multiplying their corresponding matrices

$$\mathbf{W}(\mathbf{x};\mathbf{q}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}) = \mathbf{W}(\mathbf{W}(\mathbf{x};\mathbf{p}),\mathbf{q}) = \mathbf{W}(\mathbf{W}(\mathbf{p})\mathbf{x},\mathbf{q}) = \mathbf{W}(\mathbf{q})\mathbf{W}(\mathbf{p})\mathbf{x} \tag{12}$$

An affine transform can also be inverted. The inverse warp of $\mathbf{W}(\mathbf{p})$ is simply the matrix inverse of $\mathbf{W}(\mathbf{p})$, $\mathbf{W}(\mathbf{p})^{-1}$. In this assignment it will sometimes be simpler to consider an affine warp as a set of 6 parameters in a vector $\mathbf{p}$ and it will sometimes be easier to work with the matrix version $\mathbf{W}(\mathbf{p})$. Fortunately, switching between these two forms is easy (Equation 3).

The minimization is performed using an iterative procedure by making a small change ($\Delta\mathbf{p}$) to $\mathbf{p}$ at each iteration. It is computationally more efficient to do the minimization by finding the $\Delta\mathbf{p}$ that helps align the template to the image, than applying the inverse warp to the image. This is because the image will change with each frame of the video, but the template is fixed at initialization. We will see soon that doing this allows us to write the Hessian and Jacobian in terms of the template, and so this can be computed once at the beginning of the tracking. Hence at each step, we want to find the $\Delta\mathbf{p}$ to minimize

$$L = \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{W}\left(\mathbf{x};\Delta\mathbf{p}\right)\right) - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right]^2 \tag{13}$$

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

$$L \approx \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{x}\right) + \nabla\mathbf{T}(\mathbf{x})\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p} - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right]^2 \tag{14}$$

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

$$\frac{\partial L}{\partial\Delta\mathbf{p}} = 2\sum_{\mathbf{x}} \left[\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^T \left[\mathbf{T}\left(\mathbf{x}\right) + \nabla\mathbf{T}(\mathbf{x})\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p} - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}\right)\right)\right] \tag{15}$$

Setting to zero, switching from summation to vector notation and solving for $\Delta\mathbf{p}$ we get

$$\Delta\mathbf{p} = \mathbf{H}^{-1}\mathbf{J}^T\left[\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p})) - \mathbf{T}\right] \tag{16}$$

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

Once $\Delta\mathbf{p}$ has been solved for, it needs to be inverted and composed with $\mathbf{p}$ to get the new warp parameters for the next iteration.

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1} \tag{17}$$

The next iteration solves Equation 16 starting with the new value of $\mathbf{p}$. Possible termination criteria include the absolute value of $\Delta\mathbf{p}$ falling below some value or running for some fixed number of iterations.

1 Theory Questions (25 points)

Type down your answers for the following questions in your write-up. Each question should only take a couple of lines. In particular, the "proofs" do not require any lengthy calculations. If you are lost in many lines of complicated algebra you are doing something much too complicated (or wrong).

Q1.1: Calculating the Jacobian (15 points)

Assuming the affine warp model defined in Equation 3, derive the expression for the Jacobian Matrix $\mathbf{J}$ in terms of the warp parameters $\mathbf{p} = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6]'$.

Q1.2: Computational complexity (10 points)

Find the computational complexity (Big O notation) for the initialization step (Pre-computing $\mathbf{J}$ and $\mathbf{H}^{-1}$) and for each runtime iteration (Equation 16) of the Inverse Compositional method. Express your answers in terms of n, m and p where n is the number of pixels in the template $\mathbf{T}$, m is the number of pixels in an input image $\mathbf{I}$ and p is the number of parameters used to describe the warp W. How does this compare to the run time of the regular Lucas-Kanade method?

2 Lucas-Kanade Tracker (60 points)

For this section, TA will grade your tracker based on the performance you achieved on the two provided video sequences: (1) `data/car1/`. The provided script files `lk demo.m` and `mb demo.m` handle reading in images, template region marking, making tracker function calls and displaying output onto the screen. The function prototypes provided are guidelines. Please make sure that your code runs functionally with the original script and generates the outputs we are looking for (a frame sequence with the bounding box of the target being tracked on each frame) so that we can replicate your results.

Note that the only thing TA would do for you during grading is change the input data directory, and initialize your tracker based on what you mentioned in your write-up. Please submit one video for each of them in the `results/` directory, with file name `car.mp4`. Also, please mention the initialization coordinates of your tracker for both video sequences in your write-up and in your code.

Q2.1: Write a Lucas-Kanade Tracker for a Flow Warp (20 points)

Write the function with the following function signature:

```
[u,v] = LucasKanade(It, It1, rect)
```

that computes the optimal local motion from frame $\mathbf{I}_t$ to frame $\mathbf{I}_{t+1}$ that minimizes Equation 1. Here `It` is the image frame $\mathbf{I}_t$, `It1` is the image frame $\mathbf{I}_{t+1}$, and `rect` is the 4×1 vector that represents a rectangle on the image frame `It`. The four components of the rectangle are `[x, y, w, h]`, where `(x, y)` is the top-left corner and `(w, h)` is the width and height of the bounding box. The rectangle is inclusive, i.e., in includes all the four corners. To deal with fractional movement of the template, you will need to interpolate the image using the Matlab function `interp2`. You will also need to iterate the estimation until the change in warp parameters `(u, v)` is below a threshold. Use the forward compositional (Lucas-Kanade method) for this question.

Q2.2: Initializing the Matthew-Baker Tracker (10 points)

Write the function `initAffineMBTracker()` that initializes the inverse compositional tracker by precomputing important matrices needed to track a template patch.

```
function [affineMBContext] = initAffineMBTracker(img, rect)
```

The function will input a greyscale image (`img`) along with a bounding box (`rect`) (in the format `[x y w h]`).

The function should output a Matlab structure `affineMBContext` that contains the Jacobian of the affine warp with respect to the 6 affine warp parameters and the inverse of the approximated Hessian matrix ($\mathbf{J}$ and $\mathbf{H}^{-1}$ in Equation 16).

Q2.3: The Main Matthew-Baker Tracker (20 points)

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

```
function [Wout] = affineMBTracker(img, tmp, rect, Win, context)
```

The function will input a greyscale image of the current frame (`img`), the template image (`tmp`), the bounding box `rect` that marks the template region in `tmp`, The affine warp matrix for the previous frame (`Win`) and the precomputed $\mathbf{J}$ and $\mathbf{H}^{-1}$ 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 used 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.

Q2.4: Tracking a Car (10 points)

Test your trackers on the short car video sequence (`data/car1/`) by running the wrapper scripts `lk_demo.m` and `mb_demo.m`. What sort of templates work well for tracking? At what point does the tracker break down? Why does this happen?

In your write-up: Submit your best video of the car being tracked. Save it as `results/car.mp4`.

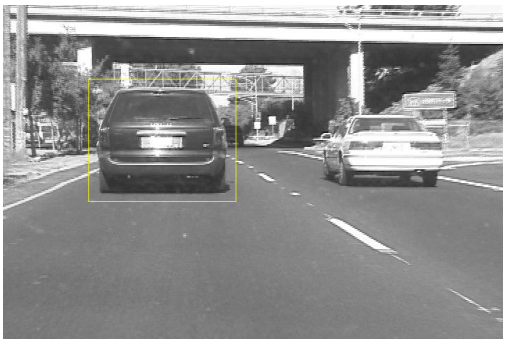

Figure 2: Tracking in the car image sequences

4 Submission Summary

- **Q1.1** Derive the expression for the Jacobian Matrix
- **Q1.2** What is the computational complexity of inverse compositional method?
- **Q2.1** Write the forward compositional tracker (LK Tracker)
- **Q2.2** Initialize the inverse compositional tracker (MB Tracker)
- **Q2.3** Write the inverse compositional tracker (MB Tracker)
- **Q2.4** Run the inverse compositional tracker on the car dataset. What templates does it work well with? When does the tracker break down? Why does this happen?
- **Q2.5** Run the inverse compositional tracker on the run markings dataset.

References

[1] Simon Baker, et al. Lucas-Kanade 20 Years On: A Unifying Framework: Part 1, CMU-RI-TR-02-16, Robotics Institute, Carnegie Mellon University, 2002

[2] Simon Baker, et al. Lucas-Kanade 20 Years On: A Unifying Framework: Part 2, CMU-RI-TR-03-35, Robotics Institute, Carnegie Mellon University, 2003

[3] Bouguet, Jean-Yves. Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the algorithm, Intel Corporation, 2001

Credit

This project is adapted directly from Ioannis Gkioulekas.