Computer Vision (16-720 B): Homework 3

Lucas-Kanade Tracking

Name - Saharsh Agarwal
AndrewID - saharsh2

Question 1.1

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

Answer : For the given question, **p** is pure translation. Thus, known -

$$W(x;p) = x + p \tag{1}$$

x is position (X,Y) and constant for calculation at a particular position. p is the offset (p1,p2). Hence,

$$\frac{\partial W(x;p)}{\partial p^T} = \frac{\partial (x+p)}{\partial p^T} = \begin{bmatrix} \frac{\partial p1}{\partial p1} & \frac{\partial p1}{\partial p2} \\ \frac{\partial p2}{\partial p1} & \frac{\partial p2}{\partial p2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2}$$

$\frac{\partial W(x;p)}{\partial p^T}$ is Identity matrix of dimension 2 (Ans).

0.2 What is A and b ?

Answer : Using equation 3 and 4 from the write-up provided, where x'_j is $x_j + p$, $\nabla I_{t+1}(x'_j)$ is $\frac{\partial I_{t+1}(x'_j))}{\partial x'^T_j}$. j ∈ [1,N]

$$A = \begin{bmatrix} \nabla I_{t+1}(x'_1) & . & . & . & 0^T \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 0^T. & . & . & . & \nabla I_{t+1}(x'_N) \end{bmatrix} \begin{bmatrix} \frac{\partial W(x_1;p)}{\partial p^T} \\ . \\ . \\ . \\ \frac{\partial W(x_N;p)}{\partial p^T} \end{bmatrix} \tag{3}$$

$$b = \begin{bmatrix} I_{t+1}(x'_1) - I_t(x_1) \\ . \\ . \\ . \\ I_{t+1}(x'_N) - I_t(x_N) \end{bmatrix} \tag{4}$$

0.3 What conditions must A^TA meet so that a unique solution to Δp can be found ?

Answer : A^TA is a square matrix. To find a unique solution to Δp, A^TA should be invertible, i.e., its determinant is not zero or rows of the matrix are linearly independent.

Question 1.2

```python
import numpy as np
from scipy.interpolate import RectBivariateSpline

def LucasKanade(It, It1, rect, threshold, num_iters, p0=np.zeros(2)):

    # Put your implementation here
    p = p0
    xl,yl,xr,yr = rect # l is left and r is right
    i = 0

    # Adding 1 to include last values
    #RectBivariateSpline - X and Y length ke image patch pe spline/ curve
    #                       fit karna taaki interpolate kar sake
    # .ev - evaluates the value at spline -> derivatives can also be checked

    # current image
    X1 = np.arange(0, It1.shape[0], 1)
    Y1 = np.arange(0, It1.shape[1], 1)
    It_real_spline = RectBivariateSpline(X1,Y1,It1)
    ### print("Real",len(X1),len(Y1)) - 240 by 320

    # Template - t
    X = np.arange(0, It.shape[0], 1)
    Y = np.arange(0, It.shape[1], 1)
    It_temp_spline = RectBivariateSpline(X,Y,It)
    ## print("Template",len(X),len(Y))

    del_p = threshold + 1 # random value just so that begining above threshold

    while (i < num_iters and del_p > threshold):
        # Moving through Real Image and Meshing
        xreal_patchlen = np.arange(xl + p[0], xr + p[0] + 0.01)
        yreal_patchlen = np.arange(yl + p[1], yr + p[1] + 0.01)
        xreal, yreal = np.meshgrid(xreal_patchlen, yreal_patchlen)
        patch_real = It_real_spline.ev(yreal, xreal)
        #print(i, "Real Patch", patch_real.shape)

        # Meshing of Template - t
        xtemp_patchlen = np.arange(xl, xr + 0.01)
        ytemp_patchlen = np.arange(yl, yr + 0.01)
        xtemp, ytemp = np.meshgrid(xtemp_patchlen, ytemp_patchlen)
        template = It_temp_spline.ev(ytemp, xtemp) #spline values for all (y,x)
        ## print(template.shape, "temp shape")

        #Derivatives
        dIt1x = It_real_spline.ev(yreal, xreal, 0, 1).flatten()
        dIt1y = It_real_spline.ev(yreal, xreal, 1, 0).flatten()

        A = np.hstack((np.expand_dims(dIt1x, axis = 1), np.expand_dims(dIt1y, axis = 1)))
        b = template - patch_real
        dp = np.linalg.lstsq(A,b.flatten(),rcond=None)[0]
        p[0] = p[0] + dp[0]
        p[1] = p[1] + dp[1]
        del_p = np.sqrt(dp[0]**2 + dp[1]**2)
        i = i + 1

    return p
    """
    :param It: template image
```

Figure 1: Code for **LucasKanade.py**

Question 1.3

```python
import argparse
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from LucasKanade import LucasKanade

# write your script here, we recommend the above libraries for making your animation

parser = argparse.ArgumentParser()
parser.add_argument('--num_iters', type=int, default=1e4, help='number of iterations of Lucas-Kanade')
parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold of Lucas-Kanade for terminating optimization')
args = parser.parse_args()
num_iters = args.num_iters
threshold = args.threshold

seq = np.load("../data/carseq.npy") #2Dimage - 3rd dimension is time (240,320) in 415 frames
rect = [59, 116, 145, 151] #bounding box

rects = []
rects.append(rect)

for i in range(seq.shape[2]-1):
    template = seq[:,:,i]
    rect = rects[i]
    p = LucasKanade(template, seq[:,:,i+1], rect, threshold, num_iters, p0=np.zeros(2))
    frame_rect = np.asarray([rect[0]+p[0], rect[1]+p[1],rect[2]+p[0], rect[3]+p[1]])
    rects.append(frame_rect)

    width = rect[2]-rect[0]
    height = rect[3]-rect[1]
    ### print("patch shape", width,height)

    #Visualization
    if (i == 0 or i == 99 or i == 199 or i==299 or i == 399):
        fig, ax = plt.subplots()
        ax.imshow(seq[:,:,i], cmap= 'gray')
        rectan = patches.Rectangle((rect[0], rect[1]), width+1, height+1, linewidth=1, edgecolor='r', facecolor='none')
        ax.add_patch(rectan)
        plt.show()

rects = np.asarray(rects)
print(rects.shape)
with open('carseqrects.npy', 'wb') as f:
    np.save(f,rects)
```

Figure 2: Code for **testCarSequence.py** - Car Tracking with Lucas Kanade

carseqrects.npy submitted with the files.

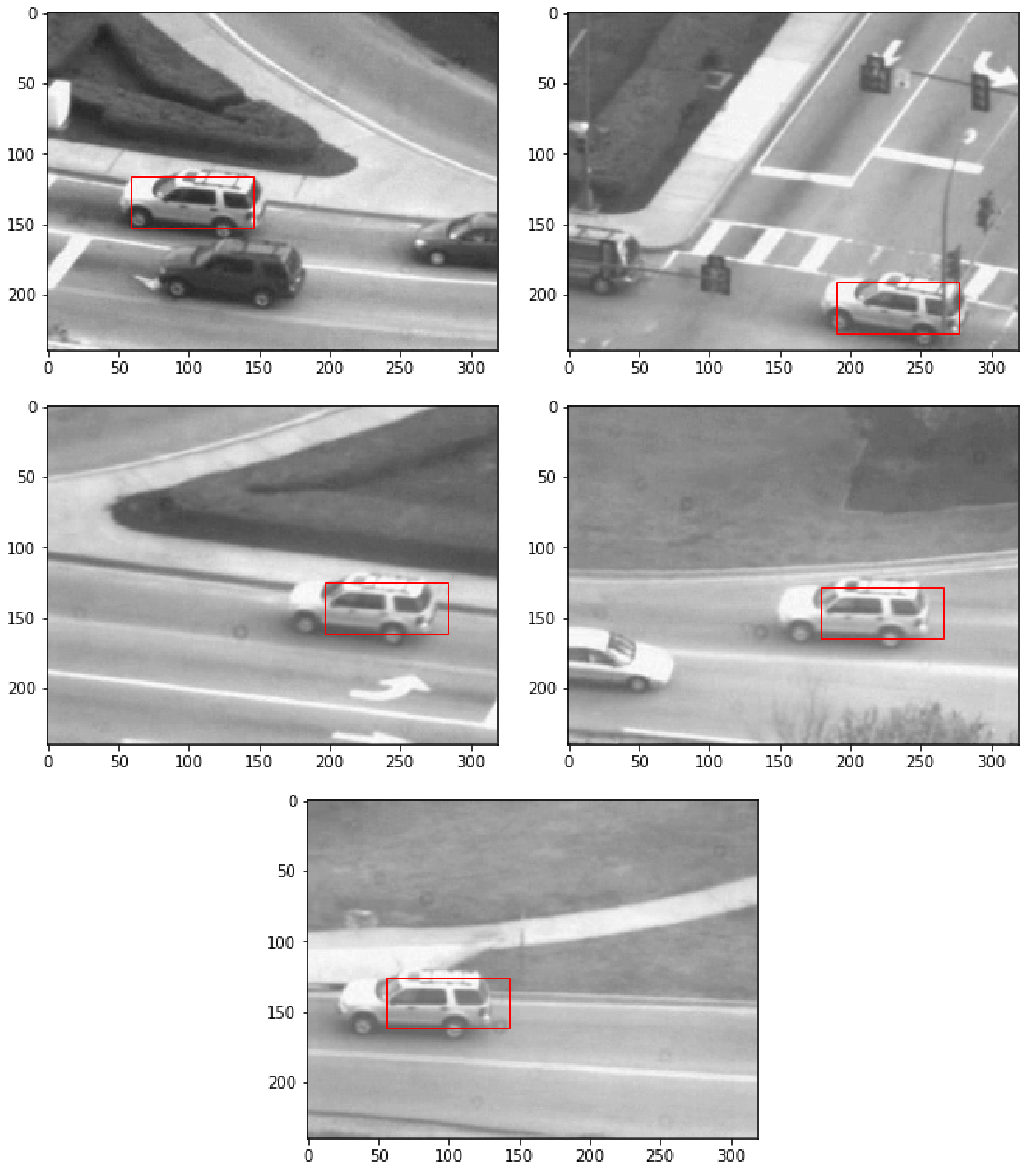


Figure 3: Car Tracking using Lucas Kanade (for pure translation)

Figure 4: Code for **testGirlSequence.py** - Tracking Girl with Lucas Kanade

```python
import argparse
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from LucasKanade import LucasKanade
# write your script here, we recommend the above libraries for making your animation

parser = argparse.ArgumentParser()
parser.add_argument('--num_iters', type=int, default=1e4, help='number of iterations of Lucas-Kanade')
parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold of Lucas-Kanade for terminating optimization')
args = parser.parse_args()
num_iters = args.num_iters
threshold = args.threshold

seq = np.load("../data/girlseq.npy")
rect = [280, 152, 330, 318]

rects = []
rects.append(rect)

for i in range(seq.shape[2]-1):
    template = seq[:,:,i]
    rect = rects[i]
    p = LucasKanade(template, seq[:,:,i+1], rect, threshold, num_iters, p0=np.zeros(2))
    frame_rect = np.asarray([rect[0]+p[0], rect[1]+p[1],rect[2]+p[0], rect[3]+p[1]])
    rects.append(frame_rect)

    width = rect[2]-rect[0]
    height = rect[3]-rect[1]
    ### print("patch shape", width,height)

    #Visualization
    if (i == 0 or i == 19 or i == 39 or i==59 or i == 79):
        fig, ax = plt.subplots()
        ax.imshow(seq[:,:,i], cmap= 'gray')
        rect = patches.Rectangle((rect[0], rect[1]), width+1, height+1, linewidth=1, edgecolor='r', facecolor='none')
        ax.add_patch(rect)
        plt.show()

rects = np.asarray(rects)
print(rects.shape)
with open('girlseqrects.npy', 'wb') as f:
    np.save(f,rects)
```

girlseqrects.npy submitted with the files.

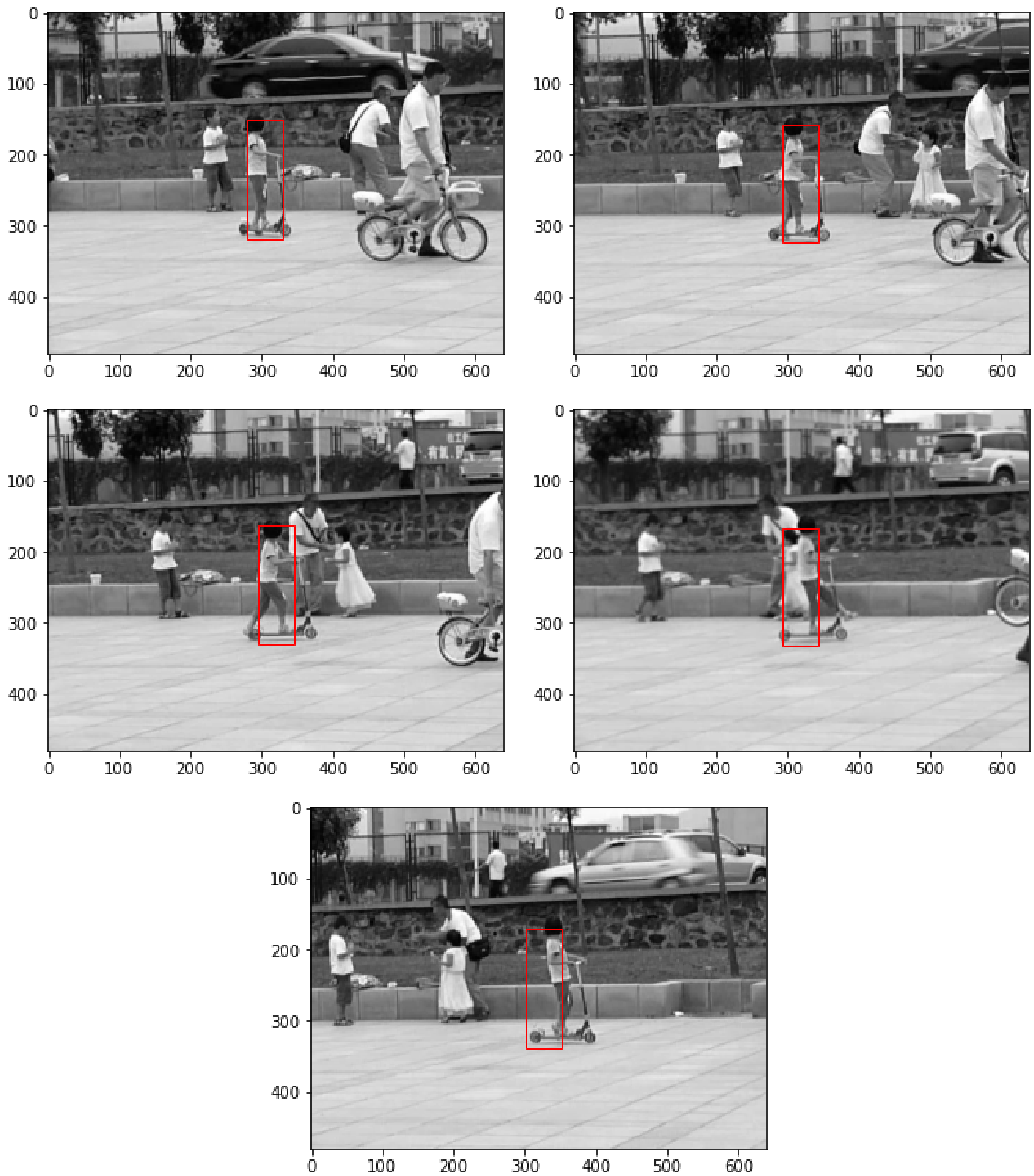


Figure 5: Tracking Girl using Lucas Kanade (for pure translation)

Question 1.4

Figure 6: Code for **testCarSequenceWithTemplateCorrection.py** - Tracking Car with Lucas Kanade (avoiding template drifting)

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from LucasKanade import LucasKanade

parser = argparse.ArgumentParser()
parser.add_argument('--num_iters', type=int, default=1e4, help='number of iterations of Lucas-Kanade')
parser.add_argument('--threshold', type=float, default=1e-2, help='dp threshold of Lucas-Kanade for terminating optimization')
parser.add_argument('--template_threshold', type=float, default=0.005, help='threshold for determining whether to update template')
args = parser.parse_args()
num_iters = args.num_iters
threshold = args.threshold
template_threshold = args.template_threshold

seq = np.load("../data/carseq.npy")
rects_not_corr = np.load("../code/carseqrects.npy")
rect = [59, 116, 145, 151]
corr_rect = []
corr_rect.append(rect)
pstar = None
p = None
start_frame = seq[:,:,0]
thresh = 0.05

for i in range(seq.shape[2]-1):
    next_frame = seq[:,:,i+1]
    p = LucasKanade(start_frame, next_frame, rect, threshold, num_iters)

    frame_rect = np.asarray([rect[0]+p[0], rect[1]+p[1],rect[2]+p[0], rect[3]+p[1]])
    corr_rect.append(frame_rect)

    width = frame_rect[2]-frame_rect[0]
    height = frame_rect[3]-frame_rect[1]
    ### print("patch shape", width,height)

    temp_img = next_frame

    #Visualization
    if i in [0,99,199,299,399]:
        fig, ax = plt.subplots()
        ax.imshow(temp_img, cmap= 'gray')
        rectan = patches.Rectangle((frame_rect[0], frame_rect[1]), width+1, height+1, linewidth=2, edgecolor='r', facecolor='none')
        a,b,c,d = rects_not_corr[i+1]
        rectancorr = patches.Rectangle((a, b), c-a+1, d-b+1, linewidth=1, edgecolor='b', facecolor='none')
        ax.add_patch(rectan)
        ax.add_patch(rectancorr)
        plt.show()

    if pstar is None or np.sum((pstar-p)**2)**0.5 < thresh:
        start_frame = next_frame
        rect = frame_rect
        pstar = np.copy(p)

corr_rect = np.asarray(corr_rect)
with open('carseqrects-wcrt.npy', 'wb') as f:
    np.save(f,corr_rect)
```

carseqrects-wcrt.npy submitted with the files.

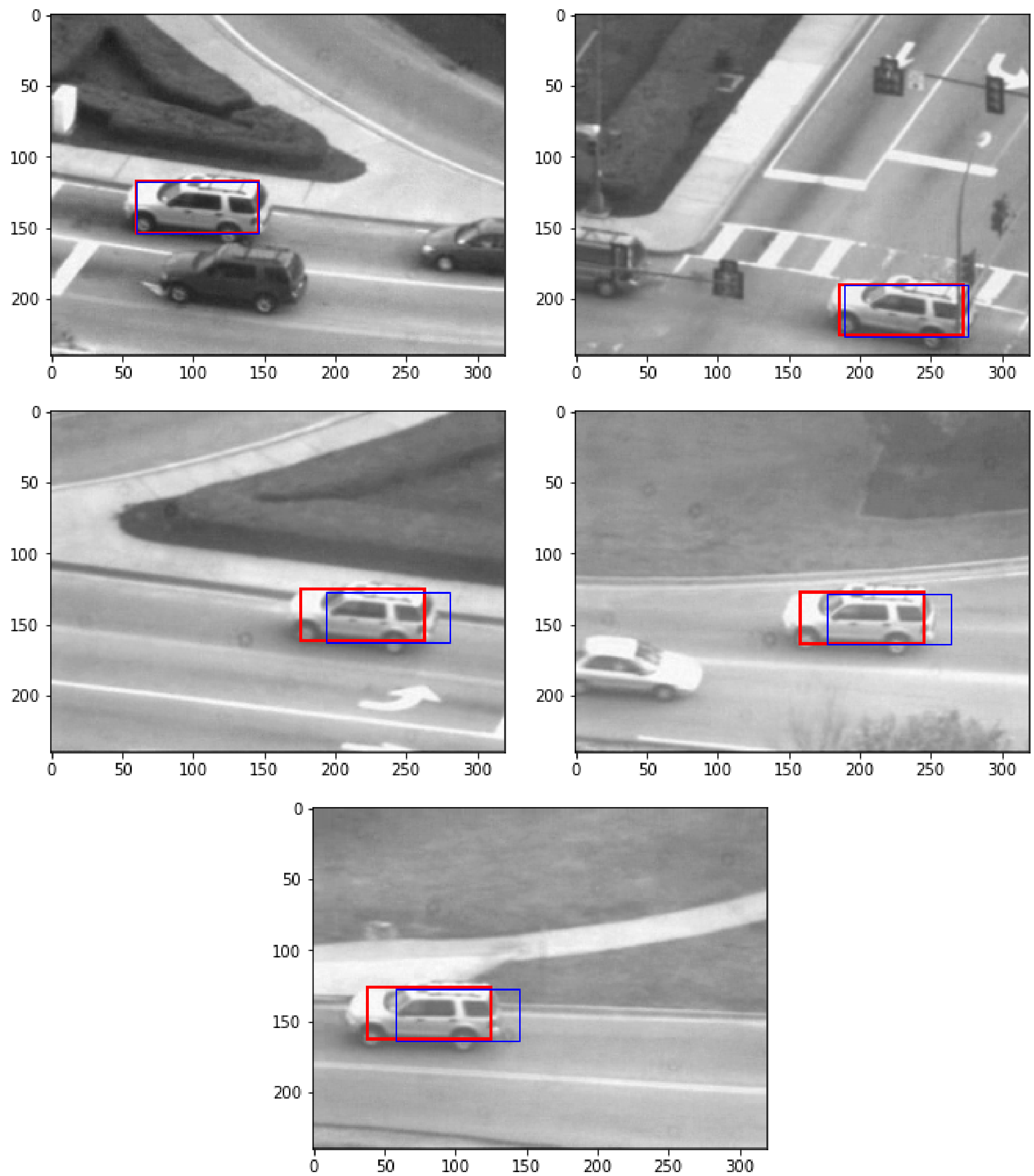


Figure 7: Tracking Car using Lucas Kanade (for pure translation) - comparing results after removal of template drift (RED)

The Blue Rectangles are from the baseline tracker in Question 1.3. Red Rectangle is the improvement post the removal of template drifting.

Figure 8: Code for **testGirlSequenceWithTemplateCorrection.py** - Tracking Girl with Lucas Kanade (avoiding template drifting)

```python
import matplotlib.patches as patches
from LucasKanade import LucasKanade

# write your script here, we recommend the above libraries for making your animation

parser = argparse.ArgumentParser()
parser.add_argument('--num_iters', type=int, default=1e4, help='number of iterations of Lucas-Kanade')
parser.add_argument('--threshold', type=float, default=0.1, help='dp threshold of Lucas-Kanade for terminating optimization')
parser.add_argument('--template_threshold', type=float, default=0, help='threshold for determining whether to update template')
args = parser.parse_args()
num_iters = args.num_iters
threshold = args.threshold
template_threshold = args.template_threshold

seq = np.load("../data/girlseq.npy")
rect = [280, 152, 330, 318]

rects_not_corr = np.load("../code/girlseqrects.npy")
corr_rect = []
corr_rect.append(rect)
pstar = None
p = None

start_frame = seq[:,:,0]

for i in range(seq.shape[2]-1):
    next_frame = seq[:,:,i+1]
    p = LucasKanade(start_frame, next_frame, rect, threshold, num_iters)

    frame_rect = np.asarray([rect[0]+p[0], rect[1]+p[1],rect[2]+p[0], rect[3]+p[1]])
    corr_rect.append(frame_rect)

    width = frame_rect[2]-frame_rect[0]
    height = frame_rect[3]-frame_rect[1]
    ### print("patch shape", width,height)

    temp_img = next_frame

    #Visualization
    if i in [0,19,39,59,79]:
        print(i)
        fig, ax = plt.subplots()
        ax.imshow(temp_img, cmap= 'gray')
        rectan = patches.Rectangle((frame_rect[0], frame_rect[1]), width+1, height+1, linewidth=2, edgecolor='r', facecolor='none')
        a,b,c,d = rects_not_corr[i+1]
        rectancorr = patches.Rectangle((a, b), c-a+1, d-b+1, linewidth=1, edgecolor='b', facecolor='none')
        ax.add_patch(rectan)
        ax.add_patch(rectancorr)
        plt.show()

    if pstar is None or np.sum((pstar-p)**2)**0.5 < template_threshold:
        print("############################################################")
        start_frame = next_frame
        rect = frame_rect
        pstar = np.copy(p)

corr_rect = np.asarray(corr_rect)
with open('girlseqrects-wcrt.npy', 'wb') as f:
    np.save(f,corr_rect)
```

girlseqrects-wcrt.npy submitted with the files.

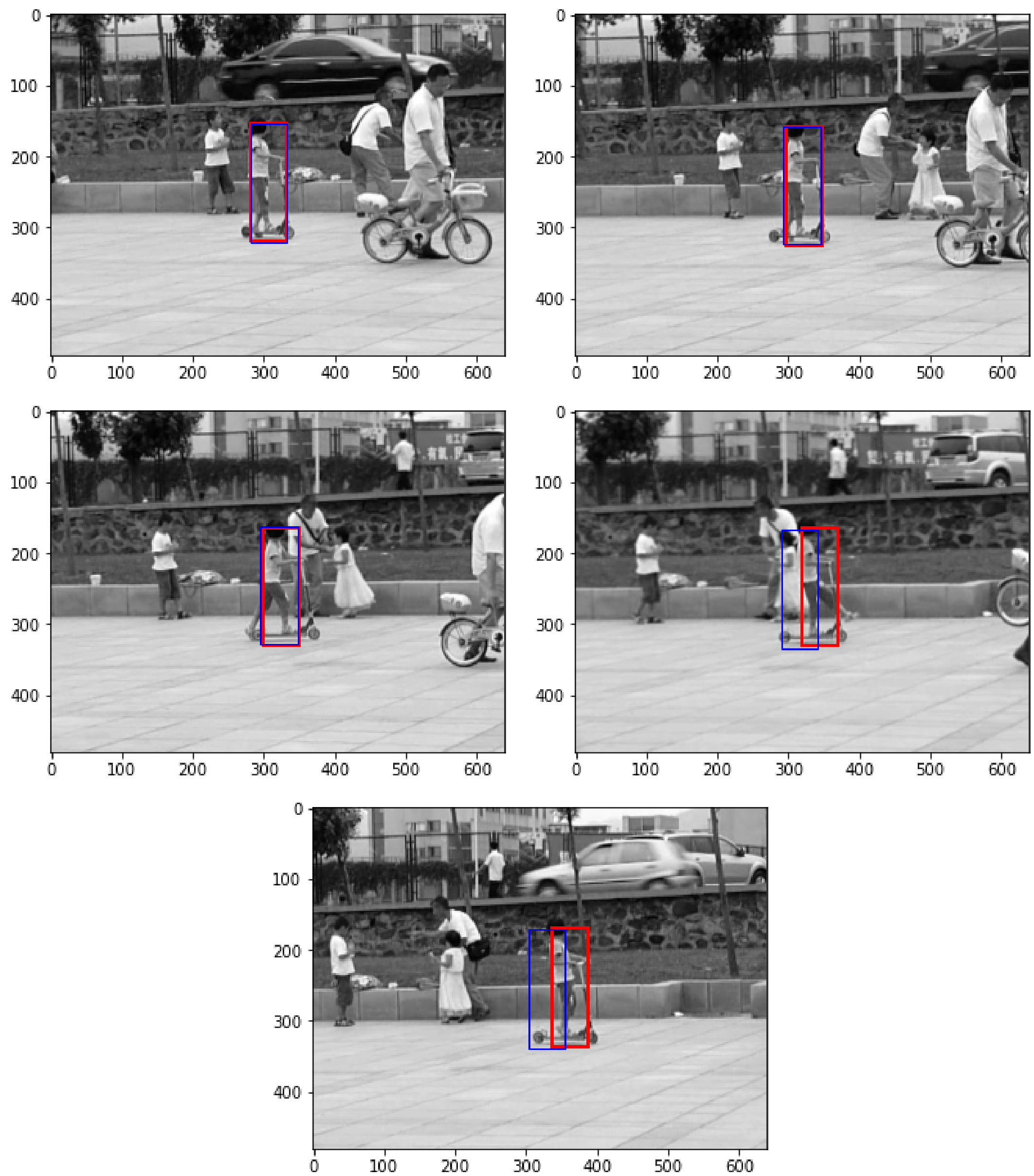


Figure 9: Tracking Girl using Lucas Kanade (for pure translation) - comparing results after removal of template drift (RED)

The Blue Rectangles are from the baseline tracker in Question 1.3. Red Rectangle is the improvement post the removal of template drifting.

Question 2.1

Figure 10: Code for **LucasKanadeAffine.py**

```python
def LucasKanadeAffine(It, It1, threshold, num_iters):

    # put your implementation here
    M = np.array([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0]]) #add last [0.0, 0.0, 1.0]

    i = 0
    xl,yl,xr,yr = 0,0,It.shape[0]-1,It.shape[1]-1
    del_p = threshold+1

    # current image
    X1 = np.arange(0, It1.shape[0], 1)
    Y1 = np.arange(0, It1.shape[1], 1)
    It_real_spline = RectBivariateSpline(X1,Y1,It1)
    ### print("Real",len(X1),len(Y1)) - 240 by 320

    # Template - t
    X = np.arange(0, It.shape[0], 1)
    Y = np.arange(0, It.shape[1], 1)
    It_temp_spline = RectBivariateSpline(X,Y,It)

    while (i<num_iters and del_p >= threshold):

        # Meshing of Template - threshold
        xtemp_patchlen = np.arange(xl, xr + 0.01)
        ytemp_patchlen = np.arange(yl, yr + 0.01)
        xtemp, ytemp = np.meshgrid(xtemp_patchlen, ytemp_patchlen)
        template = It_temp_spline.ev(ytemp, xtemp)

        ### Warped
        xt1 = M[0,0]*xtemp + M[0,1]*ytemp + M[0,2]
        yt1 = M[1,0]*xtemp + M[1,1]*ytemp + M[1,2]

        ### For inside the frame
        idx = (xt1 > 0) & (xt1 < It.shape[1]) & (yt1>0) & (yt1<It.shape[0])

        xt1 = xt1[idx]
        yt1 = yt1[idx]
        xtemp = xtemp[idx]
        ytemp = ytemp[idx]
        real_patch = It_real_spline.ev(yt1,xt1)

        #Derivatives
        dIt1x = It_real_spline.ev(yt1, xt1, 0, 1).flatten()
        dIt1y = It_real_spline.ev(yt1, xt1, 1, 0).flatten()

        dIt1x = np.expand_dims(dIt1x, axis = 1)
        dIt1y = np.expand_dims(dIt1y, axis = 1)

        ## try
        xt1 = np.expand_dims(xtemp.flatten(), axis = 1)
        yt1 = np.expand_dims(ytemp.flatten(), axis = 1)

        #xt1 = np.expand_dims(xt1.flatten(), axis = 1)
        #yt1 = np.expand_dims(yt1.flatten(), axis = 1)

        A = np.hstack((dIt1x*xt1, dIt1x*yt1, dIt1x, dIt1y*xt1, dIt1y*yt1, dIt1y))
        b = template[idx] - real_patch

        dp = np.linalg.lstsq(A,b.flatten(),rcond=None)[0]
        M = M + np.reshape(dp, (2,3))

        del_p = np.linalg.norm(dp)
        i = i+1
        #print(i, M)
        print(i, "LKA")

    return M
```

Question 2.2

Figure 11: Code for **SubtractDominantMotion.py**

```
def SubtractDominantMotion(image1, image2, threshold, num_iters, tolerance):

    # put your implementation here
    mask = None
    mask = np.zeros(image1.shape, dtype=bool)

    M = lka.LucasKanadeAffine(image1,image2,threshold,num_iters)

    #M = ica.InverseCompositionAffine(image1,image2,threshold*200,num_iters)

    M = np.vstack((M, np.asarray([[0,0,1]])))
    M = np.linalg.inv(M)

    spline_image1 = RectBivariateSpline(np.arange(image1.shape[0]), np.arange(image1.shape[1]), image1)
    spline_image2 = RectBivariateSpline(np.arange(image2.shape[0]), np.arange(image2.shape[1]), image2)

    x = np.arange(0, image2.shape[1])
    y = np.arange(0, image2.shape[0])
    xx, yy = np.meshgrid(x, y)
    X = M[0, 0] * xx + M[0, 1] * yy + M[0, 2]
    Y = M[1, 0] * xx + M[1, 1] * yy + M[1, 2]

    invalid = (X < 0) | (X >= image1.shape[1]) | (Y < 0) & (Y >= image1.shape[0])

    I1 = spline_image1.ev(Y, X)
    I2 = spline_image2.ev(Y, X)
    I1[invalid] = 0
    I2[invalid] = 0
    #print(I1.shape)
    #plt.imshow(I1)
    #plt.show()

    # calculate the difference
    diff = np.absolute(I2 - I1)
    ind = (diff > tolerance) & (I2 != 0)
    mask[ind] = 1
    #print(mask.shape)

    '''
    aerial :
    mask = binary_erosion(mask, structure=np.eye(2), iterations=1)
    mask = binary_erosion(mask, structure=np.ones((1,3)), iterations=1)
    st = np.asarray([[1,1,1],[1,1,1],[1,1,1]])
    mask = binary_dilation(mask, structure=st, iterations = 2)
    '''
    #mask = binary_erosion(mask, structure=np.eye(2), iterations=1)
    #mask = binary_erosion(mask, structure=np.ones((1,3)), iterations=1)
    st = np.asarray([[1,1,1],[1,1,1],[1,1,1]])
    mask = binary_dilation(mask, structure=st, iterations = 1)
    #plt.imshow(mask)
    #plt.show()

    return mask
```

Another method was also tried but got better results with this. The code for the same is enclosed in the file submitted (commented out).

Question 2.3

Figure 12: Code for **testAntSequence.py**

```python
import argparse
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import time

import SubtractDominantMotion

# write your script here, we recommend the above libraries for making your animation

parser = argparse.ArgumentParser()
parser.add_argument('--num_iters', type=int, default=1e2, help='number of iterations of Lucas-Kanade')
parser.add_argument('--threshold', type=float, default=0.001, help='dp threshold of Lucas-Kanade for terminating optimization')
parser.add_argument('--tolerance', type=float, default=0.05, help='binary threshold of intensity difference when computing the mask')
args = parser.parse_args()
num_iters = args.num_iters
threshold = args.threshold
tolerance = args.tolerance

seq = np.load('../data/antseq.npy')

frame = seq[:,:,0]
start = time.time()
for i in range(seq.shape[2]-1):
    print(i)
    next_frame = seq[:,:,i+1]
    mask = SubtractDominantMotion.SubtractDominantMotion(frame,next_frame,threshold,num_iters,tolerance)
    temp_img = np.zeros((next_frame.shape[0],next_frame.shape[1],3))
    for kk in range(3):
        temp_img[:,:,kk] = next_frame

    temp_img[:,:,2][mask==1] = 1  ### to make blue


    if i in [29,59,89,119]:
        fig, ax = plt.subplots()
        ax.imshow(temp_img)
        plt.show()

    frame = next_frame
print("Time Taken - ", time.time()-start)
```

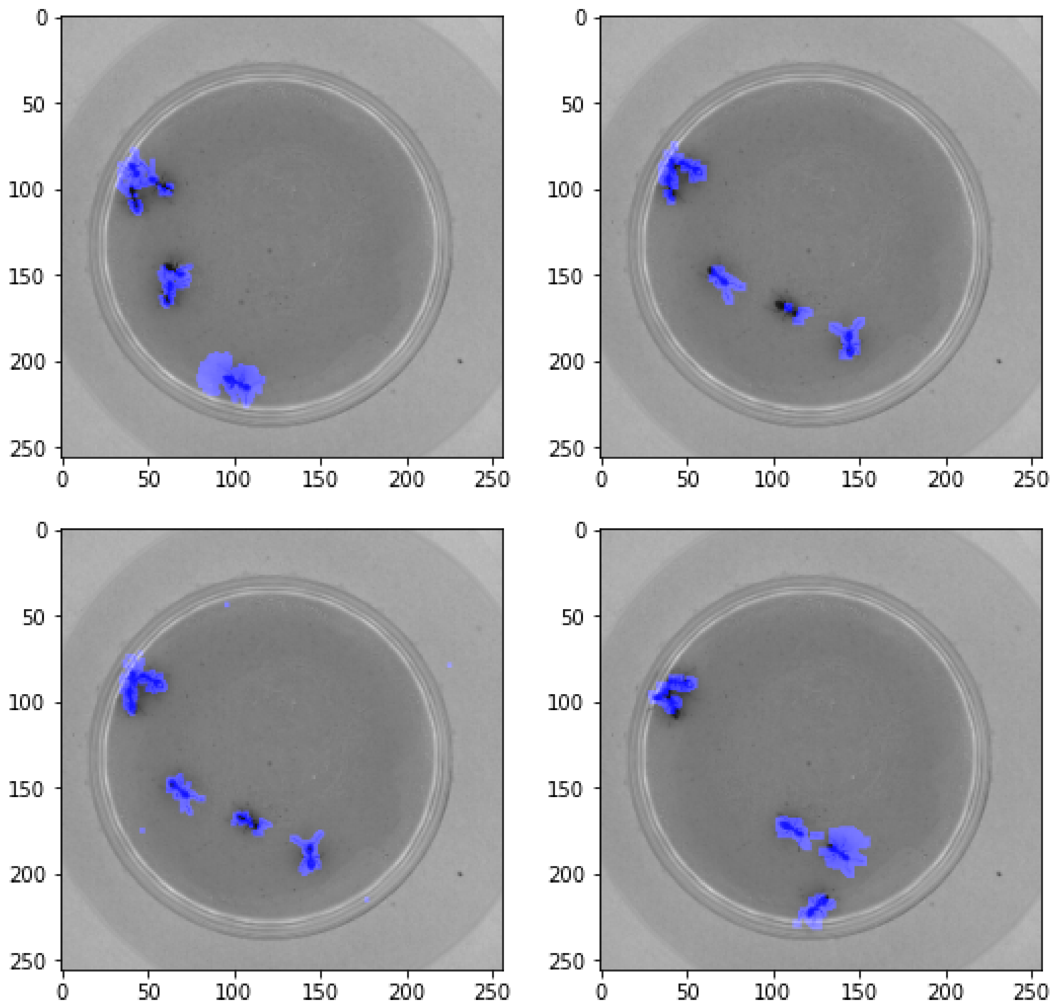


Figure 13: Tracking Ants

Figure 14: Code for **testAerialSequence.py**

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import time

import SubtractDominantMotion

# write your script here, we recommend the above libraries for making your animation

parser = argparse.ArgumentParser()
parser.add_argument('--num_iters', type=int, default=1e3, help='number of iterations of Lucas-Kanade')
parser.add_argument('--threshold', type=float, default=0.01, help='dp threshold of Lucas-Kanade for terminating optimization')
parser.add_argument('--tolerance', type=float, default=0.25, help='binary threshold of intensity difference when computing the mask')
args = parser.parse_args()
num_iters = args.num_iters
threshold = args.threshold
tolerance = args.tolerance

seq = np.load('../data/aerialseq.npy')

frame = seq[:,:,0]
start = time.time()
for i in range(seq.shape[2]-1):
    print(i)
    ## try thing in lucas affine also
    next_frame = seq[:,:,i+1]
    mask = SubtractDominantMotion.SubtractDominantMotion(frame,next_frame,threshold,num_iters,tolerance)
    temp_img = np.zeros((next_frame.shape[0],next_frame.shape[1],3))
    for kk in range(3):
        temp_img[:,:,kk] = next_frame
    #print(temp_img)
    temp_img[:,:,2][mask==1] = 1


    if i in [29,59,89,119]:
        fig, ax = plt.subplots()
        ax.imshow(temp_img)
        plt.show()

    frame = next_frame
print("Time taken -", time.time()-start)
```

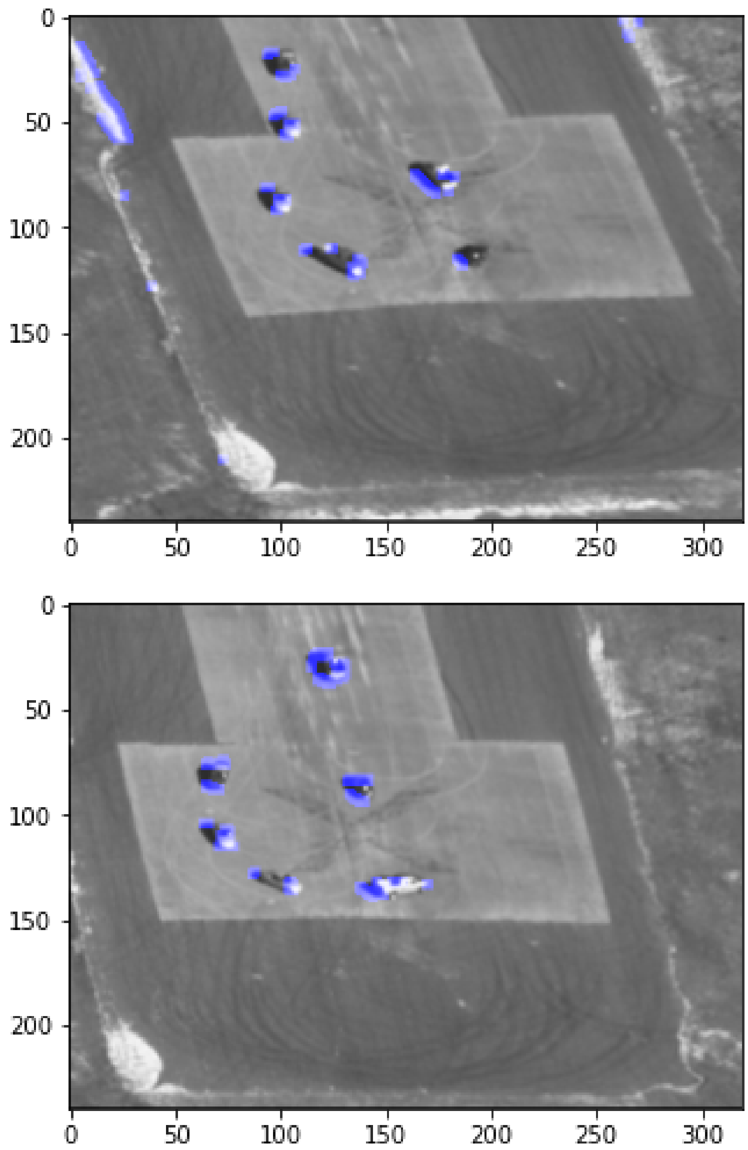


Figure 15: Tracking Aerial

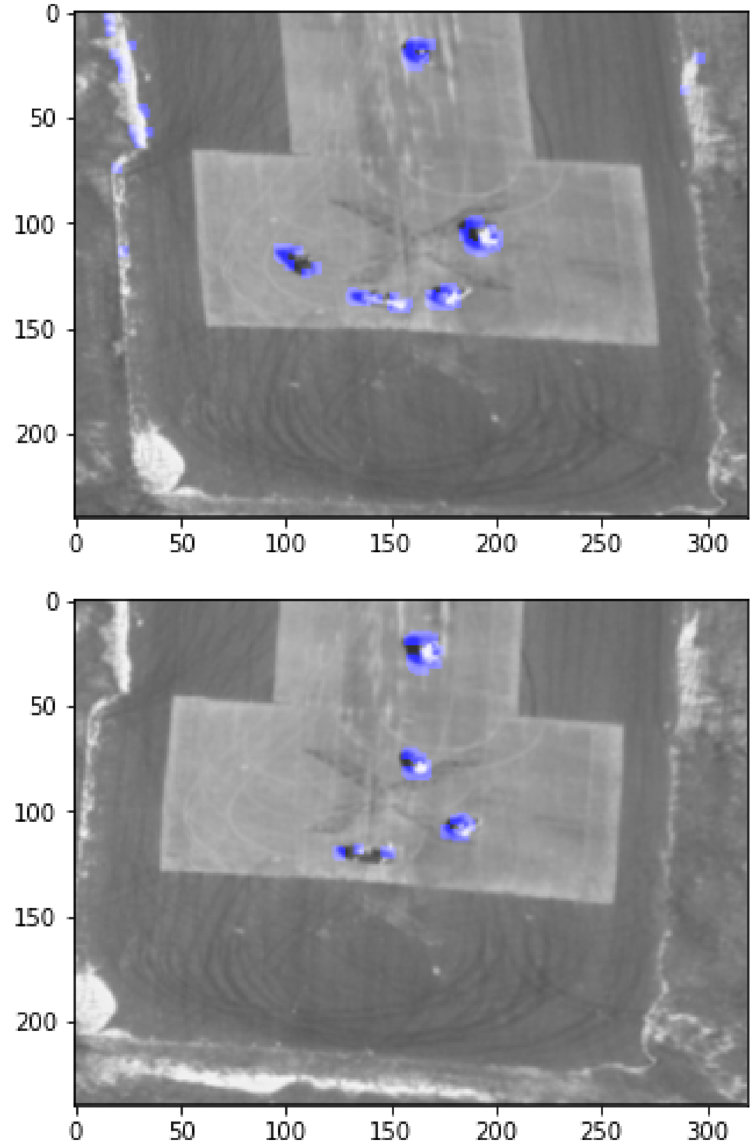


Figure 16: Tracking Aerial

Question 3.1

```
import numpy as np
from scipy.interpolate import RectBivariateSpline

def InverseCompositionAffine(It, It1, threshold, num_iters):

    # put your implementation here
    M = np.array([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0]])

    i = 0
    xl,yl,xr,yr = 0,0,It.shape[0]-1,It.shape[1]-1
    del_p = threshold+1

    # current image
    X1 = np.arange(0, It1.shape[0], 1)
    Y1 = np.arange(0, It1.shape[1], 1)
    It_real_spline = RectBivariateSpline(X1,Y1,It1)
    ### print("Real",len(X1),len(Y1)) - 240 by 320

    # Template - t
    X = np.arange(0, It.shape[0], 1)
    Y = np.arange(0, It.shape[1], 1)
    It_temp_spline = RectBivariateSpline(X,Y,It)

    # Meshing of Template - threshold
    xtemp_patchlen = np.arange(xl, xr + 0.01)
    ytemp_patchlen = np.arange(yl, yr + 0.01)
    xtemp, ytemp = np.meshgrid(xtemp_patchlen, ytemp_patchlen)

    template = It_temp_spline.ev(ytemp, xtemp)

    #Derivatives
    dItx = It_temp_spline.ev(ytemp, xtemp, 0, 1).flatten()
    dIty = It_temp_spline.ev(ytemp, xtemp, 1, 0).flatten()

    dItx = np.expand_dims(dItx, axis = 1)
    dIty = np.expand_dims(dIty, axis = 1)

    xt = np.expand_dims(xtemp.flatten(), axis = 1)
    yt = np.expand_dims(ytemp.flatten(), axis = 1)

    A = np.hstack((dItx*xt, dItx*yt, dItx, dIty*xt, dIty*yt, dIty))
    H = A.T@A

    while (i<num_iters and del_p >= threshold):
        #print(del_p, threshold)
        xreal, yreal = np.meshgrid(xtemp_patchlen, ytemp_patchlen)
        ### For inside the frame
        idx = (xreal > 0) & (xreal < It.shape[1]) & (yreal>0) & (yreal<It.shape[0])
        ### Warped
        xreal = xreal[idx]
        yreal = yreal[idx]
        xt1 = M[0,0]*xreal + M[0,1]*yreal + M[0,2]
        yt1 = M[1,0]*xreal + M[1,1]*yreal + M[1,2]

        real_patch = It_real_spline.ev(yt1,xt1)

        ## try
        #xt1 = np.expand_dims(xt1.flatten(), axis = 1)
        #yt1 = np.expand_dims(yt1.flatten(), axis = 1)

        b = template[idx] - real_patch

        dp = np.linalg.pinv(H)@A[idx.flatten()].T@b.flatten()
        dm = np.reshape(dp, (2,3))
        dm = dm + np.asarray([[1,0,0],[0,1,0]])

        M = np.vstack((M,np.asarray([[0,0,1]])))
        dm = np.vstack((dm,np.asarray([[0,0,1]])))

        M =( M @ np.linalg.pinv(dm))[:2,:]
        del_p = np.linalg.norm(dp)
        i = i+1

        #print(del_p, "LK-ICA")

    return M
```

Figure 17: Code for **InverseCompositionAffine.py**

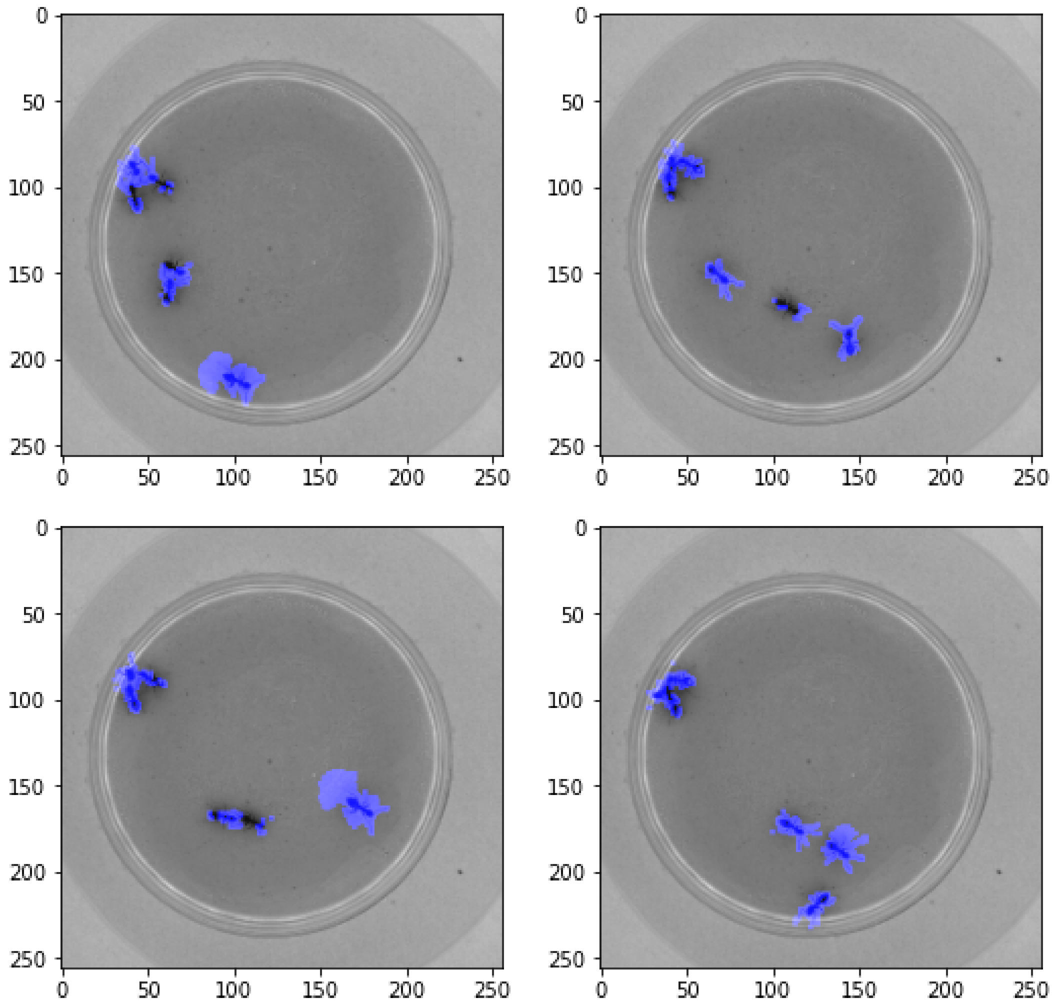
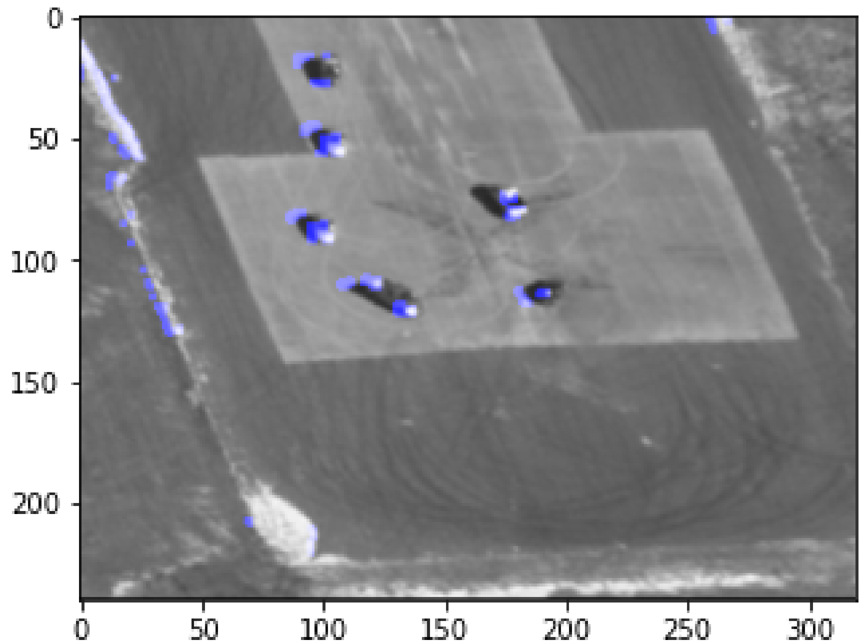
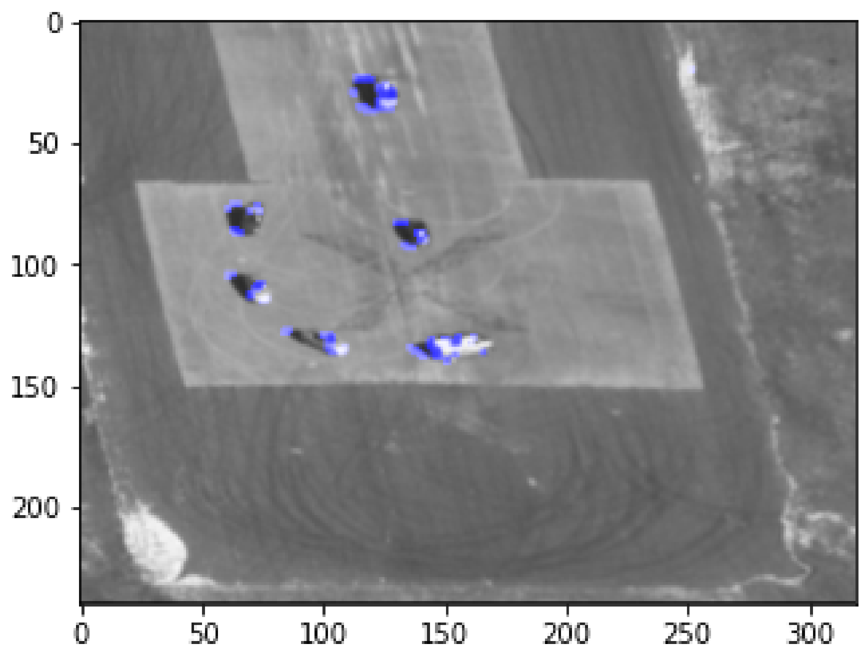


Figure 18: Tracking Ants - Inverse Compositional Affine Method


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

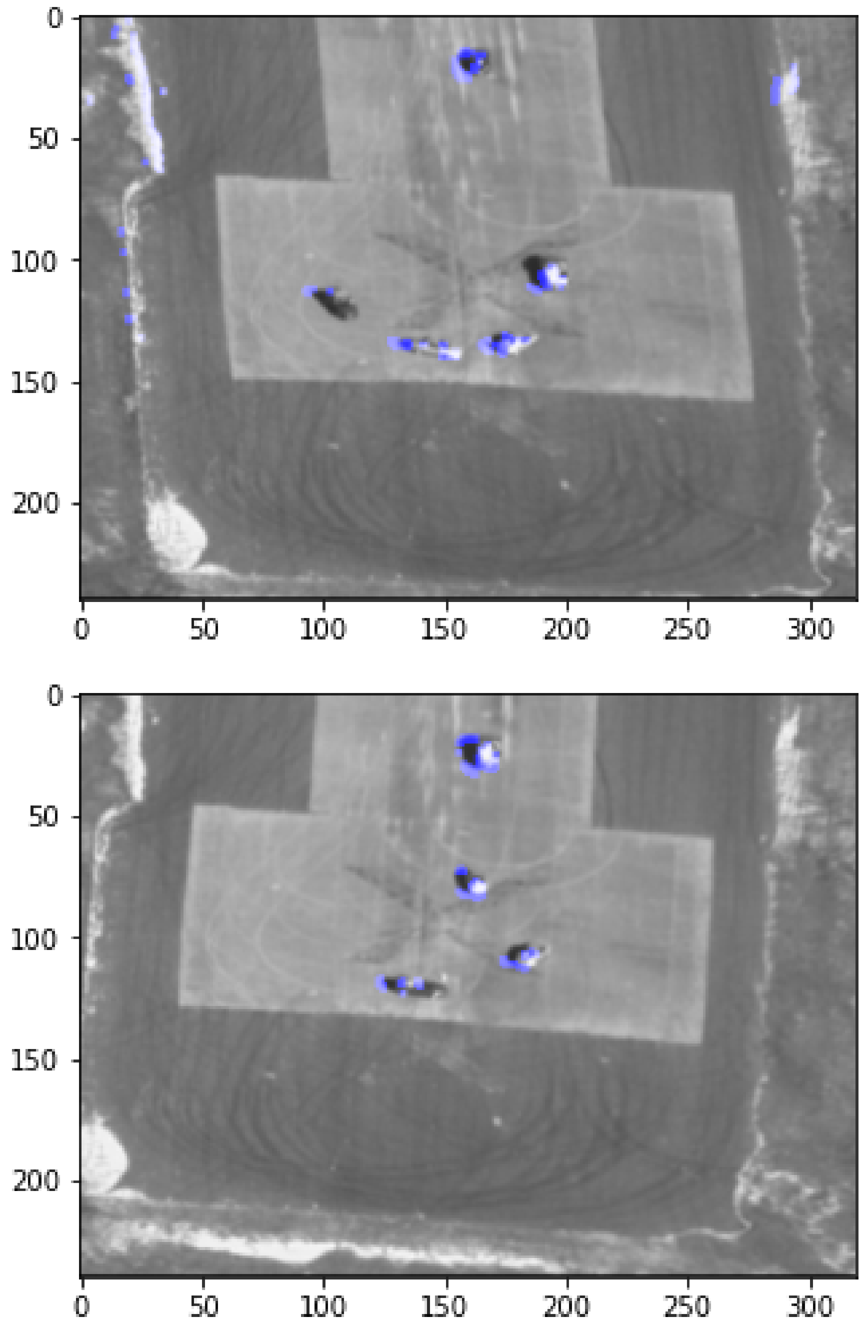


Figure 19: Tracking Aerial - Inverse Compositional Affine Method

LucasKanadeAffine -
For Ants : time = 60.4708s
For Aerial : time = 70.7588s
Inverse Compositional Affine -
For Ants : time = 15.8089s
For Aerial : time = 20.3608s

The runtime gain is very high. I have checked for different parameters as well. Also, for individual checks inside each frame was also evaluated. All the results suggested that Inverse Compositional Affine method is way faster.

In classical approach, p keeps changing thus making the matrix A dynamic. Thus, A has to be recalculated at each step. In the inverse compositional affine approach, A is not a function of p but of x. Thus, it is static and computed once. This saves time and makes the process efficient.