Homework Assignment 2

Abhay Gupta (Andrew Id: abhayg)

October 24, 2018

1 Question 1

1.1 Derivation and Theory

Our aim is to find $\Delta p = [u, v]^T$ that minimizes the equation:

$$\begin{aligned}
&\sum_{(x,y)\in\mathcal{R}^t} (I_{t+1}(x+u, y+v) - I_t(x,y))^2 \\
&\approx \sum_{(x,y)\in\mathcal{R}^t} (I_{t+1}(x,y) - I_t(x,y) + u(I_{t+1})_x(x,y) + v(I_{t+1})_y(x,y))^2 \\
&= \sum_{(x,y)\in\mathcal{R}^t} \left([(I_{t+1})_x(x,y)\,(I_{t+1})_y(x,y)][uv]^T - (I_t(x,y) - I_{t+1}(x,y)) \right)^2 \\
&= \sum_{(x,y)\in\mathcal{R}^t} (A\Delta p - b)^2
\end{aligned}$$

Taking derivative with repsect to Δp, we get,

$$\begin{aligned}
A^T A\Delta p - A^T b &= 0 \\
p &= (A^T A)^{-1} A^T b
\end{aligned}$$

1. $\frac{\partial \mathcal{W}(x;p)}{\partial p^T} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2. $A = [(I_{t+1})_x(x,y)\,(I_{t+1})_y(x,y)]$, $b = I_t(x,y) - I_{t+1}(x,y)$
3. To ensure that Δp can be found, $A^T A$ must be invertible and the determinant of $A^T A \neq 0$.

1.2 LK Implementation

Code is implemented in `LucasKanade.py`. The threshold for stopping LK is 0.5.

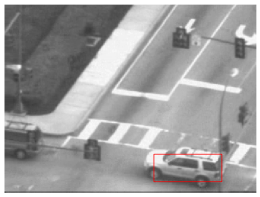
1.3 Tracking with No Correction

The result is shown in Figure 1a—1e and the results stored in `carseqrects.npy`. The threshold for stopping LK with template correction is 0.5.

1.4 Tracking with Template Correction

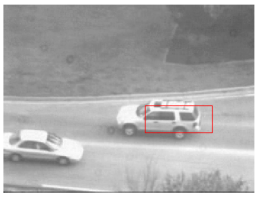
The result is shown in Figures 2a—2e and the results stored in `carseqrects-wcrt.npy`. The black rectangle is the corrected frame and the red rectangle is the original frame from Q1.3. The threshold for stopping LK with template correction is 0.5.

(a) Frame 1



(b) Frame 100

(c) Frame 200



(d) Frame 300

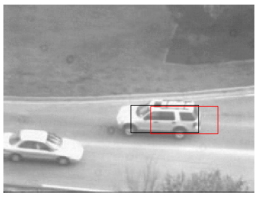
(e) Frame 400

Figure 1: Luckas Kanade with No Template Correction

(a) Frame 1

(b) Frame 100

(c) Frame 200



(d) Frame 300

(e) Frame 400

Figure 2: Luckas Kanade with Template Correction

2 Question 2

2.1 Deriving Weights For Bases

We know that the bases B_k are orthogonal to each other. From the appearance bias equation, we have,

$$
\begin{aligned}
I_{t+1} &= I_t + \sum_{i=1}^{k} w_i B_i \\
I_{t+1} - I_t &= \sum_{i=1}^{k} w_i B_i \\
B_i(I_{t+1} - I_t) &= B_i(\sum_{i=1}^{k} w_i B_i) \\
&= B_i(w_1 B_1 + w_2 B_2 + ... + w_k B_k) \\
&= w_1 B_i B_1 + w_2 B_i B_2 + ... + w_i B_i B_i + ... + w_k B_i B_k \\
&= w_i ||B_i||_2^2 \\
\implies w_i &= \frac{B_i(I_{t+1} - I_t)}{||B_i||^2}
\end{aligned}
$$

Thus for all $c = 1, 2, ..., k$, we get

$$
w_c = \frac{B_c(I_{t+1} - I_t)}{||B_c||^2}
$$

2.2 Tracking - Implementation

Code is implemented as `LucasKanadeBasis.py`. The threshold for stopping LK with template correction is 0.5.

2.3 Tracking - Results

The result is shown in Figure 3a—3e and the results stored in `sylvseqrects.npy`

(a) Frame 1

(b) Frame 200

(c) Frame 300

(d) Frame 350

(e) Frame 400

Figure 3: Luckas Kanade with Appearance Bases

3 Question 3

3.1 Dominant Motion Estimation

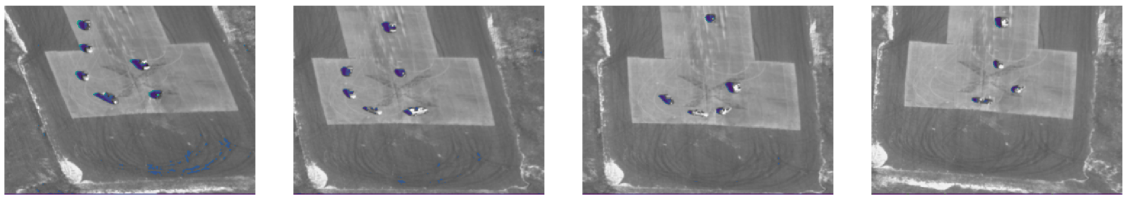
Code is implemented in `LucasKanadeAffine.py`. The threshold for stopping LKAffine is 0.1.

3.2 Moving Object Detection - Implementation

Code is implemented in `SubtractDominantMotion.py`. The threshold for selecting the mask is 0.35.

3.3 Moving Object Detection - Results

The result is shown in Figure 4a—4d. The white patches depict the tracked area.



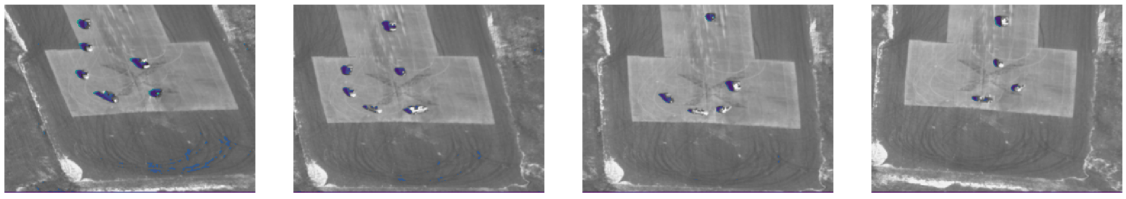
(a) Frame 30 (b) Frame 60 (c) Frame 90 (d) Frame 120

Figure 4: Luckas Kanade Affine with Dominant Motion Subtraction

4 Question 4

4.1 Inverse Composition Affine

Code is implemented in `InverseCompositionAffine.py`. It gives similar performance to `LucasKanadeAffine.py`. When using inverse composition, the H (hessian matrix) is precomputed on the template and remains static over the computations of the image interpolations when we update Δp. The only change here is that there is a possibility that there are pixels that move out of the image frame because of the image warping and for that we have to ensure that we take only the valid pixels corresponding to the warp from the matrix $A = \sum_x \nabla T \frac{\partial \mathcal{W}(x;p)}{\partial p}$ and then recompute the hessian as $H = A_{valid}^T A_{valid}$ and the difference in images as $b = A_{valid}^T(\text{image} - \text{template})$. In constrast, the regular algorithm recomputes $A_{LKOriginal} = \sum_x \nabla I \frac{\partial \mathcal{W}(x;p)}{\partial p}$. Here since $\mathcal{W}(x;p)$ changes the product also changes and hence every iteration, there is a computation that needs to be done. Results for inverse composition are shown below in Figure 5a—5d. The threshold for stopping LK-IC is 0.1 and the mask threshold is 0.35.



(a) Frame 30 (b) Frame 60 (c) Frame 90 (d) Frame 120

Figure 5: Luckas Kanade Inverse Composition Affine with Dominant Motion Subtraction

4.2 Correlation Filters - Derivation

We have to compute

$$\underset{g}{\operatorname{argmin}} \frac{1}{2}||y - X^T g||_2^2 + \frac{\lambda}{2}||g||_2^2$$
$$\underset{g}{\operatorname{argmin}} \left(\frac{1}{2}(y - X^T g)^T (y - Xg) + \frac{\lambda}{2} g^T g \right)$$

We differentiate with respect to g, to get,

$$\begin{aligned}(y - X^T g)(-X) + \lambda g &= 0 \\ -Xy + XX^T g + \lambda g &= 0 \\ (XX^T + \lambda I)g &= Xy \\ g &= (XX^T + \lambda I)^{-1} Xy\end{aligned}$$

The filters visualized for $\lambda = 0, 1$ are given in Figure 6

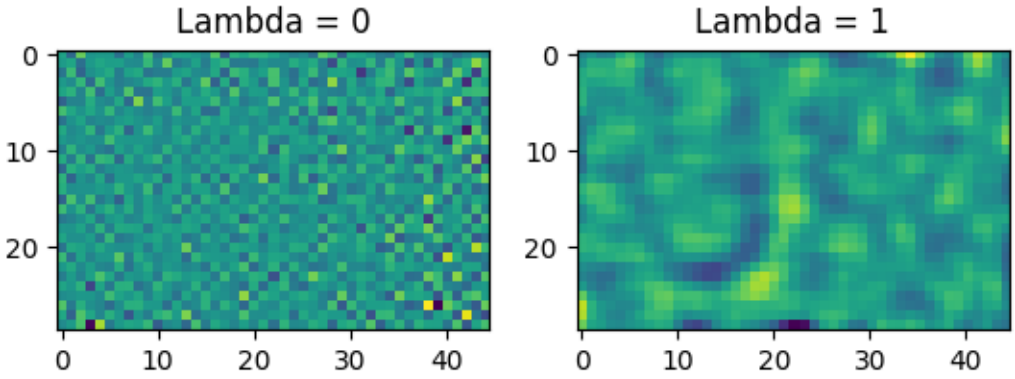


Figure 6: Different Filters for $\lambda = 0$ and 1

$\lambda = 1$ performs better than $\lambda = 0$, because the resulting filter acts like a blurring filter for the image, and removes noisy artifacts from the image (low-pass filter) and thus when it is correlated with the actual image, it can detect the actual location of the template (with brighter white point on image after correlation - seen in Figure 7b) very easily.

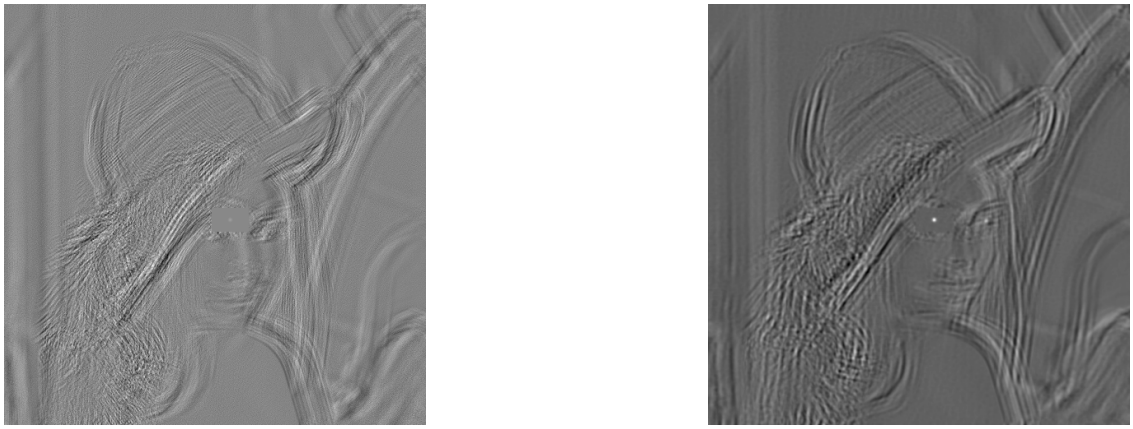
4.3 Correlation Filters - Correlation Visualization

Performing correlation on the images, we get, Figures 7a—7b.

4.4 Correlation Filters - Convolution Visualization

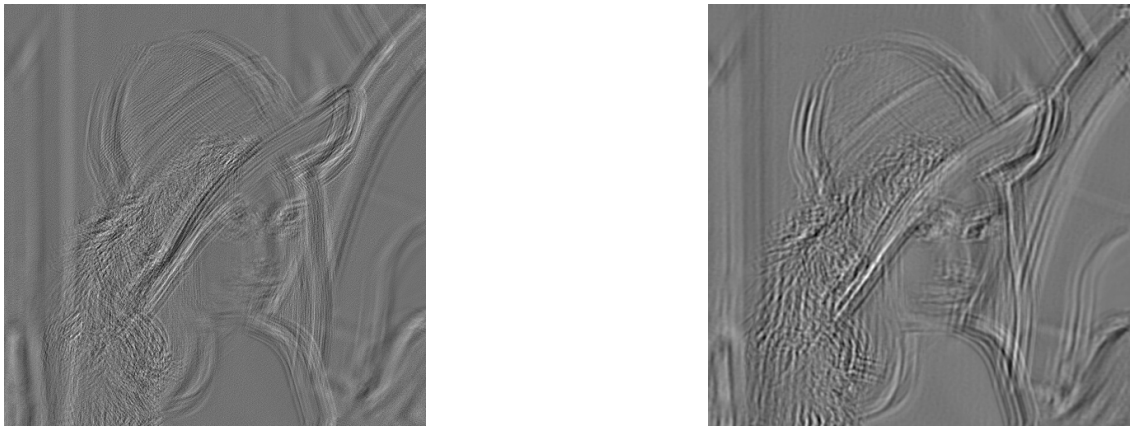
Performing convolution on the images, we get, Figures 8a—8b.

The difference between convolution and correlation is that convolution takes the filter, flips it left to right and up to down and then applies it to the image. For example, the filter $g = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ becomes $gd = \begin{pmatrix} d & c \\ b & a \end{pmatrix}$ for convolution. That is we can do `gd = np.flipur(np.flipud(g))` and then apply correlation on the image with the filter `gd` to get the convolution output.



(a) $\lambda = 0$ (b) $\lambda = 1$

Figure 7: Correlation with Filters



(a) $\lambda = 0$ (b) $\lambda = 1$

Figure 8: Convolution with Filters