Initialization

Run the following code to import the modules you'll need. After your finish the assignment, **remember to run all cells** and save the note book to your local machine as a PDF for gradescope submission.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

Download data

In this section we will download the data and setup the paths.

```
# Download the data
if not os.path.exists('/content/carseq.npy'):
    !wget https://www.cs.cmu.edu/~deva/data/carseq.npy -O /content/carseq.npy
if not os.path.exists('/content/girlseq.npy'):
    !wget https://www.cs.cmu.edu/~deva/data/girlseq.npy -O /content/girlseq.npy
```

```
--2025-09-24 02:23:55--  https://www.cs.cmu.edu/~deva/data/carseq.npy
Resolving www.cs.cmu.edu (www.cs.cmu.edu)... 128.2.42.95
Connecting to www.cs.cmu.edu (www.cs.cmu.edu)|128.2.42.95|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 254976128 (243M)
Saving to: ‘/content/carseq.npy’

/content/carseq.npy 100%[===================>] 243.16M  3.41MB/s    in 72s

2025-09-24 02:25:08 (3.37 MB/s) - ‘/content/carseq.npy’ saved [254976128/254976128]

--2025-09-24 02:25:08--  https://www.cs.cmu.edu/~deva/data/girlseq.npy
Resolving www.cs.cmu.edu (www.cs.cmu.edu)... 128.2.42.95
Connecting to www.cs.cmu.edu (www.cs.cmu.edu)|128.2.42.95|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 27648128 (26M)
Saving to: ‘/content/girlseq.npy’

/content/girlseq.np 100%[===================>]  26.37M  3.59MB/s    in 7.4s

2025-09-24 02:25:16 (3.56 MB/s) - ‘/content/girlseq.npy’ saved [27648128/27648128]
```

Q2.1: Theory Questions (5 points)

Please refer to the handout for the detailed questions.

Q2.1.1: What is $\frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}$? (**Hint**: It should be a 2x2 matrix)

===== your answer here! =====

We know that W(x; p) is pure translational.

$W(x;p) = x + p$

W(x; p) =

$$\begin{bmatrix} x + p_x \\ y + p_y \end{bmatrix}$$

The derivate is a jacobian matrix defined as:

$\frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}$ =

$$\begin{bmatrix} \frac{\partial (x+p_x)}{\partial p_x} & \frac{\partial (x+p_x)}{\partial p_y} \\ \frac{\partial (y+p_y)}{\partial p_x} & \frac{\partial (y+p_y)}{\partial p_y} \end{bmatrix}$$

=

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

===== end of your answer =====

Q2.1.2: What is $\mathbf{A}$ and $\mathbf{b}$?

===== your answer here! =====

A is a (D, 2) matrix.

A = $\frac{\partial I_{t+1}(x')}{\partial x'^T} \frac{\partial W(x;p)}{\partial p^T}$

b is a (D, 1) vector.

b = $I_t(x) - I_{t+1}(x')$
b = $I_t(x) - I_{t+1}(W(x;p))$

===== end of your answer =====

Q2.1.3 What conditions must $\mathbf{A}^T\mathbf{A}$ meet so that a unique solution to $\Delta\mathbf{p}$ can be found?

===== your answer here! =====

$\mathbf{A}^T\mathbf{A}$ must be invertible which means the image gradients should have sufficient variation in both x and y directions.

===== end of your answer =====

Q2.2: Lucas-Kanade (20 points)

Make sure to comment your code and use proper names for your variables.

```
from scipy.interpolate import RectBivariateSpline
from numpy.linalg import lstsq

def LucasKanade(It, It1, rect, threshold, num_iters, p0=np.zeros(2)):
    """
    :param[np.array(H, W)] It   : Grayscale image at time t [float]
    :param[np.array(H, W)] It1  : Grayscale image at time t+1 [float]
    :param[np.array(4, 1)] rect : [x1 y1 x2 y2] coordinates of the rectangular template to extract from the image at time t,
                                  where [x1, y1] is the top-left, and [x2, y2] is the bottom-right. Note that coordinates
                                  [floats] that maybe fractional.
    :param[float] threshold     : If change in parameters is less than thresh, terminate the optimization
    :param[int] num_iters       : Maximum number of optimization iterations
    :param[np.array(2, 1)] p0   : Initial translation parameters [p_x0, p_y0] to add to rect, which defaults to [0 0]
    :return[np.array(2, 1)] p   : Final translation parameters [p_x, p_y]
    """

    # Initialize p to p0.
    p = p0

    # ===== your code here! =====
    # Hint: Iterate over num_iters and for each iteration, construct a linear system (Ax=b) that solves for a x=delta_p update
    # Construct [A] by computing image gradients at (possibly fractional) pixel locations.
    # We suggest using RectBivariateSpline from scipy.interpolate to interpolate pixel values at fractional pixel locations
    # We suggest using lstsq from numpy.linalg to solve the linear system
    # Once you solve for [delta_p], add it to [p] (and move on to next iteration)
    #
    # HINT/WARNING:
    # RectBivariateSpline and Meshgrid use inconsistent defaults with respect to 'xy' versus 'ij' indexing:
    # https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RectBivariateSpline.ev.html#scipy.interpolate.RectBiv
    # https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html

    height, width = It.shape
    x1, y1, x2, y2 = rect[0], rect[1], rect[2], rect[3]
    top_left = [x1, y1]
    top_right = [x2, y1]
    bottom_left = [x1, y2]
    bottom_right = [x2, y2]

    spline_t = RectBivariateSpline(np.arange(height), np.arange(width), It)
    spline_t1 = RectBivariateSpline(np.arange(height), np.arange(width), It1)
    rows = np.arange(x1, x2 + 1)
    col = np.arange(y1, y2 + 1)
    x, y = np.meshgrid(rows, col)
    template_patch = spline_t.ev(y, x)

    for i in range(num_iters):
      wx = x + p[0]
      wy = y + p[1]

      if (wx.min() < 0 or wy.min() < 0 or wx.max() >= width or wy.max() >= height):
          break

      imaget1_warp = spline_t1.ev(wy, wx)

      error = (template_patch - imaget1_warp)
      error = error.flatten()
      gradient_x = spline_t1.ev(wy, wx, dx = 0, dy = 1).flatten()
      gradient_y = spline_t1.ev(wy, wx, dx = 1, dy = 0).flatten()

      A = np.vstack((gradient_x, gradient_y)).T
      delta_p, residuals, rank, s = lstsq(A, error, rcond=None)

      p += delta_p
      if np.linalg.norm(delta_p) < threshold:
          break


    # ===== End of code =====
    return p
```

Debug Q2.2

A few tips to debug your implementation:

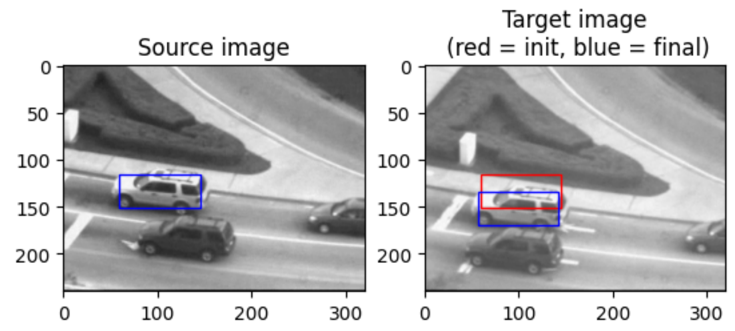
- Feel free to use and modify the following snippet to debug your implementation. The snippet simply visualizes the translation resulting from running LK on a single frame. You should be able to see a slight shift in the template.
- You may also want to visualize the image gradients you compute within your LK implementation
- Plot iterations vs the norm of delta_p

```
def draw_rect(rect,color):
    w = rect[2] - rect[0]
    h = rect[3] - rect[1]
    plt.gca().add_patch(patches.Rectangle((rect[0],rect[1]), w, h, linewidth=1, edgecolor=color, facecolor='none'))
```

```
num_iters = 100
threshold = 0.01
seq = np.load("/content/carseq.npy")
rect = [59, 116, 145, 151]
It = seq[:,:,0]

# Source frame
plt.figure()
plt.subplot(1,2,1)
plt.imshow(It, cmap='gray')
plt.title('Source image')
draw_rect(rect,'b')

# Target frame + LK
It1  = seq[:,:, 20]
plt.subplot(1,2,2)
plt.imshow(It1, cmap='gray')
plt.title('Target image\n (red = init, blue = final)')
p = LucasKanade(It, It1, rect, threshold, num_iters, p0=np.zeros(2))
rect_t1 = rect + np.concatenate((p,p))
draw_rect(rect,'r')
draw_rect(rect_t1,'b')
```



Q2.3: Tracking with template update (15 points)

```
def TrackSequence(seq, rect, num_iters, threshold):
    """
    :param seq       : (H, W, T), sequence of frames
    :param rect      : (4, 1), coordinates of template in the initial frame. top-left and bottom-right corners.
    :param num_iters : int, number of iterations for running the optimization
    :param threshold : float, threshold for terminating the LK optimization
    :return: rects   : (T, 4) tracked rectangles for each frame
    """
    H, W, N = seq.shape

    rects =[]
    It = seq[:,:,0]

    initial_frame = seq[:, :, 0]
    initial_rect = rect.copy()
    epsilon = 1.5

    rects.append(rect)
    p0 = np.zeros(2)

    # Iterate over the car sequence and track the car
    for i in range(1, seq.shape[2]):

        # ===== your code here! =====
        # TODO: add your code track the object of interest in the sequence

        It1 = seq[:, :, i]
        p = LucasKanade(It, It1, rect, threshold, num_iters, p0).flatten()

        p_star = LucasKanade(initial_frame, It1, initial_rect, threshold, num_iters, p0).flatten()

        if np.linalg.norm(p_star - p) <= epsilon:
            rect = rect + np.array([p_star[0], p_star[1], p_star[0], p_star[1]])
        else:
            rect = rect + np.array([p[0], p[1], p[0], p[1]])
            p0 = p

        # rect = rect + np.array([p[0], p[1], p[0], p[1]])
        rects.append(rect.copy())
        It = It1



        # ===== End of code =====

    rects = np.array(rects)
    assert rects.shape == (N, 4), f"Your output sequence {rects.shape} is not ({N}x{4})"
    return rects
```

Q2.3 (a) - Track Car Sequence

Run the following snippets. If you have implemented LucasKanade and TrackSequence function correctly, you should see the box tracking the car accurately. Please note that the tracking might drift slightly towards the end, and that is entirely normal.

Feel free to play with these snippets of code by playing with the parameters.

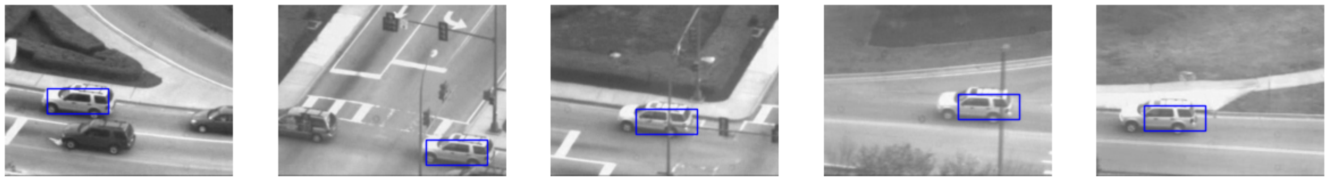
```
def visualize_track(seq,rects,frames):
    # Visualize tracks on an image sequence for a select number of frames
    plt.figure(figsize=(15,15))
    for i in range(len(frames)):
        idx = frames[i]
        frame = seq[:, :, idx]
        plt.subplot(1,len(frames),i+1)
        plt.imshow(frame, cmap='gray')
        plt.axis('off')
        draw_rect(rects[idx],'b');
```

```
seq = np.load("/content/carseq.npy")
rect = [59, 116, 145, 151]

# NOTE: feel free to play with these parameters
# TUNED PARAMETERS
num_iters = 10000
threshold = 0.02

rects = TrackSequence(seq, rect, num_iters, threshold)

visualize_track(seq,rects,[0, 79, 159, 279, 409])
```



Q2.3 (b) - Track Girl Sequence

Same as the car sequence.

```
# Loads the squence
seq = np.load("/content/girlseq.npy")
rect = [280, 152, 330, 318]

# NOTE: feel free to play with these parameters
# TUNED PARAMETERS
num_iters = 10000
threshold = 0.02

rects = TrackSequence(seq, rect, num_iters, threshold)

visualize_track(seq,rects,[0, 14, 34, 64, 84])
```