

16-720A — Spring 2021 — Homework 3

Jen-Hung Ho
jenhungh@andrew.cmu.edu

March 27, 2021

Question 1

Q1.1

We know that the pure translation warp function is

$$\mathcal{W}(\mathbf{x}; \mathbf{p}) = \mathbf{x} + \mathbf{p} \quad (1)$$

$$\mathcal{W}(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} \mathcal{W}_x(x; p_x) \\ \mathcal{W}_y(y; p_y) \end{bmatrix} = \begin{bmatrix} x + p_x \\ y + p_y \end{bmatrix} \quad (2)$$

Take partial derivative with respect to \mathbf{p} , we can get

$$\frac{\partial \mathcal{W}(\mathbf{x}; \mathbf{p})}{\partial \mathbf{p}^T} = \begin{bmatrix} \frac{\partial \mathcal{W}_x(x; p_x)}{\partial p_x} & \frac{\partial \mathcal{W}_x(x; p_x)}{\partial p_y} \\ \frac{\partial \mathcal{W}_y(y; p_y)}{\partial p_x} & \frac{\partial \mathcal{W}_y(y; p_y)}{\partial p_y} \end{bmatrix} = \begin{bmatrix} \frac{\partial(x+p_x)}{\partial p_x} & \frac{\partial(x+p_x)}{\partial p_y} \\ \frac{\partial(y+p_y)}{\partial p_x} & \frac{\partial(y+p_y)}{\partial p_y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (3)$$

The Least Square problem we try to solve

$$\underset{\Delta \mathbf{p}}{\operatorname{argmin}} \sum_{x \in \mathbb{N}} \|\mathcal{I}_{t+1}(\mathbf{x} + \mathbf{p}) - \mathcal{I}_t(\mathbf{x})\|_2^2 \quad (4)$$

$$= \underset{\Delta \mathbf{p}}{\operatorname{argmin}} \sum_{x \in \mathbb{N}} \left\| \mathcal{I}_{t+1}(\mathbf{x}') + \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T} \frac{\partial \mathcal{W}(\mathbf{x}; \mathbf{p})}{\partial \mathbf{p}^T} \Delta \mathbf{p} - \mathcal{I}_t(\mathbf{x}) \right\|_2^2 \quad (5)$$

$$= \underset{\Delta \mathbf{p}}{\operatorname{argmin}} \sum_{x \in \mathbb{N}} \left\| \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T} \frac{\partial \mathcal{W}(\mathbf{x}; \mathbf{p})}{\partial \mathbf{p}^T} \Delta \mathbf{p} - (\mathcal{I}_t(\mathbf{x}) - \mathcal{I}_{t+1}(\mathbf{x}')) \right\|_2^2 \quad (6)$$

$$= \underset{\Delta \mathbf{p}}{\operatorname{argmin}} \|\mathbf{A} \Delta \mathbf{p} - \mathbf{b}\|_2^2 \quad (7)$$

From equation 6 and 7, we know that

$$\mathbf{A} = \sum_{x \in \mathbb{N}} \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T} \frac{\partial \mathcal{W}(\mathbf{x}; \mathbf{p})}{\partial \mathbf{p}^T} = \begin{bmatrix} \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}'_1)}{\partial \mathbf{x}'_1{}^T} & \dots & \mathbf{0}^T \\ \vdots & \ddots & \vdots \\ \mathbf{0}^T & \dots & \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}'_N)}{\partial \mathbf{x}'_N{}^T} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{W}(\mathbf{x}_1; \mathbf{p})}{\partial \mathbf{p}^T} \\ \vdots \\ \frac{\partial \mathcal{W}(\mathbf{x}_N; \mathbf{p})}{\partial \mathbf{p}^T} \end{bmatrix} \quad (8)$$

$$\mathbf{b} = \sum_{x \in \mathbb{N}} \mathcal{I}_{t+1}(\mathbf{x}) - \mathcal{I}_{t+1}(\mathbf{x}') = \begin{bmatrix} \mathcal{I}_t(\mathbf{x}_1) - \mathcal{I}_{t+1}(\mathbf{x}'_1) \\ \vdots \\ \mathcal{I}_t(\mathbf{x}_N) - \mathcal{I}_{t+1}(\mathbf{x}'_N) \end{bmatrix} \quad (9)$$

We can solve the Least Square problem by using the pseudo inverse.

$$\Delta \mathbf{p} = \sum_{x \in \mathbb{N}} (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T b \quad (10)$$

From equation 10, we know that $\mathbf{A}^T \mathbf{A}$ must be invertible so that an unique solution to $\Delta \mathbf{p}$ can be found.

Q1.3



Figure 1: Car Sequence: Lucas-Kanade Tracking with One Single Template



Figure 2: Girl Sequence: Lucas-Kanade Tracking with One Single Template

Q1.4



Figure 3: Car Sequence: Lucas-Kanade Tracking with Template Correction (blue: without correction, red: with correction)



Figure 4: Girl Sequence: Lucas-Kanade Tracking with Template Correction (blue: without correction, red: with correction)

Question 2

Q2.3

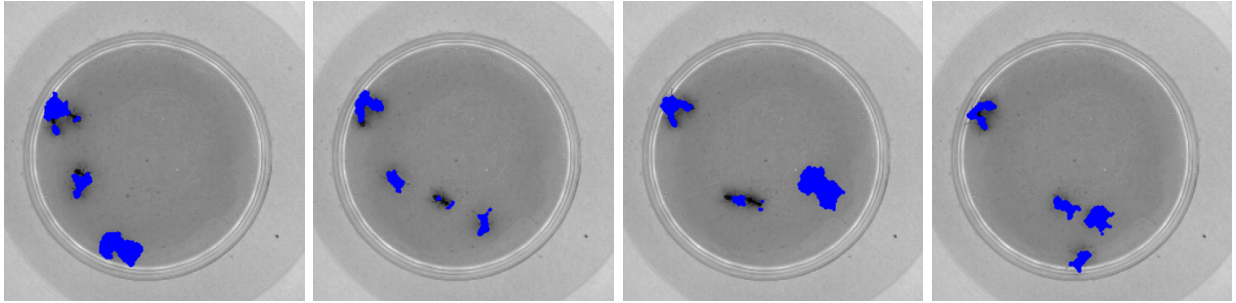


Figure 5: Ant Sequence: Lucas-Kanade Tracking with Motion Detection

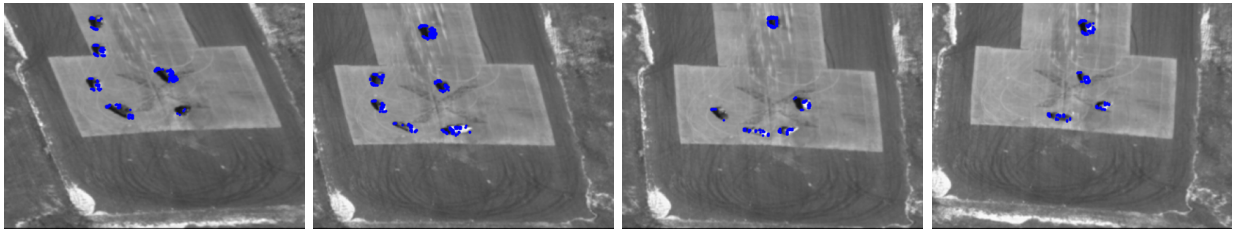


Figure 6: Aerial Sequence: Lucas-Kanade Tracking with Motion Detection

Question 3

Q3.2

The reason that inverse compositional approach is more computationally efficient than the classical approach is because of the benefits to precompute matrix \mathbf{A} and the Hessian matrix ($\mathbf{H} = \mathbf{A}^T \mathbf{A}$). The calculation of the gradients and the jacobians are all moved out from the iteration loop, which saves a lot of time and thus increases the computational efficiency.

Code Appendix

0.1 LucasKanade.py

```

1  import numpy as np
2  from scipy.interpolate import RectBivariateSpline
3
4  def LucasKanade(It, It1, rect, threshold, num_iters, p0 = np.zeros(2)):
5      """
6          :param It: template image
7          :param It1: Current image
8          :param rect: Current position of the car (top left, bot right coordinates)
9          :param threshold: if the length of dp is smaller than the threshold,
10                          terminate the optimization
11          :param num_iters: number of iterations of the optimization
12          :param p0: Initial movement vector [dp_x0, dp_y0]
13          :return: p: movement vector [dp_x, dp_y]
14      """
15
16      # Set up the initial parameter guess and the corners of the rectangle
17      p = p0
18      # Print out rectangle to check
19      x1, y1, x2, y2 = rect
20      print(f"rect = {x1}, {y1}, {x2}, {y2}")
21
22      # Compute the spline for sub-pixel interpolation
23      h, w = It.shape
24      h1, w1 = It1.shape
25      spline_temp = RectBivariateSpline(np.arange(h), np.arange(w), It)
26      spline_current = RectBivariateSpline(np.arange(h1), np.arange(w1), It1)
27
28      # Compute I_temp
29      xx, yy = np.meshgrid(np.arange(x1, x2 + 0.1), np.arange(y1, y2 + 0.1))
30      total_size = xx.shape[0] * xx.shape[1]
31      I_temp = spline_temp.ev(yy, xx)
32
33      # Iterations
34      for iters in range(int(num_iters)):
35          # Compute I_current
36          I_current = spline_current.ev(yy + p[1], xx + p[0])
37
38          # Calculate the gradient: dI/dx and dI/dy
39          dIdx = spline_current.ev(yy + p[1], xx + p[0], dy = 1).flatten()
40          dIdy = spline_current.ev(yy + p[1], xx + p[0], dx = 1).flatten()
41
42          # Calculate the jacobian: dW/dp
43          dWdp = np.eye(2)
44

```

```
45     # Compute A and b
46     # Compute A = dI/dX * dW/dp
47     A = np.zeros((total_size, 2))
48     for index in range(total_size):
49         A[index, 0] = dIdx[index] * dWdp[0, 0] + dIdy[index] * dWdp[1, 0]
50         A[index, 1] = dIdx[index] * dWdp[0, 1] + dIdy[index] * dWdp[1, 1]
51     # Compute b
52     b = (I_temp - I_current).reshape(-1, 1)
53
54     # Solve the Least Square problem using Pseudo Inverse
55     dp = np.linalg.inv(A.T @ A) @ A.T @ b
56
57     # Update p
58     p = np.asarray([p[0] + dp[0, 0], p[1] + dp[1, 0]])
59
60     # Check the norm of dp
61     if (np.linalg.norm(dp, ord = 2) < threshold):
62         break
63
64     return p
```

0.2 LucasKanadeAffine.py

```

1  import numpy as np
2  from scipy.interpolate import RectBivariateSpline
3
4  def LucasKanadeAffine(It, It1, threshold, num_iters):
5      """
6          :param It: template image
7          :param It1: Current image
8          :param threshold: if the length of dp is smaller than the threshold,
9                          terminate the optimization
10         :param num_iters: number of iterations of the optimization
11         :return: M: the Affine warp matrix [2x3 numpy array] put your implementation
12                 here
13         """
14
15         # Set up the initial affine matrix M
16         M = np.array([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0]])
17
18         # Compute the spline for interpolation
19         h, w = It.shape
20         h1, w1 = It1.shape
21         spline_temp = RectBivariateSpline(np.arange(h), np.arange(w), It)
22         spline_current = RectBivariateSpline(np.arange(h1), np.arange(w1), It1)
23
24         # Compute the meshgrid of the template
25         XX, YY = np.meshgrid(np.arange(w), np.arange(h))
26
27         # Iterations
28         for iters in range(int(num_iters)):
29             # Warp the image: compute X_bar
30             xx, yy = XX, YY
31             xx_bar = M[0, 0] * xx + M[0, 1] * yy + M[0, 2] * np.ones_like(xx)
32             yy_bar = M[1, 0] * xx + M[1, 1] * yy + M[1, 2] * np.ones_like(xx)
33
34             # Check the valid area
35             valid_index = (xx_bar >= 0) & (xx_bar[0] < w) & (yy_bar >= 0) &
36                                     (yy_bar < h)
37             xx, yy = xx[valid_index], yy[valid_index]
38             xx_bar, yy_bar = xx_bar[valid_index], yy_bar[valid_index]
39
40             # Compute I_temp and I_current
41             I_temp = spline_temp.ev(yy, xx).flatten()
42             I_current = spline_current.ev(yy_bar, xx_bar).flatten()
43
44             # Calculate the gradient: dI/dx and dI/dy
45             dIdx = spline_current.ev(yy_bar, xx_bar, dy = 1).flatten()
46             dIdy = spline_current.ev(yy_bar, xx_bar, dx = 1).flatten()

```

```

47
48     # Calculate the jacobian: dW/dp
49     dWdp1 = xx.flatten()
50     dWdp2 = yy.flatten()
51     dWdp3 = np.ones_like(xx.flatten())
52     dWdp4 = xx.flatten()
53     dWdp5 = yy.flatten()
54     dWdp6 = np.ones_like(xx.flatten())
55
56     # Compute A and b
57     # Compute A = dI/dX * dW/dp
58     A = np.zeros((xx.flatten().shape[0], 6))
59     A[:, 0] = dIdx * dWdp1
60     A[:, 1] = dIdx * dWdp2
61     A[:, 2] = dIdx * dWdp3
62     A[:, 3] = dIdy * dWdp4
63     A[:, 4] = dIdy * dWdp5
64     A[:, 5] = dIdy * dWdp6
65     # Compute b
66     b = (I_temp - I_current).reshape(-1, 1)
67
68     # Solve the Least Square problem using Pseudo Inverse
69     dp = np.linalg.inv(A.T @ A) @ A.T @ b
70
71     # Update M
72     M[0, :] += dp[0:3, 0]
73     M[1, :] += dp[3:6, 0]
74
75     # Check the norm of dp
76     if (np.linalg.norm(dp, ord = 2) < threshold):
77         break
78
79     return M

```


0.3 SubtractDominantMotion.py

```

1  import numpy as np
2  from scipy.interpolate import RectBivariateSpline
3  from LucasKanadeAffine import LucasKanadeAffine
4  from InverseCompositionAffine import InverseCompositionAffine
5
6  def SubtractDominantMotion(image1, image2, threshold, num_iters, tolerance):
7      """
8          :param image1: Images at time t
9          :param image2: Images at time t+1
10         :param threshold: used for LucasKanadeAffine
11         :param num_iters: used for LucasKanadeAffine
12         :param tolerance: binary threshold of intensity difference when computing
13                         the mask
14         :return: mask: [nxm]
15         """
16
17         # Initialize the binary mask
18         mask = np.zeros(image1.shape, dtype=bool)
19
20         # Compute the spline for interpolation
21         h1, w1 = image1.shape
22         h2, w2 = image2.shape
23         spline1 = RectBivariateSpline(np.arange(h1), np.arange(w1), image1)
24         spline2 = RectBivariateSpline(np.arange(h2), np.arange(w2), image2)
25
26         # Affine Warp
27         # Compute the meshgrid of image1
28         xx, yy = np.meshgrid(np.arange(w1), np.arange(h1))
29         # Compute transformation matrix M : LucasKanadeAffine or
30         # InverseCompositionAffine
31         M = LucasKanadeAffine(image1, image2, threshold, num_iters)
32         # M = InverseCompositionAffine(image1, image2, threshold, num_iters)
33         print(f"M = {M}")
34         # Warp image1
35         xx_bar = M[0, 0] * xx + M[0, 1] * yy + M[0, 2] * np.ones_like(xx)
36         yy_bar = M[1, 0] * xx + M[1, 1] * yy + M[1, 2] * np.ones_like(xx)
37
38         # Check the valid area
39         valid_index = (xx_bar >= 0) & (xx_bar < w1) & (yy_bar >= 0) & (yy_bar < h1)
40         valid_map = valid_index.astype(int)
41         xx, yy = xx * valid_map, yy * valid_map
42         xx_bar, yy_bar = xx_bar * valid_map, yy_bar * valid_map
43
44         # Compute Image1_warp and Image2
45         Image1_warp = spline1.ev(yy, xx)
46         Image2 = spline2.ev(yy_bar, xx_bar)

```

```
47  
48     # Compute the subtraction  
49     diff = abs(Image2 - Image1_warp)  
50     mask[diff > tolerance] = 1  
51  
52     return mask
```

0.4 InverseCompositionAffine.py

```

1  import numpy as np
2  from scipy.interpolate import RectBivariateSpline
3
4  def InverseCompositionAffine(It, It1, threshold, num_iters):
5      """
6          :param It: template image
7          :param It1: Current image
8          :param threshold: if the length of dp is smaller than the threshold,
9                          terminate the optimization
10         :param num_iters: number of iterations of the optimization
11         :return: M: the Affine warp matrix [2x3 numpy array]
12         """
13
14         # Set up the initial M
15         M = np.array([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0]])
16
17         # Compute the spline for interpolation
18         h, w = It.shape
19         h1, w1 = It1.shape
20         spline_temp = RectBivariateSpline(np.arange(h), np.arange(w), It)
21         spline_current = RectBivariateSpline(np.arange(h1), np.arange(w1), It1)
22
23         # Compute the meshgrid of the template
24         xx, yy = np.meshgrid(np.arange(w), np.arange(h))
25
26         # Compute A
27         # Calculate the gradient: dI/dx and dI/dy
28         dIdx = spline_temp.ev(yy, xx, dy = 1).flatten()
29         dIdy = spline_temp.ev(yy, xx, dx = 1).flatten()
30         # Calculate the jacobian: dW/dp
31         dWdp1 = xx.flatten()
32         dWdp2 = yy.flatten()
33         dWdp3 = np.ones_like(xx.flatten())
34         dWdp4 = xx.flatten()
35         dWdp5 = yy.flatten()
36         dWdp6 = np.ones_like(xx.flatten())
37         # A = dI/dX * dW/dp
38         A = np.zeros((xx.flatten().shape[0], 6))
39         A[:, 0] = dIdx * dWdp1
40         A[:, 1] = dIdx * dWdp2
41         A[:, 2] = dIdx * dWdp3
42         A[:, 3] = dIdy * dWdp4
43         A[:, 4] = dIdy * dWdp5
44         A[:, 5] = dIdy * dWdp6
45
46         # Compute the Hessian = A.T @ A

```

```

47     Hessian = A.T @ A
48
49     # Iterations
50     for iters in range(int(num_iters)):
51         # Warp the image: compute X_bar
52         xx_bar = M[0, 0] * xx + M[0, 1] * yy + M[0, 2] * np.ones_like(xx)
53         yy_bar = M[1, 0] * xx + M[1, 1] * yy + M[1, 2] * np.ones_like(xx)
54
55         # Check the valid area and set the error to 0 if invalid
56         valid_index = (xx_bar >= 0) & (xx_bar[0] < w) & (yy_bar >= 0) &
57                     (yy_bar < h)
58         valid_map = valid_index.astype(int)
59         xx, yy = xx * valid_map, yy * valid_map
60         xx_bar, yy_bar = xx_bar * valid_map, yy_bar * valid_map
61
62         # Compute I_temp and I_current
63         I_temp = spline_temp.ev(yy, xx).flatten()
64         I_current = spline_current.ev(yy_bar, xx_bar).flatten()
65
66         # A and Hessian are precomputed
67         # Compute b
68         b = (I_current - I_temp).reshape(-1, 1)
69
70         # Solve the Least Square problem using Pseudo Inverse
71         # Compute dp and dM
72         dp = np.linalg.inv(Hessian) @ A.T @ b
73         dM = np.array([[1 + dp[0, 0], dp[1, 0], dp[2, 0]],
74                       [dp[3, 0], 1 + dp[4, 0], dp[5, 0]],
75                       [0, 0, 1]])
76
77         # Update M
78         M = np.concatenate((M, [[0, 0, 1]]), axis = 0)
79         M = M @ np.linalg.inv(dM)
80         M = M[0:2, :]
81
82         # Check dp
83         if (np.linalg.norm(dp, ord = 2) < threshold):
84             break
85
86     return M

```

0.5 testCarSequence.py

```

1  import argparse
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.patches as patches
5  import os
6  from LucasKanade import LucasKanade
7
8  # write your script here, we recommend the above libraries
9  # Set up iterations and threshold
10 parser = argparse.ArgumentParser()
11 parser.add_argument('--num_iters', type=int, default=1e4, help='number of
12                        iterations of Lucas-Kanade')
13 parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold
14                        of Lucas-Kanade for terminating optimization')
15 args = parser.parse_args()
16 num_iters = args.num_iters
17 threshold = args.threshold
18
19 # Load the video frames and Set up the reported frames
20 seq = np.load("../data/carseq.npy")
21 reported_frames = [0, 99, 199, 299, 399]
22
23 # Set up the initial rectangle and the rectangle history
24 rect = [59, 116, 145, 151]
25 rect_history = []
26 rect_history.append(rect)
27
28 # Apply Lucas-Kanade tracker to every frames
29 num_frames = seq.shape[2]
30 for index in range(num_frames-1):
31     # Print out the frame index
32     print(f"frame = {index+1}")
33
34     # Load template and current frames
35     It = seq[:, :, index]
36     It1 = seq[:, :, index+1]
37
38     # Apply Lucas-Kanade tracker
39     p = LucasKanade(It, It1, rect, threshold, num_iters)
40
41     # Update and Append rectangle
42     rect = [rect[0] + p[0], rect[1] + p[1], rect[2] + p[0], rect[3] + p[1]]
43     rect_history.append(rect)
44
45     # Plot the reported frames
46     if index in reported_frames:

```

```
47     # Plot frames
48     fig = plt.figure()
49     ax = fig.add_subplot(1, 1, 1)
50     plt.imshow(It, cmap = 'gray')
51     plt.axis('off')
52
53     # Plot the rectangles
54     x1, y1, x2, y2 = rect_history[index]
55     width, height = (x2 - x1), (y2 - y1)
56     rectangle = patches.Rectangle((x1, y1), width, height, edgecolor = 'r',
57                                   facecolor = 'none', linewidth = 3)
58     ax.add_patch(rectangle)
59
60     # Save the figures
61     plots_dir = '../plots/'
62     if not os.path.exists(plots_dir):
63         os.makedirs(plots_dir)
64     fig.savefig('../plots/Car_f' + str(index + 1) + '.png',
65                 bbox_inches = 'tight', pad_inches = 0)
66
67     # Save the rectangle history
68     results_dir = '../results/'
69     if not os.path.exists(results_dir):
70         os.makedirs(results_dir)
71     np.save('../results/carseqrects.npy', np.asarray(rect_history))
```

0.6 testCarSequenceWithTemplateCorrection.py

```

1  import argparse
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.patches as patches
5  import os
6  from LucasKanade import LucasKanade
7
8  # write your script here, we recommend the above libraries
9  # Set up iterations and thresholds
10 parser = argparse.ArgumentParser()
11 parser.add_argument('--num_iters', type=int, default=1e4, help='number of
12                        iterations of Lucas-Kanade')
13 parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold
14                        of Lucas-Kanade for terminating optimization')
15 parser.add_argument('--template_threshold', type=float, default=5,
16                        help='threshold for determining whether to update template')
17 args = parser.parse_args()
18 num_iters = args.num_iters
19 threshold = args.threshold
20 template_threshold = args.template_threshold
21
22 # Load the video frames and Set up the reported frames
23 seq = np.load("../data/carseq.npy")
24 reported_frames = [0, 99, 199, 299, 399]
25
26 # Set up initial guess, initial template, and first frame
27 # The first frame is used to correct the drift
28 p0 = np.zeros(2)
29 template = seq[:, :, 0]
30 template_first = seq[:, :, 0]
31
32 # Set up the initial rectangle and the rectangle history
33 rect_initial = [59, 116, 145, 151]
34 rect = rect_initial
35 rect_history = []
36 rect_history.append(rect)
37
38 # Apply Lucas-Kanade tracker with Template Correction to every frames
39 num_frames = seq.shape[2]
40 for index in range(num_frames-1):
41     # Print out the frame index
42     print(f"frame = {index+1}")
43
44     # Load the current frames
45     It1 = seq[:, :, index+1]
46

```

```

47     # Apply Lucas-Kanade tracker with Template Correction
48     # Compute p
49     p = LucasKanade(template, It1, rect, threshold, num_iters, p0)
50     # Compute pn
51     pn_x = (rect[0] + p[0]) - rect_initial[0]
52     pn_y = (rect[1] + p[1]) - rect_initial[1]
53     pn = np.array([pn_x, pn_y])
54     # Compute pn_star
55     pn_star = LucasKanade(template_first, It1, rect_initial, threshold,
56                           num_iters, pn)
57
58     # Template Correction : Update the rectangle
59     # Naive Update
60     if (np.linalg.norm(pn_star - pn, ord = 2) <= template_threshold):
61         # Update the rectangle : drift correction
62         p_current_x = pn_star[0] - (rect[0] - rect_initial[0])
63         p_current_y = pn_star[1] - (rect[1] - rect_initial[1])
64         rect = [rect[0] + p_current_x, rect[1] + p_current_y,
65               rect[2] + p_current_x, rect[3] + p_current_y]
66         # Update template
67         template = seq[:, :, index+1]
68         # Reset p0
69         p0 = np.zeros(2)
70     # No Update
71     else:
72         p0 = p
73
74     # Append the rectangle
75     rect_history.append(rect)
76
77     # Plot the reported frames
78     rect_without_correction = np.load('../results/carseqrects.npy')
79     if index in reported_frames:
80         # Plot frames
81         fig = plt.figure()
82         ax = fig.add_subplot(1, 1, 1)
83         plt.imshow(seq[:, :, index], cmap = 'gray')
84         plt.axis('off')
85         # Plot the rectangles without correction
86         x1_wo, y1_wo, x2_wo, y2_wo = rect_without_correction[index, :]
87         width_wo, height_wo = (x2_wo - x1_wo), (y2_wo - y1_wo)
88         rectangle_wo = patches.Rectangle((x1_wo, y1_wo), width_wo, height_wo,
89                                         edgecolor = 'b', facecolor = 'None', linewidth = 3)
90         ax.add_patch(rectangle_wo)
91         # Plot the rectangles with correction
92         x1, y1, x2, y2 = rect_history[index]
93         rectangle = patches.Rectangle((x1, y1), (x2 - x1), (y2 - y1),
94                                     edgecolor = 'r', facecolor = 'None', linewidth = 2)

```



```
95         ax.add_patch(rectangle)
96
97         # Save the figures
98         plots_dir = '../plots/'
99         if not os.path.exists(plots_dir):
100             os.makedirs(plots_dir)
101         fig.savefig('../plots/Car-wcrt_f' + str(index+1) + '.png',
102                     bbox_inches = 'tight', pad_inches = 0)
103
104         # Save the rectangle history
105         results_dir = '../results/'
106         if not os.path.exists(results_dir):
107             os.makedirs(results_dir)
108         np.save('../results/carseqrects-wcrt.npy', np.asarray(rect_history))
```

0.7 testGirlSequence.py

```

1  import argparse
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.patches as patches
5  import os
6  from LucasKanade import LucasKanade
7
8  # write your script here, we recommend the above libraries
9  # Set up iterations and threshold
10 parser = argparse.ArgumentParser()
11 parser.add_argument('--num_iters', type=int, default=1e4, help='number of
12                        iterations of Lucas-Kanade')
13 parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold
14                        of Lucas-Kanade for terminating optimization')
15 args = parser.parse_args()
16 num_iters = args.num_iters
17 threshold = args.threshold
18
19 # Load the video frames and Set up the reported frames
20 seq = np.load("../data/girlseq.npy")
21 reported_frames = [0, 19, 39, 59, 79]
22
23 # Set up the initial rectangle and the rectangle history
24 rect = [280, 152, 330, 318]
25 rect_history = []
26 rect_history.append(rect)
27
28 # Apply Lucas-Kanade tracker to every frames
29 num_frames = seq.shape[2]
30 for index in range(num_frames-1):
31     # Print out the frame index
32     print(f"frame = {index+1}")
33
34     # Load template and current frames
35     It = seq[:, :, index]
36     It1 = seq[:, :, index+1]
37
38     # Apply Lucas-Kanade tracker
39     p = LucasKanade(It, It1, rect, threshold, num_iters)
40
41     # Update and Append the rectangles
42     rect = [rect[0] + p[0], rect[1] + p[1], rect[2] + p[0], rect[3] + p[1]]
43     rect_history.append(rect)
44
45     # Plot the reported frames
46     if index in reported_frames:

```

```
47     # Plot the frames
48     fig = plt.figure()
49     ax = fig.add_subplot(1, 1, 1)
50     plt.imshow(It, cmap = 'gray')
51     plt.axis('off')
52
53     # Plot the rectangles
54     x1, y1, x2, y2 = rect_history[index]
55     width, height = (x2 - x1), (y2 - y1)
56     rectangle = patches.Rectangle((x1, y1), width, height, edgecolor = 'r',
57                                   facecolor = 'None', linewidth = 3)
58     ax.add_patch(rectangle)
59
60     # Save the figures
61     plots_dir = '../plots/'
62     if not os.path.exists(plots_dir):
63         os.makedirs(plots_dir)
64     fig.savefig('../plots/Girl_f' + str(index+1) + '.png',
65               bbox_inches = 'tight', pad_inches = 0)
66
67     # Save the rectangle history
68     results_dir = '../results/'
69     if not os.path.exists(results_dir):
70         os.makedirs(results_dir)
71     np.save('../results/girlseqrects.npy', np.asarray(rect_history))
```

0.8 testGirlSequenceWithTemplateCorrection.py

```

1  import argparse
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.patches as patches
5  import os
6  from LucasKanade import LucasKanade
7
8  # write your script here, we recommend the above libraries
9  # Set up iterations and thresholds
10 parser = argparse.ArgumentParser()
11 parser.add_argument('--num_iters', type=int, default=1e4, help='number of
12                        iterations of Lucas-Kanade')
13 parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold
14                        of Lucas-Kanade for terminating optimization')
15 parser.add_argument('--template_threshold', type=float, default=5,
16                        help='threshold for determining whether to update template')
17 args = parser.parse_args()
18 num_iters = args.num_iters
19 threshold = args.threshold
20 template_threshold = args.template_threshold
21
22 # Load the video frames and Set up the reported frames
23 seq = np.load("../data/girlseq.npy")
24 reported_frames = [0, 19, 39, 59, 79]
25
26 # Set up initial guess, initial template, and first frame
27 # The first frame is used to correct the drift
28 p0 = np.zeros(2)
29 template = seq[:, :, 0]
30 template_first = seq[:, :, 0]
31
32 # Set up the initial rectangle and the rectangle history
33 rect_initial = [280, 152, 330, 318]
34 rect = rect_initial
35 rect_history = []
36 rect_history.append(rect)
37
38 # Apply Lucas-Kanade tracker with Template Correction to every frames
39 num_frames = seq.shape[2]
40 for index in range(num_frames-1):
41     # Print out the frame index
42     print(f"frame = {index+1}")
43
44     # Load the current frames
45     It1 = seq[:, :, index+1]
46

```

```

47     # Apply Lucas-Kanade tracker with Template Correction
48     # Compute p
49     p = LucasKanade(template, It1, rect, threshold, num_iters, p0)
50     # Compute pn
51     pn_x = (rect[0] + p[0]) - rect_initial[0]
52     pn_y = (rect[1] + p[1]) - rect_initial[1]
53     pn = np.array([pn_x, pn_y])
54     # Compute pn_star
55     pn_star = LucasKanade(template_first, It1, rect_initial, threshold,
56                           num_iters, pn)
57
58     # Template Correction : Update the rectangle
59     # Naive Update
60     if (np.linalg.norm(pn_star - pn, ord = 2) <= template_threshold):
61         # Update the rectangle : drift correction
62         p_current_x = pn_star[0] - (rect[0] - rect_initial[0])
63         p_current_y = pn_star[1] - (rect[1] - rect_initial[1])
64         rect = [rect[0] + p_current_x, rect[1] + p_current_y,
65               rect[2] + p_current_x, rect[3] + p_current_y]
66         # Update template
67         template = seq[:, :, index+1]
68         # Reset p0
69         p0 = np.zeros(2)
70     # No Update
71     else:
72         p0 = p
73
74     # Append the rectangle
75     rect_history.append(rect)
76
77     # Plot the reported frames
78     rect_without_correction = np.load('../results/girlseqrects.npy')
79     if index in reported_frames:
80         fig = plt.figure()
81         ax = fig.add_subplot(1, 1, 1)
82         # Plot frames
83         plt.imshow(seq[:, :, index], cmap = 'gray')
84         plt.axis('off')
85         # Plot the rectangles without correction
86         x1_wo, y1_wo, x2_wo, y2_wo = rect_without_correction[index, :]
87         rectangle_wo = patches.Rectangle((x1_wo, y1_wo), (x2_wo - x1_wo),
88                                         (y2_wo - y1_wo), edgecolor = 'b', facecolor = 'None', linewidth = 3)
89         ax.add_patch(rectangle_wo)
90         # Plot the rectangles with correction
91         x1, y1, x2, y2 = rect
92         rectangle = patches.Rectangle((x1, y1), (x2 - x1), (y2 - y1),
93                                     edgecolor = 'r', facecolor = 'None', linewidth = 2)
94         ax.add_patch(rectangle)

```

```
95
96     # Save the figures
97     plots_dir = '../plots/'
98     if not os.path.exists(plots_dir):
99         os.makedirs(plots_dir)
100     fig.savefig('../plots/Girl-wcrt_f' + str(index+1) + '.png',
101                 bbox_inches = 'tight', pad_inches = 0)
102
103     # Save the rectangle history
104     results_dir = '../results/'
105     if not os.path.exists(results_dir):
106         os.makedirs(results_dir)
107     np.save('../results/girlseqrects-wcrt.npy', np.asarray(rect_history))
```

0.9 testAntSequence.py

```

1  import argparse
2  import numpy as np
3  import scipy.ndimage
4  import matplotlib.pyplot as plt
5  import matplotlib.patches as patches
6  import os
7  from SubtractDominantMotion import SubtractDominantMotion
8
9
10 # write your script here, we recommend the above libraries
11 # Set up iterations, threshold, and tolerance
12 parser = argparse.ArgumentParser()
13 parser.add_argument('--num_iters', type=int, default=1e3, help='number of
14                     iterations of Lucas-Kanade')
15 parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold
16                     of Lucas-Kanade for terminating optimization')
17 parser.add_argument('--tolerance', type=float, default=0.03, help='binary
18                     threshold of intensity difference when computing the mask')
19 args = parser.parse_args()
20 num_iters = args.num_iters
21 threshold = args.threshold
22 tolerance = args.tolerance
23
24 # Load the video frames and Set up the reported frames
25 seq = np.load('../data/antseq.npy')
26 reported_frames = [29, 59, 89, 119]
27
28 # Apply Subtract Dominant Motion to every frames
29 frames = seq.shape[2]
30 for index in range(frames-1):
31     # Print out the frame index
32     print(f"frame = {index+1}")
33
34     # Load images
35     image1 = seq[:, :, index]
36     image2 = seq[:, :, index+1]
37
38     # Apply Subtract Dominant Motion
39     mask = SubtractDominantMotion(image1, image2, threshold, num_iters,
40                                   tolerance)
41
42     # Improvement: binary dilation and erosion
43     mask = scipy.ndimage.binary_dilation(mask, iterations = 3)
44     mask = scipy.ndimage.binary_erosion(mask, iterations = 4)
45
46     # Extract the moving objects

```

```

47 target = np.where(mask == 1)
48
49 # Plot the reported frames
50 if index in reported_frames:
51     # Plot frames
52     fig = plt.figure()
53     ax = fig.add_subplot(1, 1, 1)
54     plt.imshow(image1, cmap = 'gray')
55     plt.axis('off')
56
57 # Plot the moving objects
58 plt.plot(target[1], target[0], 'bo', markersize = 2)
59
60 # Save the reported figures
61 plots_dir = '../plots/'
62 if not os.path.exists(plots_dir):
63     os.makedirs(plots_dir)
64 fig.savefig('../plots/Ant_f' + str(index+1) + '.png',
65             bbox_inches = 'tight', pad_inches = 0)

```


0.10 testAerialSequence.py

```

1  import argparse
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.patches as patches
5  import os
6  from SubtractDominantMotion import SubtractDominantMotion
7
8  # write your script here, we recommend the above libraries
9  # Set up iterations, threshold, and tolerance
10 parser = argparse.ArgumentParser()
11 parser.add_argument('--num_iters', type=int, default=1e3, help='number of
12                        iterations of Lucas-Kanade')
13 parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold
14                        of Lucas-Kanade for terminating optimization')
15 parser.add_argument('--tolerance', type=float, default=0.2, help='binary
16                        threshold of intensity difference when computing the mask')
17 args = parser.parse_args()
18 num_iters = args.num_iters
19 threshold = args.threshold
20 tolerance = args.tolerance
21
22 # Load the video frames and Set up the reported frames
23 seq = np.load('../data/aerialseq.npy')
24 reported_frames = [29, 59, 89, 119]
25
26 # Apply Subtract Dominant Motion to every frames
27 frames = seq.shape[2]
28 for index in range(frames-1):
29     # Print out the frame index
30     print(f"frame = {index+1}")
31
32     # Load images
33     image1 = seq[:, :, index]
34     image2 = seq[:, :, index+1]
35
36     # Apply Subtract Dominant Motion
37     mask = SubtractDominantMotion(image1, image2, threshold, num_iters,
38                                   tolerance)
39
40     # Extract the moving objects
41     target = np.where(mask == 1)
42
43     # Plot the reported frames
44     if index in reported_frames:
45         # Plot frames
46         fig = plt.figure()

```

```

47 ax = fig.add_subplot(1, 1, 1)
48 plt.imshow(image1, cmap = 'gray')
49 plt.axis('off')
50
51 # Plot the moving objects
52 plt.plot(target[1], target[0], 'bo', markersize = 2)
53
54 # Save the reported figures
55 plots_dir = '../plots/'
56 if not os.path.exists(plots_dir):
57     os.makedirs(plots_dir)
58 fig.savefig('../plots/Aerial_f' + str(index+1) + '.png',
59             bbox_inches = 'tight', pad_inches = 0)

```