16-720 Computer Vision: Homework 3
Lucas-Kanade Tracking & Background Subtraction

Instructor: Deva Ramanan
TAs: Achal Dave, Sashank Jujjavarapu
Siddarth Malreddy, Brian Pugh

See course website for deadline: `http://16720.courses.cs.cmu.edu/`

- Please pack your system and write-up into a single file `<andrewid>.zip`, in accordance with the complete submission checklist at the end of this document.
- All tasks marked with a **Q** require a submission.
- Please stick to the provided function signatures, variable names, and file names.
- **Start early**! This homework cannot be completed within two hours!
- **Verify your implementation as you proceed**: otherwise you will risk having a huge mess of malfunctioning code that can go wrong anywhere.
- If you have any questions, please contact the TAs - Achal Dave (`achald@cmu.edu`), Sashank (`bjujjava@andrew.cmu.edu`), Siddarth (`smalredd@andrew.cmu.edu`), Brian (`bpugh@andrew.cmu.edu`).

\- \- \-

This homework consists of three sections. In the first section you will be implementing the Inverse Compositional Lucas-Kanade (LK) tracker with one single template. In the second section, the tracker will be generalized to accommodate for large appearance variance. The last section requires you to implement a motion subtraction method for tracking moving pixels in a scene. Note that all three sections are based on the Lucas-Kanade tracking framework; besides the course slide decks, the following reference may also be extremely helpful:

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

Both are available at:
`http://www.ri.cmu.edu/research_project_detail.html?type=publication&project_id=515`.

1 Lucas-Kanade Tracking

In this section you will be implementing the Inverse Compositional Lucas-Kanade tracker with one single template. In the scenario of two-dimensional tracking with pure translation, the problem can be described as follows: starting with a rectangle R_t on frame I_t, the

Lucas-Kanade tracker aims to move it by an offset $(\Delta u, \Delta v)$ to obtain another rectangle R_{t+1} on frame I_{t+1}, so that the pixel squared difference in the two rectangles is minimized:

$$\min_{u,v} J(u,v) = \sum_{(x,y)\in R_t} (I_{t+1}(x+u, y+v) - I_t(x,y))^2 \tag{1}$$

Q1.1 (5 points) Starting with an initial guess of (u, v) (for instance, $(0, 0)$), we can compute the optimal (u^*, v^*) iteratively. In each iteration, the objective function is locally linearized by first-order Taylor expansion and optimized by solving a linear system that has the form $A\Delta p = b$, where $\Delta p = (u, v)^T$, the template offset.

- What is $A^T A$?
- What conditions must $A^T A$ meet so that the template offset can be calculated reliably?

Q1.2 (15 points) Implement a function with the following signature

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

that computes the optimal local motion from frame I_t to frame I_{t+1} that minimizes Equation 1. Here `It` is the image frame I_t, `It1` is the image frame I_{t+1}, and `rect` is the 4-by-1 vector that represents a rectangle on the image frame I_t. The four components of the rectangle are $[x1, y1, x2, y2]$, where $(x1, y1)$ is the top-left corner and (x_2, y_2) is the bottom-right corner. 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 (u, v) is below a threshold. You have to implement the *inverse compositional* version of the Lucas-Kanade tracker (Section 2.2 in [2]). You are encouraged (but not required) to implement the original Lucas-Kanade algorithm (Section 2.1 in [2]). Implementing the original L-K algorithm will help you appreciate the performance improvement of the inverse compositional algorithm.

Q1.3 (10 points) Write a script `testCarSequence.m` that loads the video frames from `carseq.mat`, and runs the Lucas-Kanade tracker that you have implemented in the previous task to track the car. `carseq.mat` can be located in the `data` directory and it contains one single three-dimensional matrix: the first two dimensions correspond to the *height* and *width* of the frames respectively, and the third dimension contain the indices of the frames (that is, the first frame can be visualized with `imshow(frames(:, :, 1))`). The rectangle in the first frame is $[x1, y1, x2, y2] = [60, 117, 146, 152]$. Report your tracking performance (image + bounding rectangle) at frames 2, 100, 200, 300 and 400 in a format similar to Figure 1. Also, create a file called `carseqrects.mat`, which contains one single $n \times 4$ matrix `rects`, where each row stores the `rect` that you have obtained for each frame, and n is the total number of frames.

Now, lets apply the your algorithm on a challenging problem: tracking a beating vessel in an ultrasound volume. Unlike regular scenes in the world, medical images acquired by an ultrasound transducer undergo significant non-rigid motion. Write a new script

Figure 1: Lucas-Kanade Tracking with One Single Template for the car sequence

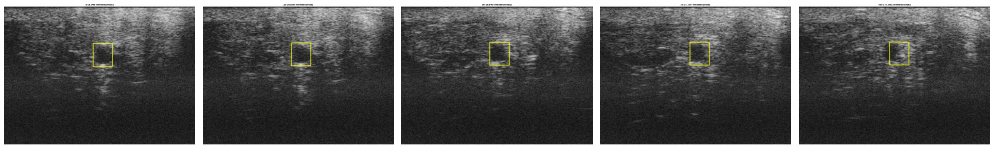

Figure 2: Lucas-Kanade Tracking with One Single Template for the ultrasound sequence

`testUltrasoundSequence.m` that loads the video frames from `usseq.mat`, and tracks the beating vessel using the Lucas-Kanade tracker that you have implemented in the previous question. The rectangle in the first frame is $[x1, y1, x2, y2] = [255, 105, 310, 170]$. Report your tracking performance (image + bounding rectangle) at frames 5, 25, 50, 75 and 100 in a format similar to Figure 2. Also, create a file called `usseqrects.mat`, which contains one single $n \times 4$ matrix `rects`, where each row stores the `rect` that you have obtained for each frame, and n is the total number of frames.

Q1.4 (Extra credit, 20 points) As you might have noticed, the image content we are tracking in the first frame differs from the one in the last frame. This tracker lags and loses some tracking capability. This is understandable since we are updating the template after processing each frame and the error can be accumulating. This problem is known as *template drifting*. There are several ways to mitigate this problem. Iain Matthews et al. (2003, `https://www.ri.cmu.edu/publication_view.html?pub_id=4433`) suggested one possible approach. Write two scripts `testCarSequenceWithTemplateCorrection.m` and `testUSSequenceWithTemplateCorrection.m` with a similar functionality to Q1.3, but with a template correction routine incorporated. Save the resulting rects as `carseqrects-wcrt.mat` and `usseqrects-wcrt.mat` respectively, and also report the performance at the same frames described above. An example is given in Figure 3 and Figure 4.

Figure 3: Lucas-Kanade Tracking with Template Correction for the car sequence

In the above figures, the green rectangles are created with the baseline tracker in **Q1.3**, the yellow ones with the tracker in **Q1.4**. The tracking performance has been improved non-trivially. Note that you do not necessarily have to draw two rectangles in each frame,

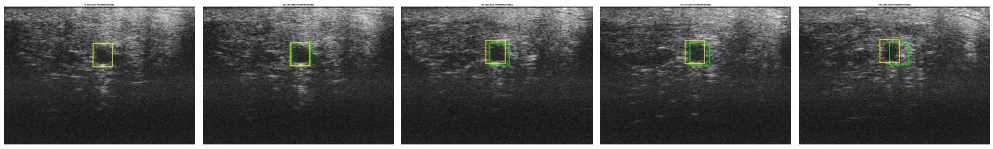

Figure 4: Lucas-Kanade Tracking with Template Correction for the ultra sound sequence

but make sure that the performance improvement can be easily visually inspected.

2 Lucas-Kanade Tracking with Appearance Basis

The tracker we have implemented in the first section, with or without template drifting correction, may suffice if the object being tracked is not subject to drastic appearance variance. However, in real life, this can hardly be the case. In this section, you will implement a variant of the Lucas-Kanade tracker (see section 3.4 in [2]), to model linear appearance variation in the tracking. We have prepared another sequence `sylvseq.mat` (the initial rectangle is $[102, 62, 156, 108]$), with exactly the same format as `carseq.mat`, on which you can test the implementation.

2.1 Appearance Basis

One way to address this issue is to use eigen-space approach (aka, principal component analysis, or PCA). The idea is to analyze the historic data we have collected on the object, and produce a few bases, whose linear combination would most likely to constitute the appearance of the object in the new frame. This is actually similar to the idea of having a lot of templates, but looking for too many templates may be expensive, so we only worry about the *principal* templates.

Mathematically, suppose we are given a set of k image bases $\{B_c\}_{c=1}^k$ of the same size. We can approximate the appearance variation of the new frame I_{t+1} as a linear combination of the previous frame I_t and the bases weighted by $\vec{w} = [w_1, \ldots, w_k]^T$, such that

$$I_{t+1} = I_t + \sum_{c=1}^{k} w_c B_c \tag{2}$$

Q2.1 (5 points) Express w_c for $c = 1, 2, \ldots, k$, as a function of I_{t+1}, I_t, and $\{B_c\}_{c=1}^k$, given Equation 2. Note that since the B_c's are bases, they are orthogonal to each other.

2.2 Tracking

Given k bases, $\{B_c\}_{c=1}^k$, our goal is then to simultaneously find the translation (u, v) and the weights $\vec{w}$ that minimizes the following objective function:

$$\min_{u,v,\vec{w}} = \sum_{(x,y)\in R_t} (I_{t+1}(x+u, y+v) - I_t(x,y) - [\sum_c w_c B_c](x,y))^2 \tag{3}$$

where we define $\sum_c w_c B_c(x, y)$ to be the value of the combination of bases at the position associated with the pixel location (x, y).

Q2.2 (15 points) Implement a function with the following signature

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

where `bases` is a three-dimensional matrix that contains the bases. It has the same format as `frames` as is described earlier and can be found in `sylvbases.mat`.

Q2.3 (15 points) Write a script `testSylvSequence.m` that loads the video frames from `sylvseq.mat` and runs the new Lucas-Kanade tracker to track the sylv (the toy). The bases are available in `sylvbases.mat` in the `data` directory. The rectangle in the first frame is $[x1, y1, x2, y2] = [102, 62, 156, 108]$. Please report the performance of this tracker at frames 2, 200, 300, 350 and 400 (the frame + bounding box) in a format similar to Figure 5. Also, create a `sylvseqrects.mat` for all the `rects` you have obtained for each frame. It should contain one single $n \times 4$ matrix named `rects`, where n is the number of frames, and each row contains $[x1, y1, x2, y2]$, where $(x1, y1)$ is the coordinate of the top-left corner of the tracking box, and $(x2, y2)$ the bottom-right corner.

Figure 5: Lucas-Kanade Tracking with Appearance Basis

3 Affine Motion Subtraction

In this section, you will implement a tracker for estimating dominant affine motion in a sequence of images, and subsequently identify pixels corresponding to moving objects in the scene. You will be using the images in the file `aerialseq.mat`, which consists aerial views of moving vehicles from a non-stationary camera.

3.1 Dominant Motion Estimation

You will start by implementing a tracker for affine motion using the equations for affine flow. Essentially in the first section we assume that the motion is limited to pure translation, but in this scenario it has been relaxed to any affine motion. To estimate dominant motion, the entire image I_t will serve as the template to be tracked in image I_{t+1}, that is, I_{t+1} is assumed to be approximately an affine warped version of I_t. This approach is reasonable under the assumption that a majority of the pixels correspond to the stationary objects in the scene whose depth variation is small relative to their distance from the camera.

Using the equations for the affine model of flow, you can recover the vector $\Delta p = [a, b, c, d, e, f]^T$ of affine flow parameters. They will be related to the equivalent affine trans-

formation matrix as:

$$M = \begin{bmatrix} 1+a & b & c \\ d & 1+e & f \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The homogenous image coordinates of I_t to I_{t+1} can be related with $\vec{x}_{t+1} = M\vec{x}_t$, where $\vec{x} = [x, y, 1]^T$. For the next pair of temporally adjacent images in the sequence, image I_{t+1} will serve as the template to be tracked in image I_{t+2}, and so on through the rest of the sequence. Note that M will differ between successive image pairs. As before, each update of the affine parameter vector, Δp is computed via a least-squares method using pesudo-inverse as described in the class.

Note that unlike previous exmaples where the template to be tracked is usually small in comparison with the size of the image, image I_t will almost always not be contained fully in the warped version I_{t+1}. Hence the matrix of image derivatives, A, and the temporal derivatives, $\partial_t I_t$, must be computed only on the pixels lying in the region common to I_t and the warped versoin of I_{t+1} to be meaningful.

Q3.1 (15 points) Write a function with the following signature

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

where `M` is the affine transformation matrix, and `It` and `It1` are I_t and I_{t+1} respectively. `LucasKanadeAffine` should be relatively similar to `LucasKanade` from the first section.

3.2 Moving Object Detection

Once you are able to compute the transformation matrix M relating an image pair I_t and I_{t+1}, a naive way for determining pixels lying on moving objects is as follows: warp the image I_t using M so that it is registered to I_{t+1} and subtract it from I_{t+1}; the locations where the absolute difference exceeds a threshold can then be declared as corresponding to locations of moving objects. To obtain better results, you can check out the following MATLAB functions: `bwselect`, `bwareaopen`, `imdilate`, and `imerode`.

Q3.2 (10 points) Using the function you have developed for dominant motion estimation, write a function with the following signature

```
mask = SubtractDominantMotion(image1, image2)
```

where `image1` and `image2` form the input image pair, and `mask` is a binary image of the same size that dictates which pixels are considered to be corresponding to moving objects. You should invoke `LucasKanadeAffine` in this function to derive the transformation matrix M, and produce the aforementioned binary mask accordingly.

Q3.3 (10 points) Write a script `testAerialSequence.m` that loads the image sequence from `aerialseq.mat` and run the motion detection routine you have developed to detect the moving objects. Report the performance at frames 30, 60, 90 and 120 with the corresponding binary masks superimposed, as exemplified in Figure 6. Feel free to visualize the motion detection performance in a way that you would prefer, but please make sure it can be visually inspected without undue effort. The MATLAB function `imfuse` may be useful.

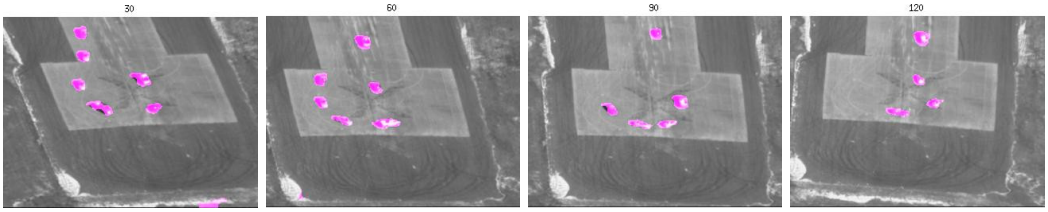

Figure 6: Lucas-Kanade Tracking of affine motion

4 Deliverables

Upload one single file, `andrewid.zip` to Blackboard, which, when uncompressed, produces one folder `<andrewid>` containing two folders 'code', and 'results'.

Your written report:

- `<andrewid>.pdf`: your writeup for this homework should be submitted to Gradescope.

The following should be present in the code and results folders:

- `code/LucasKanadeInverseCompositional.m`
- `code/LucasKanadeBasis.m`
- `code/LucasKanadeAffine.m`
- `code/SubtractDominantMotion.m`
- `code/testCarSequence.m`
- `code/testUltrasoundSequence.m`
- `code/testSylvSequence.m`
- `code/testAerialSequence.m`
- `code/testCarSequenceWithTemplateCorrection.m` (optional, extra credit)
- `code/testUSSequenceWithTemplateCorrection.m` (optional, extra credit)
- `results/carseqrects.mat`
- `results/usseqrects.mat`
- `results/sylvseqrects.mat`
- `results/carseqrects-wcrt.mat` (optional, extra credit)
- `results/usseqrects-wcrt.mat` (optional, extra credit)

DO NOT include the `data` directory in your submission.