16-720: Assignment 3

Abhinav Maurya
amaurya@andrew.cmu.edu

1.1.1: Nature of $A'A$

When we linearize the optimization objective for Lucas-Kanade alignment (given below), we get loading matrix A which has two columns, one for gradients along X-direction and the second for gradients along Y-direction. Therefore, $A'A$ is a 2x2 Hessian matrix which is computed using the gradients of the image. The $A'\Delta u = b$ linearized equation is as follows:

$$[Dx, Dy] * [\Delta u, \Delta v]' = \Delta I$$

Here ΔI is the difference between consecutive template windows, where the successor template window is at a displacement of u, v with respect to the previous tracking window. Intuitively, we are trying to perform a first-order approximation by equating the change in successive template windows with the product of gradients (rate of change) and the u, v column vector (actual displacement). The optimization is performed iteratively starting with $u, v = 0$.

$$[Dx, Dy] * [\Delta u, \Delta v]' = \Delta I_t$$

$$A\Delta p = \Delta I$$

$$A'A\Delta p = A'\Delta I$$

$$\Delta p = (A'A)\backslash(A'\Delta I)$$

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

1.1.2: Conditions on $A'A$

$A'A$ needs to be well-conditioned so that it is invertible. It implies that none of its eigenvalues can be zero or very close to zero. Eigenvalues of the inverted resulting matrix are reciprocals of the eigenvalues of the matrix being inverted. If any eigenvalue is zero or close to zero, inverting the matrix will lead to infinite or very high eigenvalues of the inverted matrix which can lead to problems in finite-precision calculations. The ratio of the largest eigenvalue to smallest eigenvalue called the conditioning number of a matrix cannot be very small.

1.2: LucasKanade

Included in `code/LucasKanade.m` file.

1.3: testCarSequence

Included in `code/testCarSequence.m` file. Results in figure (1)

1.4: testCarSequenceWithTemplateCorrection

Included in `code/testCarSequenceWithTemplateCorrection.m` file. Results in figure (2)

We used the `code/subtractDominantMotion.m` function written as part of this assignment to improve the tracking. Every 50 frames, we computed the mask using the `code/subtractDominantMotion.m` function, performed a search over altertive rectangles around the current rectangle (+/-5 pixels both horizontally and vertically), and chose an alternate rectangle if it improved a score by at least a certain positove amount (100 in our case). The score of competing rectangles was calculated using the mask image by counting the number of pixels in the mask image corresponding to the rectangle which were detected to contain a fast moving object against a slower moving background. The results were visibly better and the slight optimization allowed the tracked rectangle to refocus on the object. Since the top and windows of the car have the maximum contrast, the tracking rectangle latches onto this aspect and tracks it throughout the video.

2.1: Weight vector for basis functions

We are given that

$$\sum_{c=1..C} w_c B_c = I_{t+1} - I_t$$

Since the basic images B_i are othogonal to each other, we multiply both sides by B'_c if we are trying to determine w_c.

$$w_c B'_c B_c + \sum_{i=1..C:i\neq c} w_i B'_c B_i = B'_c(I_{t+1} - I_t)$$

$$\therefore w_c B'_c B_c = B'_c(I_{t+1} - I_t)$$

$$\therefore w_c = (B'_c B_c) \backslash (B'_c(I_{t+1} - I_t))$$

The weight vector w_c is essentially discovered as a solution to the optimization problem mentioned in the assignment.

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

As we see, it is an extension to the original Lucas-Kanade optimization problem we optimized for in question (1). We naturally extend the original Lucas-Kanade optimization procedure by extending the optimization vector from $[u, v]$ to $[u, v, \boldsymbol{w_c}]$. The basis vectors are linearized into column vectors are appended to the original $A = [Dx, Dy]$ matrix. Thus, $A = [Dx, Dy, B_1, B_2, ..., B_n]$ where $B_1, B_2, ..., B_n$ are the n provided basis functions that can explain the shifted template in the succesor image in addition to the gradients from the predecessor image. Thus, the linearization is transformed as follows:

$$[Dx, Dy, B_1, B2, ..., B_n] * [\Delta u, \Delta v, w_1, w_2, ..., w_n]' = \Delta I_t$$

$$A\Delta p = \Delta I$$

$$A'A\Delta p = A'\Delta I$$

$$\Delta p = (A'A)\backslash(A'\Delta I)$$

While $\Delta u, \Delta v$ are optimized cumulatively, weights w_c are solved afresh every time we incrementally search for a new $\Delta u, \Delta v$. This is because the basis vectors are constants and are not changing with the optimization like Dx, Dy.

2.2: LucasKanadeBasis

Included in `code/LucasKanadeBasis.m` file.

2.3: testSylvSequence

Included in `code/testSylvSequence.m` file. Results in figure (3).

3.1: LucasKanadeAffine

Included in `code/LucasKanadeAffine.m` file.

3.2: SubtractDominantMotion

Included in `code/SubtractDominantMotion.m` file.

3.3: testAerialSequence

Included in `code/testAerialSequence.m` file. Results in figure (4).

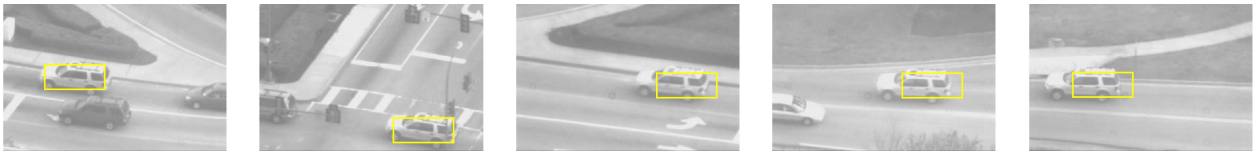

Figure 1: Output of `testCarSequence`

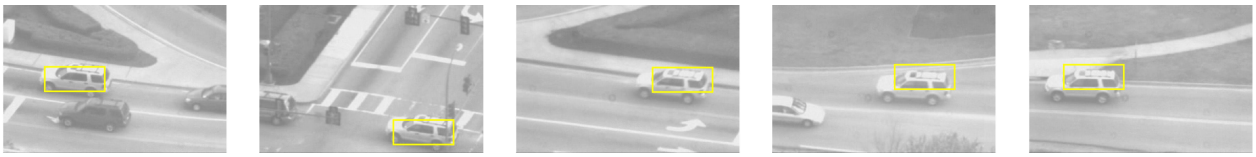

Figure 2: Output of `testCarSequenceWithTemplateCorrection`

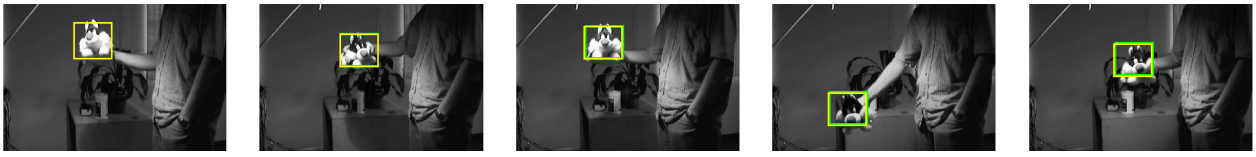

Figure 3: Output of `testSylvSequence`

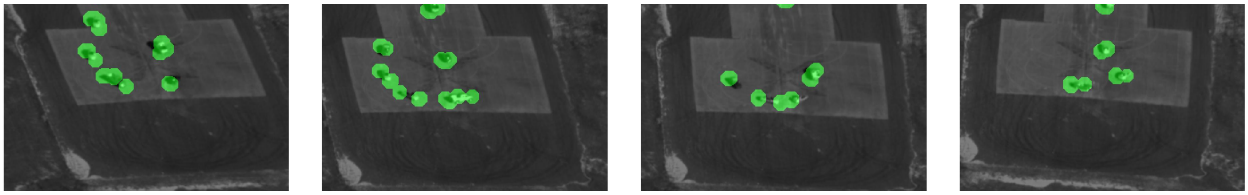

Figure 4: Output of `testAerialSequence`