16-720B Computer Vision: Homework 3

Lucas-Kanade Tracking and Correlation Filters

Due: October 24 at 11:59pm

[redacted] (AndrewID: [redacted])

1 Lucas-Kanade Tracking

Q1.1:

- What is $\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}$?

$$\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T} = \frac{\partial}{\partial \mathbf{p}^T}(\mathbf{x}+\mathbf{p}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- What is $\mathbf{A}$ and $\mathbf{b}$?

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}_1^{'})}{\partial \mathbf{x'}_1^T} & \cdots & \mathbf{0}^T \\ \vdots & \ddots & \vdots \\ \mathbf{0}^T & \cdots & \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}_N^{'})}{\partial \mathbf{x'}_N^T} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{W}(\mathbf{x}_1;\mathbf{p})}{\partial \mathbf{p}^T} \\ \vdots \\ \frac{\partial \mathcal{W}(\mathbf{x}_N;\mathbf{p})}{\partial \mathbf{p}^T} \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \mathcal{I}_{t+1}(\mathbf{x}_1^{'}) - \mathcal{I}_t(\mathbf{x}_1) \\ \vdots \\ \mathcal{I}_{t+1}(\mathbf{x}_N^{'}) - \mathcal{I}_t(\mathbf{x}_N) \end{bmatrix}$$

- What conditions must $\mathbf{A}^T\mathbf{A}$ meet so that a unique solution to $\Delta\mathbf{p}$ can be found?
$\mathbf{A}^T\mathbf{A}$ should be invertible so that a unique solution to $\Delta\mathbf{p}$ can be found.

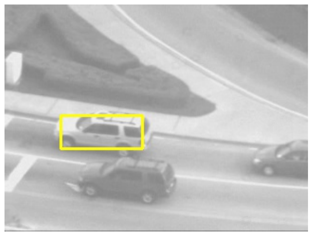
Q1.2: Function

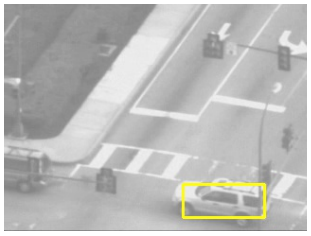
p = `LucasKanade(It, It1, rect, p0=np.zeros(2))`

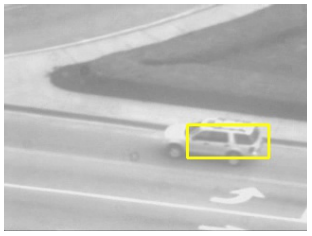
is completed to compute the optimal local motion from frame $\mathcal{I}_t$ to $\mathcal{I}_{t+1}$

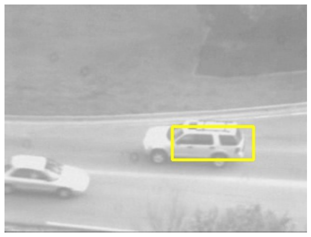
Q1.3: Script `testCarSequence.py` is completed to load the video frames and use the Lucas-Kanade tracker to track the car.

File `carseqrects.npy` is saved to record the `rect` obtained in each frame. For frames 1, 100, 200, 300 and 400, the tracking results are shown as Fig.1.


(a) frame1


(b) frame100


(c) frame200


(d) frame300

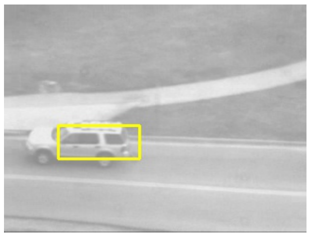
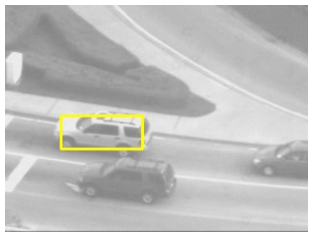
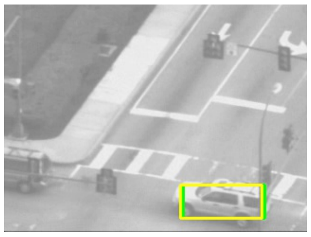

(e) frame400

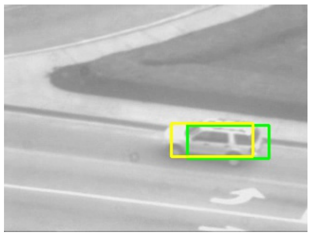
Figure 1: The car tracking results of frames 1, 100, 200, 300 and 400

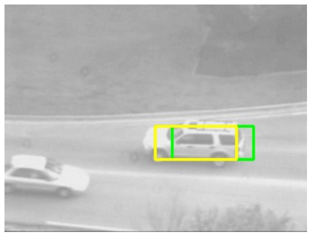
Q1.4: Script `testCarSequenceWithTemplateCorrection.py` is completed to track the car with a template correction routine incorporated.

File `carseqrects-wcrt.npy` is saved to record the `rect` in each frame. For frames 1, 100, 200, 300 and 400, the tracking results are shown as Fig.2. In this figure, the yellow rectangles are created with the tracker in **Q1.4** and the green ones are created with the tracker in **Q1.3**.


(a) frame1


(b) frame100


(c) frame200


(d) frame300

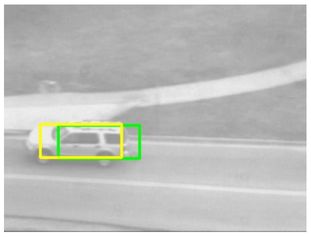

(e) frame400

Figure 2: The car tracking results of frames 1, 100, 200, 300 and 400

2 Lucas-Kanade Tracking with Appearance Basis

2.1 Appearance Basis

Q2.1:

$$\mathcal{I}_{t+1}(\mathbf{x}) = \mathcal{I}_t(\mathbf{x}) + \sum_{k=1}^{K} w_k \mathcal{B}_k(\mathbf{x})$$

$$\rightarrow \quad \begin{bmatrix} \mathcal{B}_1(\mathbf{x}) & \cdots & \mathcal{B}_K(\mathbf{x}) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_k \end{bmatrix} = \mathcal{I}_{t+1}(\mathbf{x}) - \mathcal{I}_t(\mathbf{x})$$

$$\rightarrow \quad \begin{bmatrix} \mathcal{B}_1(\mathbf{x})^T \\ \vdots \\ \mathcal{B}_K(\mathbf{x})^T \end{bmatrix} \begin{bmatrix} \mathcal{B}_1(\mathbf{x}) & \cdots & \mathcal{B}_K(\mathbf{x}) \end{bmatrix} \mathbf{w} = \begin{bmatrix} \mathcal{B}_1(\mathbf{x})^T \\ \vdots \\ \mathcal{B}_K(\mathbf{x})^T \end{bmatrix} [\mathcal{I}_{t+1}(\mathbf{x}) - \mathcal{I}_t(\mathbf{x})]$$

$$\rightarrow \quad \begin{bmatrix} \mathcal{B}_1(\mathbf{x})^T\mathcal{B}_1(\mathbf{x}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathcal{B}_K(\mathbf{x})^T\mathcal{B}_K(\mathbf{x}) \end{bmatrix} \mathbf{w} = \begin{bmatrix} \mathcal{B}_1(\mathbf{x})^T \\ \vdots \\ \mathcal{B}_K(\mathbf{x})^T \end{bmatrix} [\mathcal{I}_{t+1}(\mathbf{x}) - \mathcal{I}_t(\mathbf{x})]$$

$$\rightarrow \quad \mathbf{w} = \begin{bmatrix} \frac{1}{||\mathcal{B}_1(\mathbf{x})||}\mathcal{B}_1(\mathbf{x})^T \\ \vdots \\ \frac{1}{||\mathcal{B}_K(\mathbf{x})||}\mathcal{B}_K(\mathbf{x})^T \end{bmatrix} [\mathcal{I}_{t+1}(\mathbf{x}) - \mathcal{I}_t(\mathbf{x})]$$

2.2 Tracking

Q2.2: Function

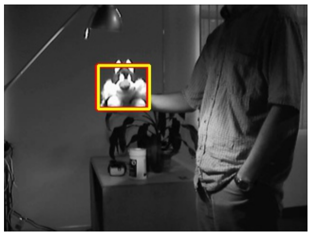
```
LucasKanadeBasis(It, It1, rect, bases, p0 = np.zeros(2))
```

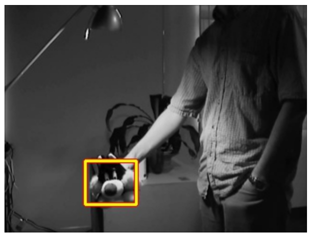
is implemented to compute the optimal local motion from $\mathcal{I}_t$ to $\mathcal{I}_{t+1}$ with bases.

Q2.3: Script `testSylvSequence.py` is completed to track the toy with the new Lucas-Kanade tracker (with bases). For frames 1, 200, 300, 350 and 400, the tracking results are shown as Fig.3. In these figures, the yellow rectangles are created with the tracker (with bases) in **Q2.2** and the red ones are the results obtained with the tracker in **Q1.2**.

(a) frame1

(b) frame200


(c) frame300


(d) frame350

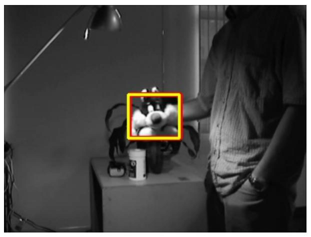

(e) frame400

Figure 3: The toy tracking results of frames 1, 200, 300, 350 and 400

3 Affine Motion Subtraction

3.1 Dominant Motion Estimation

Q3.1: Function

`LucasKanadeAffine(It, It1)`

is implemented to find the optimal affine transformation matrix $\mathbf{M}$ between the images $\mathcal{I}_t$ and $\mathcal{I}_{t+1}$.

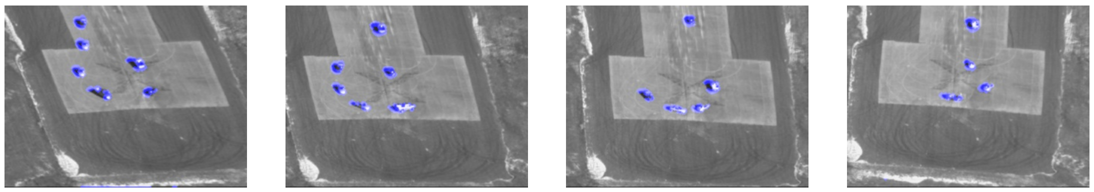
3.2 Moving Object Detection

Q3.2: Function

`SubtractDominantMotion(image1, image2)`

is implemented to find the transformation matrix $\mathbf{M}$ between the given images and produce the aforementioned binary mask for dominant motion estimation.

Q3.3: Script `testAerialSequence.py` is implemented to load the image sequence from `aerialseq.npy` and estimate the dominant motion to detect the moving objects. For frames 30, 60, 90 and 120, the detection results are shown as Fig.4.



(a) frame30 (b) frame600 (c) frame90 (d) frame120

Figure 4: The moving objects detection results of frames 30, 60, 90 and 120

4 Efficient Tracking

4.1 Inverse Composition

Q4.1: Function

`LucasKanadeAffine(It,It1)`

is completed for efficient tracking using the inverse compositional method.

The reason that the inverse compositional approach is more computationally efficient than the classical approach is that when calculating $\nabla \mathbf{p} = [\mathbf{A}^T\mathbf{A}]^{-1}\mathbf{A}^T\mathbf{b}$,
in classical approach,

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \mathcal{I}_{t+1}\{\mathcal{W}(\mathbf{x}_1;\mathbf{p})\}}{\partial \mathcal{W}(\mathbf{x}_1;\mathbf{p})^T} & \cdots & \mathbf{0}^T \\ \vdots & \ddots & \vdots \\ \mathbf{0}^T & \cdots & \frac{\partial \mathcal{I}_{t+1}\{\mathcal{W}(\mathbf{x}_N;\mathbf{p})\}}{\partial \mathcal{W}(\mathbf{x}_N;\mathbf{p})^T} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{W}(\mathbf{x}_1;\mathbf{p})}{\partial \mathbf{p}^T} \\ \vdots \\ \frac{\partial \mathcal{W}(\mathbf{x}_N;\mathbf{p})}{\partial \mathbf{p}^T} \end{bmatrix}$$

Since $\mathbf{p}$ is always changing, $\mathbf{A}$ is constantly changing. So in each iteration, the matrix $\mathbf{A}$ has to be recalculated.
But in the inverse compositional method,

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \mathcal{I}_t(\mathbf{x}_1)}{\partial \mathbf{x}_1^T} & \cdots & \mathbf{0}^T \\ \vdots & \ddots & \vdots \\ \mathbf{0}^T & \cdots & \frac{\partial \mathcal{I}_t(\mathbf{x}_N)}{\partial \mathbf{x}_N^T} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{W}(\mathbf{x}_1;\mathbf{0})}{\partial \mathbf{p}^T} \\ \vdots \\ \frac{\partial \mathcal{W}(\mathbf{x}_N;\mathbf{0})}{\partial \mathbf{p}^T} \end{bmatrix}$$

The matrix $\mathbf{A}$ is static in each iteration and just need to calculate it once. So it is more efficient.

4.2 Correlation Filters

Q4.2:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}||\mathbf{y} - \mathbf{X}^T\mathbf{g}||_2^2 + \frac{\lambda}{2}||\mathbf{g}||_2^2 \\ &= \frac{1}{2}(\mathbf{y}^T - \mathbf{g}^T\mathbf{X})(\mathbf{y} - \mathbf{X}^T\mathbf{g}) + \frac{\lambda}{2}\mathbf{g}^T\mathbf{g} \\ &= \frac{1}{2}(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}^T\mathbf{g} + \mathbf{g}^T\mathbf{X}\mathbf{X}^T\mathbf{g}) + \frac{\lambda}{2}\mathbf{g}^T\mathbf{g} \end{aligned}$$

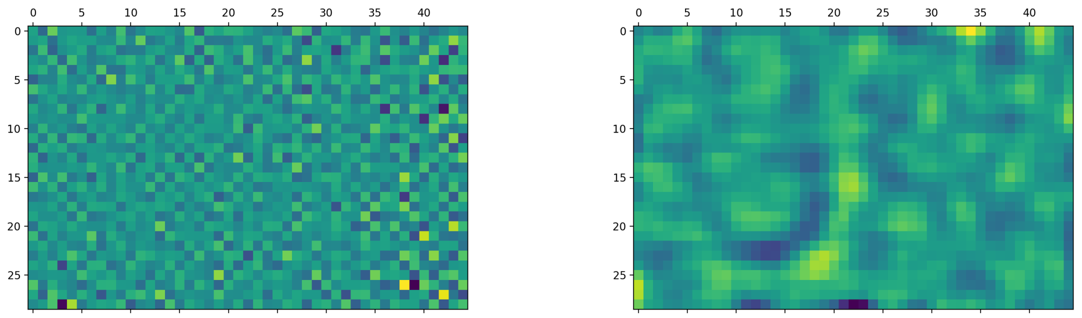
$$\begin{aligned}\rightarrow \quad \frac{\partial \mathcal{L}}{\partial \mathbf{g}} &= -\mathbf{X}\mathbf{y} + \mathbf{X}\mathbf{X}^T\mathbf{g} + \lambda\mathbf{g} \\ &= -\mathbf{X}\mathbf{y} + (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\mathbf{g} \\ &= -\mathbf{X}\mathbf{y} + (\mathbf{S} + \lambda\mathbf{I})\mathbf{g} = 0\end{aligned}$$

So,

$$\mathbf{g} = (\mathbf{S} + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{y}$$

Q4.3:

The resultant linear discriminant weight vector $\mathbf{g}$ for the penalty values $\lambda = 0$ and $\lambda = 1$ is visualized as Fig.5.



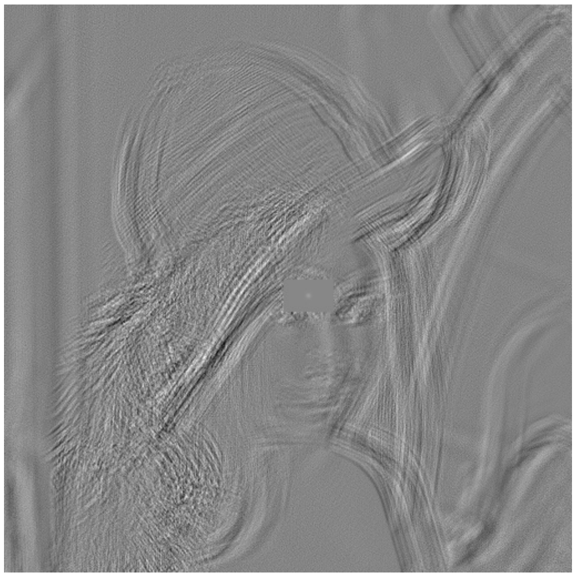
(a) $\lambda = 0$ (b) $\lambda = 1$

Figure 5: The visualized result of the resultant linear discriminant weight vector

The responses of the correlate filters on Lena image are shown as Fig.6. When $\lambda = 1$, it performs better. Compared to $\lambda = 0$, it applies the trust region method when calculating the linear discriminant weight vector $\mathbf{g}$, thus it would get a better result with the trust region.

Q4.4:

The responses of the convolve filters on Lena image are shown as Fig.7. This get a different response to the one obtained using `correlate`. This is because that during the convolution calculation, the image would be flipped but not in the correlation calculation. So the results would be different. We can use the numpy indexing operations to flip the filter $\mathbf{g}$ (use `g[::-1, ::-1]`) before performing convolution in order to get a response more similar to the one obtained using `correlate`.


(a) $\lambda = 0$

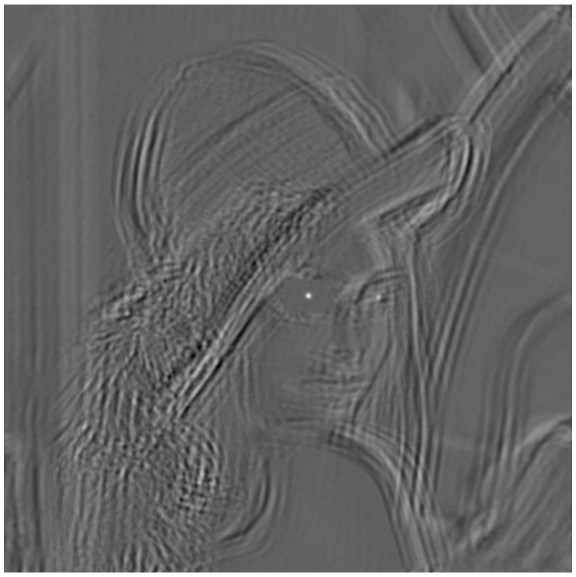
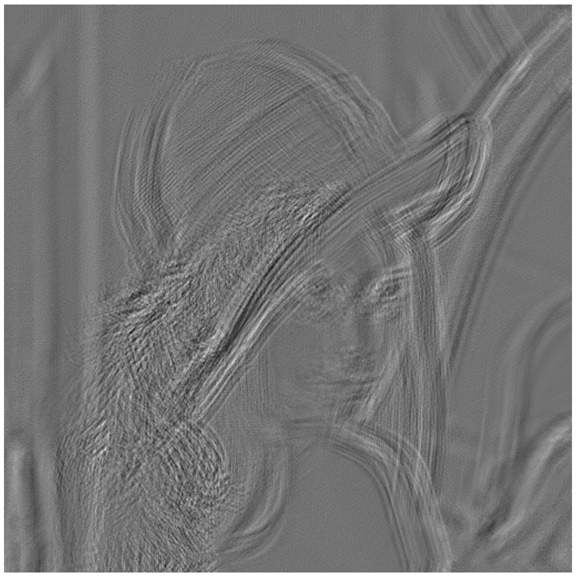

(b) $\lambda = 1$

Figure 6: The response of correlation.


(a) $\lambda = 0$

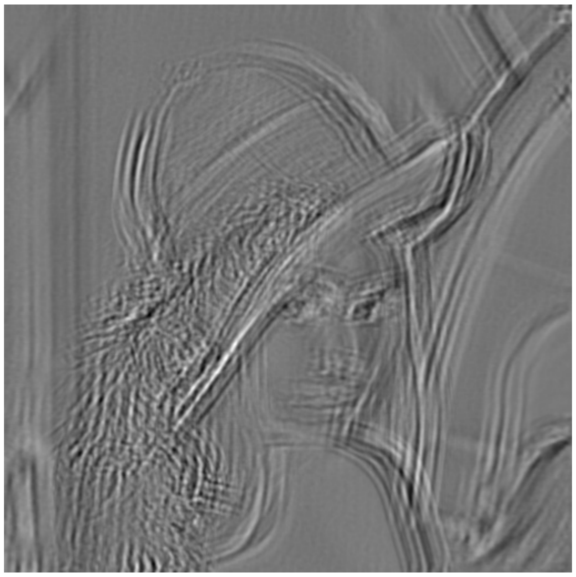

(b) $\lambda = 1$

Figure 7: The response of convolution