16-720-B, Computer Vision, Fall 2020

Object Tracking in Videos

Instructor:
Prof. Kris Kitani

TAs:
Rawal Khirodkar (Lead)
Nadine Chang
Alireza Golestaneh
Zhengyi Luo
Anand Bhoraskar

Due Date:
Dec 7, 2020 11:59 pm

Total Points: 110
EC Points: 20

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 prohibited and may lead to failure of this course.

2. **Questions:** Please keep an eye on the Piazza FAQ page for this homework.

3. **Write-up and Code:** Please stick to the coding conventions provided. Do not import external packages unless specified.

4. **Submission:** Your submission for this assignment should be a zip file, `<andrew-id.zip>`, composed of your write-up, your Python implementations (including helper functions), and your implementations, results for extra credit (optional). 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:

   - `<AndrewID>.zip`
     - `<AndrewId>/`
       - `<AndrewId>.pdf`
       - `python/`
         - `LucasKanade.py`
         - `LucasKanadeAffine.py`
         - `InverseCompositionAffine.py`
         - `test_lk.py`
         - `test_lk_affine.py`
         - `test_ic_affine.py`
         - `file_utils.py`
       - `ec/`
         - `LucasKanade_Robust.py`
         - `LucasKanade_Pyramid.py`

1 Overview

An important aspect of human vision is the ability to track objects. Tracking is integral to our everyday lives. In this assignment, you will be implementing algorithms that will track an object in a video.

- You will first implement the **Lucas-Kanade tracker** [1].
- Then a more efficient tracker, **Matthews-Baker tracker** [1].

For extra credit, we will also look at ways to make tracking more robust, by incorporating illumination invariance and implementing the algorithm in a pyramid fashion.

2 Preliminaries

In this section, we will go through the basics of the Lucas-Kanade tracker and the Matthews-Baker tracker. Table 1 contains a summary of the variables used in the upcoming sections which explain the algorithms. This will come in handy for matching the dimensions when describing the algorithms.

Template

A template describes the object of interest (eg. a car, football) which we wish to track in a video. Traditionally, the tracking algorithm is initialized with a template, which is represented by a bounding box around the object to be tracked in the first frame of the video. For each of the subsequent frames in the video, the tracker will update its estimate of the object in the image. The tracker achieves this by updating its *affine warp*.

Warps

What is a warp? An image transformation or warp $\mathbf{W}$ is a function that acts on pixel coordinates $\mathbf{x} = [u \;\; v]^T$ and maps pixel values from one place to another in an image $\mathbf{x}' = [u' \;\; v']^T$. Simply put, $\mathbf{W}$ maps a pixel with coordinates $\mathbf{x} = [u \;\; v]^T$ to $\mathbf{x}' = [u' \;\; v']^T$. Translation, rotation, and scaling are all examples of warps. We denote the parameters of the warp function $\mathbf{W}$ by $\mathbf{p}$, refer eq 1.

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

Affine Warp

An *affine warp* is a particular kind of warp that can include any combination of translation, scaling, and rotations. An *affine warp* can be represented by 6 parameters $\mathbf{p} = [p_1 \; p_2 \; p_3 \; p_4 \; p_5 \; p_6]^T$. One of the most convenient things about an *affine warp* is that it is linear; its action on a point with coordinates $\mathbf{x} = [u \;\; v]^T$ can be described as a matrix operation by a 3×3 matrix $\mathbf{W}(\mathbf{p})$, refer eq 2 and eq 3,

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

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

Note: 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 $\mathbf{W}(\mathbf{p})$. We will use *affine warp* and *affine transformation* interchangeably.

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 $\mathcal{L}$ wrt $\mathbf{p}$
$\mathbf{H}$	6×6	Pseudo Hessian of $\mathcal{L}$ wrt $\mathbf{p}$

Table 1: Summary of Variables

Lucas-Kanade Tracker

A Lucas Kanade tracker uses a warp function $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to align an image $\mathbf{I}$ to a template $\mathbf{T}$. The optimal parameters $\mathbf{p}^*$ of the warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ are found by minimizing loss $\mathcal{L}$. The loss $\mathcal{L}$ is the pixel-wise sum of square difference between the warped image $\mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ and template $\mathbf{T}$, refer eq 4 and 5.

Note: We denote pixel locations by $\mathbf{x}$, so $\mathbf{I}(\mathbf{x})$ is the pixel value (eg. *rgb*) at location $\mathbf{x}$ in image $\mathbf{I}$. For simplicity, $\mathbf{I}$ and $\mathbf{T}$ are treated as column vectors instead of a matrix (like linearized matrices!). $\mathbf{W}(\mathbf{x}; \mathbf{p})$ is the point obtained by warping $\mathbf{x}$. $\mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ is the pixel value (eg. *rgb*) after warping.

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

$$\mathbf{p}^* = \underset{\mathbf{p}}{\operatorname{argmin}}\, \mathcal{L} \tag{5}$$

In general, this is a difficult non-linear optimization of $\mathcal{L}$ in $\mathbf{p}$ for even simple warps like *affine warps*, but the Lucas-Kanade tracker circumvents this by using a key idea - **forward additive alignment**. The idea assumes we already have a very close estimate $\mathbf{p}_0$ of the correct warp $\mathbf{p}^*$, then we can assume that a small linear change $\Delta\mathbf{p}$ is enough to get the best alignment i.e. $\mathbf{p}^* = \mathbf{p}_0 + \Delta\mathbf{p}$. This is the forward additive form of the warp. The loss $\mathcal{L}$ can then be written as:

$$\mathcal{L} = \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{x}\right) - I\left(\mathbf{W}\left(\mathbf{x}; \mathbf{p}_0 + \Delta\mathbf{p}\right)\right)\right]^2 \tag{6}$$

$$\Delta\mathbf{p}^* = \underset{\Delta\mathbf{p}}{\operatorname{argmin}}\, \mathcal{L} \tag{7}$$

$$\mathbf{p}^* = \mathbf{p}_0 + \Delta\mathbf{p}^* \tag{8}$$

Expanding this to the first order with Taylor Series:

$$\mathcal{L} \approx \sum_{\mathbf{x}} \left[\mathbf{T}(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}_0)) - \nabla \mathbf{I}(\mathbf{x}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}_0} \Delta \mathbf{p} \right]^2 \tag{9}$$

Here the Jacobian of $\mathbf{I}(\mathbf{x})$, $\nabla \mathbf{I}(\mathbf{x}) = \left[\frac{\partial \mathbf{I}(\mathbf{x})}{\partial u} \frac{\partial \mathbf{I}(\mathbf{x})}{\partial v} \right]$, 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 $\Delta \mathbf{p}^* = \underset{\Delta \mathbf{p}}{\operatorname{argmin}} ||A \Delta \mathbf{p} - b||^2$, note we pull out a negative sign thanks to the squaring,

$$\Delta \mathbf{p}^* = \underset{\Delta \mathbf{p}}{\operatorname{argmin}} \sum_{\mathbf{x}} \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}_0} \Delta \mathbf{p} - \{ \mathbf{T}(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}_0)) \} \right]^2 \tag{10}$$

This can be solved with $\Delta \mathbf{p}^* = (A^T A)^{-1} A^T b$ where $A^T A$ is the Hessian $\mathbf{H}$:

$$(A^T A) = \mathbf{H} = \sum_{\mathbf{x}} \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}_0} \right]^T \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}_0} \right] \tag{11}$$

$$A = \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}_0} \right] \tag{12}$$

$$b = \mathbf{T}(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}_0)) \tag{13}$$

Once $\Delta \mathbf{p}^*$ is computed, the best estimate warp $\mathbf{p}^*$ can be estimated using eq 8. However, in practice, our initial estimate $\mathbf{p}_0$ can be well off the optimal $\mathbf{p}^*$, thus violating the **forward additive alignment** assumption. As a fix, we perform the optimization $\mathcal{L}$ in an iterative fashion using an error threshold ϵ as described in algorithm 2 from [1].

Algorithm 1 The Lucas-Kanade Algorithm

(0) $\mathbf{p} \leftarrow \mathbf{p}_0$

(1) Iterate until $||\Delta \mathbf{p}|| \leq \epsilon$:

(2) Warp $\mathbf{I}$ with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $\mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}))$

(3) Compute the error image $\mathbf{E}(\mathbf{x}) = \mathbf{T}(\mathbf{x}) - \mathbf{I}(\mathbf{W}(\mathbf{x}; \mathbf{p}))$

(4) Warp the gradient $\nabla \mathbf{I}$ with $\mathbf{W}(\mathbf{x}; \mathbf{p})$

(5) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{p})$

(6) Compute the steepest descent image $\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

(7) Compute the Hessian matrix $\mathbf{H} = \sum_{\mathbf{x}} \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$

(8) Compute $\Delta \mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \mathbf{E}(\mathbf{x})$

(9) Update the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta \mathbf{p}$

Matthews-Baker Tracker

While the Lucas-Kanade tracker works very well, it is computationally expensive due to the computation of the Hessian and Jacobian for each frame of the video. The Matthews-Baker tracker is similar to the Lucas-Kanade tracker, but requires less computation, as the Hessian and Jacobian are only computed once for the video. The Matthews-Baker tracker replaces **forward additive alignment** idea of the Lucas-Kanade tracker by the **inverse compositional alignment**. It assumes that the warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ is an invertible function. Since *affine warps* used by the Lucas-Kanade tracker are indeed invertible (why?, think!), we can use the Matthews-Baker tracker instead of a Lucas-Kanade tracker.

The key to efficiency is switching the role of the image $\mathbf{I}$ and the template $\mathbf{T}$ in the algorithm. In contrast to the Lucas-Kanade tracker (warp image $\mathbf{I}$ to template T), the Matthews-Baker tracker warps the template $\mathbf{T}$ to the image $\mathbf{I}$. This key difference leads to the computational gains because unlike the image $\mathbf{I}$ which changes with each frame of the video, the template $\mathbf{T}$ remains fixed throughout the video. If we can write the Hessian and Jacobian involved in the optimization of $\mathcal{L}$ in terms of the template T instead of image I, then, we only need to compute them once at the start of the tracking.

Here is the inverse compositional form of the Matthew-Baker tracker given without the proof of equivalence to the forward additive form of the Lucas-Kanade tracker. Please refer [1] for the proof. Given an initial estimate $\mathbf{p}_0$, we want to find the $\Delta\mathbf{p}^*$ to minimize $\mathcal{L}$ as follows,

$$\mathcal{L} = \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{W}\left(\mathbf{x};\Delta\mathbf{p}\right)\right) - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}_0\right)\right)\right]^2 \tag{14}$$

$$\Delta\mathbf{p}^* = \underset{\Delta\mathbf{p}}{\operatorname{argmin}}\, \mathcal{L} \tag{15}$$

This completes our role switch of the template $\mathbf{T}$ and the image $\mathbf{I}$, please note that $\Delta\mathbf{p}$ are the parameters of the warp $\mathbf{W}$ applied to $\mathbf{T}$ and $\mathbf{p}_0$ are the parameters of the warp $\mathbf{W}$ applied to $\mathbf{I}$.

Another key difference of Matthews-Baker tracker to the Lucas-Kanade tracker is - how do we combine the two warps $\mathbf{W}(\mathbf{x};\mathbf{p}_0)$ and $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$? In the Lucas-Kanade tracker we combined the two warps by simply adding parameter $\mathbf{p}_0$ to another parameter $\Delta\mathbf{p}$, and produce a new warp $\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})$. Specifically, $\mathbf{p}^* = \mathbf{p}_0 + \Delta\mathbf{p}^*$, $\mathbf{W}(\mathbf{x};\mathbf{p}^*) = \mathbf{W}(\mathbf{x};\mathbf{p}_0) + \mathbf{W}(\mathbf{x};\Delta\mathbf{p}^*)$. In Matthews-Baker tracker, we use another way of combination through composition as follows,

$$\mathbf{W}(\mathbf{x};\mathbf{p}^*) = \mathbf{W}(\mathbf{x};\mathbf{p}_0) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}^*)^{-1} \tag{16}$$

If we are using *affine warps* then the warps can be implemented as matrix multiplication, then we can simplify the eq 16 as follows

$$\mathbf{W}(\mathbf{x};\mathbf{p}^*) = \mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p}^*)^{-1};\mathbf{p}_0)$$
$$\mathbf{W}(\mathbf{x};\mathbf{p}^*) = \mathbf{W}(\mathbf{p}_0)\mathbf{W}(\Delta\mathbf{p}^*)^{-1}\mathbf{x}$$

Let us simplify $\mathcal{L}$ as before using first order linear approximation of $\mathbf{T}\left(\mathbf{W}\left(\mathbf{x};\Delta\mathbf{p}\right)\right)$. Please note, for tracking a template $\mathbf{T}$, the summation over $\mathbf{x}$ is performed only over the pixels lying inside $\mathbf{T}$.

$$\mathcal{L} \approx \sum_{\mathbf{x}} \left[\mathbf{T}\left(\mathbf{x}\right) + \nabla\mathbf{T}(\mathbf{x})\frac{\partial\mathbf{W}}{\partial\mathbf{p}_0}\Delta\mathbf{p} - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x};\mathbf{p}_0\right)\right)\right]^2 \tag{17}$$

where $\nabla\mathbf{T}(\mathbf{x}) = \left[\frac{\partial\mathbf{T}(\mathbf{x})}{\partial u}\frac{\partial\mathbf{T}(\mathbf{x})}{\partial v}\right]$.

We now proceed to find a closed form solution to $\Delta\mathbf{p}^*$ by equating the derivative $\frac{\partial\mathcal{L}}{\partial\Delta\mathbf{p}}$ to zero.

$$\frac{\partial\mathcal{L}}{\partial\Delta\mathbf{p}} = 2\sum_{\mathbf{x}}\left[\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}_0}\right]^T\left[\mathbf{T}(\mathbf{x}) + \nabla\mathbf{T}(\mathbf{x})\frac{\partial\mathbf{W}}{\partial\mathbf{p}_0}\Delta\mathbf{p} - \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}_0))\right] \tag{18}$$

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

$$\Delta\mathbf{p} = \mathbf{H}^{-1}\mathbf{J}^T[\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}_0)) - \mathbf{T}] \tag{19}$$

where $\mathbf{J}$ is the Jacobian of $\mathbf{T}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p}))$, $\mathbf{J} = \nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}_0}$, $\mathbf{H}$ is the approximated Hessian $\mathbf{H} = \mathbf{J}^T\mathbf{J}$ and $\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}_0))$ is the warped image. Note that for a given template, the Jacobian $\mathbf{J}$ and Hessian $\mathbf{H}$ are independent of $\mathbf{p}_0$. 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 $\mathbf{p}_0$ to get the new warp parameters for the next iteration.

$$\mathbf{W}(\mathbf{x};\mathbf{p}_0) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}_0) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}^*)^{-1} \tag{20}$$

The next iteration solves Equation 19 starting with the new value of $\mathbf{p}_0$. Possible termination criteria include the absolute value of $\Delta\mathbf{p}^*$ falling below some value or running for some fixed number of iterations.

Algorithm 2 The Matthews-Baker Algorithm

(0) $\mathbf{p} \leftarrow \mathbf{p}_0$

(1) Pre-compute

(1) Evaluate the gradient of $\nabla\mathbf{T}$ of the template $\mathbf{T}(\mathbf{x})$

(2) Evaluate the Jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ at $(\mathbf{x};\mathbf{0})$

(3) Compute the steepest descent images $\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$

(4) Compute the Hessian matrix using $\mathbf{J} = \nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}, \mathbf{H} = \mathbf{J}^T\mathbf{J}$

(5)

(6) Iterate until $||\Delta\mathbf{p}|| \leq \epsilon$:

(7) Warp $\mathbf{I}$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p}))$

(8) Compute the error image $\mathbf{E}(\mathbf{x}) = \mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p})) - \mathbf{T}(\mathbf{x})$

(9) Compute $\Delta\mathbf{p} = \mathbf{H}^{-1}\sum_{\mathbf{x}}\left[\nabla\mathbf{I}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^T\mathbf{E}(\mathbf{x})$

(10) Update the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

3 Theory Questions (30 points)

Each question should only take a couple of lines. If you are lost in many lines of complicated algebra you are doing something wrong.

Q1.1: Calculating the Jacobian (10 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 Lucas Kanade (10 points)

- Find the computational complexity (Big O notation) for each runtime iteration (computing $\mathbf{J}$ and $\mathbf{H}^{-1}$) of the Lucas Kanade 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.

Q1.3: Computational complexity Matthews-Baker (10 points)

- Find the computational complexity (Big O notation) for the initialization step (Precomputing $\mathbf{J}$ and $\mathbf{H}^{-1}$) and for each runtime iteration (Equation 19) of the Matthews-Baker 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?

4 Implementing the Trackers (80 points)

In this section, you will implement the Lucas-Kanade tracker and the Matthews-Baker tracker. Lucas-Kanade has two versions.

- with the warp $\mathbf{W}$ being translation only
- with the warp $\mathbf{W}$ being full affine transformation

.
Matthews-Baker has one version.

- with the warp $\mathbf{W}$ being full affine transformation

.
You will run all three algorithms (versions) on provided videos which can be downloaded here https://www.dropbox.com/sh/l2ip26mkgf5p3e6/AACN2STT5Sk9r6bPeEXIYKZCa?dl=0 (the files are quite big and cannot be uploaded to Piazza). To that end, we have provided script files `test_lk.py`, `test_lk_affine.py` and `test_ic_affine.py` to run three algorithms which can handle reading in images, template region marking, making tracker function calls, displaying output onto the screen and saving results to the disk. As a result, you only need to implement the actual algorithms. 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.

Please include results of three algorithms on all three videos we have provided in your writeup. Specifically, please include results on five frames with an reasonable interval (e.g., for landing.mp4, race.mp4, and ballet.mp4 with only about 50 frames in total, you can use an interval of 10 frames and for car1.mp4 with 260 frames, you can use an interval of 50 frames) in your writeup. Also, please describe the difference between three algorithms in terms of performance (e.g., accuracy and speed) and explain why.

Q2.1: Lucas-Kanade Forward Additive Alignment with Translation (20 points)

Write the function with the following function signature:

```
dx, dy = LucasKanade(It, It1, rect)
```

that computes the optimal local motion represented by only translation (motion in x and y directions) from frame $\mathbf{I}_t$ to frame $\mathbf{I}_{t+1}$ that minimizes Equation 4. 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 `[x1, y1, x2, y2]`, where `(x1, y1)` is the top-left corner and `(x2, y2)` is the bottom-right corner of the bounding box. To deal with fractional movement of the template, you will need to interpolate the image using the function `RectBivariateSpline` from scipy.interpolate. Also, the `RectBivariateSpline.ev` function can be used to compute the gradient of an image at a point location. You will also need to iterate the estimation in Equation 10 until the change in warp parameters `(dx, dy)` is below a threshold or the number of iterations is too large. To solve the least square problem in Equation 10, you can use the function `lstsq` from np.linalg.

Writeup: Please include indented code of the function `LucasKanade()`. The *verbatim* command in latex may be helpful.

Q2.2: Lucas-Kanade Forward Additive Alignment with Affine Transformation (20 points)

Write the function with the following function signature:

```
M = LucasKanadeAffine(It, It1, rect)
```

Different from Q2.1, here we compute the optimal local motion M represented by an affine transformation with 6 parameters. In other words, the output M should be a 2×3 affine transformation matrix.

Writeup: Please include indented code of the function `LucasKanadeAffine()`.

Q2.3: Matthew-Bakers Inverse Compositional Alignment with Affine (20 points)

```
M = InverseCompositionAffine(It, It1, rect)
```

Same with Q2.2, the function should output a 2×3 affine matrix so that it aligns the current frame with the template. But you need to implement the inverse compositional alignment following the equations in the preliminaries or the slides from class.

Writeup: Please include indented code of the function `InverseCompositionAffine()`.

Q2.4: Testing Your Algorithms (20 points)

Test your three algorithms on the video sequences (`car1.npy`), (`car2.npy`), (`landing.npy`), (`race.npy`) and (`ballet.npy`), using our provided scripts. How do your algorithms perform on each video? Which are the differences of three algorithms in terms of performance and why do we have those differences? At what point does the algorithm break down and why does this happen? Remember that the algorithms you are implementing are very basic tracking algorithms so that it might not work at some situations. As long as it is working reasonably (not necessarily perfect), you will receive full credits.

Writeup: Submit your results of three algorithms on all five videos (as mentioned above, please include results on frames with a reasonable interval), analyze your results and answer above questions.

5 Extra Credit (20 points)

Q3.1x: Adding Illumination Robustness (10 points)

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. The first 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 brightness of pixels in the template. The second is to use a more robust M-estimator instead of least squares (so, e.g. a Huber or a Tukey M-estimator) that does not let outliers adversely affect the cost function evaluation. Note that doing this will modify our least squares problem to a weighted least squares problem, i.e. for each residual term you will have a corresponding weight $\mathbf{\Lambda}_{ii}$ and your minimization function will look like

$$L = \sum_{\mathbf{x}} \mathbf{\Lambda}_{ii} \left[\mathbf{T}\left(\mathbf{W}\left(\mathbf{x}; \Delta\mathbf{p}\right)\right) - \mathbf{I}\left(\mathbf{W}\left(\mathbf{x}; \mathbf{p}\right)\right)\right]^2 \tag{21}$$

leading your jacobian computation to be a weighted jacobian instead of what you have seen earlier (Eq. 11)

$$A^T \mathbf{\Lambda} A \Delta\mathbf{p} = A^T \mathbf{\Lambda} b \tag{22}$$

$$\Rightarrow \Delta\mathbf{p} = \left(A^T \mathbf{\Lambda} A\right)^{-1} A^T \mathbf{\Lambda} b \tag{23}$$

Here $\mathbf{\Lambda}$ is a diagonal matrix of weights corresponding to the residual term computed as per the choice of the robust M-estimator used, A is the jacobian matrix for each pixel of the template considered in the cost function, and b is the corresponding vector of residuals. Implement these two methods and test your new tracker on the provided video sequences again.

Implement these modifications for the Lucas-Kanade tracker that you implemented for Q2.1. Use the same function names with '`Robust`' appended.

Writeup: Please include the relevant code and submit the results of your '`Robust`' algorithm on any one of the video, compare with the results from the Lucas-Kanade tracker in Q2.1 and explain any difference.

Q3.2x: LK Tracking on an Image Pyramid (10 points)

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 a set of image pyramids instead of a single image. The Pyramid tracker starts by performing tracking on a higher level (smaller image) to get a course alignment estimate, propagating 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 due to its coarse to fine approach.

Implement these modifications for the Lucas-Kanade tracker that you implemented for Q2.1. Use the same function names with '`Pyramid`' appended.

Writeup: Please include the relevant code and submit the results of your '`Pyramid`' algorithm on any one of the video, compare with the results from the Lucas-Kanade tracker in Q2.1 and explain any difference.

6 Submission Summary

- **Q1.1** Derive the expression for the Jacobian Matrix
- **Q1.2** What is the computational complexity of Lucas-Kanade method?
- **Q1.3** What is the computational complexity of Matthews-Baker method?
- **Q2.1** Write the Lucas-Kanade tracker with translation
- **Q2.2** Write the Lucas-Kanade tracker with affine transformation
- **Q2.3** Write the Matthews-Baker tracker with affine transformation
- **Q2.4** Test three algorithms on the provided sequences and analyze the results.
- **Q3.1x** Add illumination robustness
- **Q3.2x** LK Tracking on an image pyramid.

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