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.

Collaborators: Steven Man, Ethan Holand

In [1]:
```
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.

In [2]:
```
# Download the data
if not os.path.exists('/content/carseq.npy'):
    !wget https://www.cs.cmu.edu/~deva/data/carseq.npy -O /content/car
seq.npy
if not os.path.exists('/content/girlseq.npy'):
    !wget https://www.cs.cmu.edu/~deva/data/girlseq.npy -O /content/gi
rlseq.npy
```

```
--2024-02-16 04:08:07--  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... conn
ected.
HTTP request sent, awaiting response... 200 OK
Length: 254976128 (243M)
Saving to: ‘/content/carseq.npy’

/content/carseq.npy 100%[===================>] 243.16M  4.42MB/s    in
57s

2024-02-16 04:09:05 (4.25 MB/s) - ‘/content/carseq.npy’ saved [2549761
28/254976128]

--2024-02-16 04:09:05--  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... conn
ected.
HTTP request sent, awaiting response... 200 OK
Length: 27648128 (26M)
Saving to: ‘/content/girlseq.npy’

/content/girlseq.np 100%[===================>]  26.37M  4.38MB/s    in
6.2s

2024-02-16 04:09:11 (4.26 MB/s) - ‘/content/girlseq.npy’ saved [276481
28/27648128]
```

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! =====

$$\frac{\partial W}{\partial p} = \frac{\partial (x+p)}{\partial p} = \frac{\partial x}{\partial p} + \frac{\partial p}{\partial p} = 0 + \begin{bmatrix} \frac{\partial p_1}{\partial p_1} & \frac{\partial p_1}{\partial p_2} \\ \frac{\partial p_2}{\partial p_1} & \frac{\partial p_2}{\partial p_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

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

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

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

$$A = \frac{\partial I_{t+1}(x')}{\partial x^T}$$
$$b = T_t(x) - I_{t+1}(x')$$

===== 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 have a full column rank. ===== end of your answer =====

Q2.2: Lucas-Kanade (20 points)

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

In [10]:
```
from scipy.interpolate import RectBivariateSpline
from numpy.linalg import lstsq
import scipy

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 rec
tangular 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 itera
tions
    :param[np.array(2, 1)] p0   : Initial translation parameters [p_x
0, 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 fraction
al) pixel locations.
    # We suggest using RectBivariateSpline from scipy.interpolate to i
nterpolate pixel values at fractional pixel locations
    # We suggest using lstsq from numpy.linalg to solve the linear sys
tem
    # 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.inter
polate.RectBivariateSpline.ev.html#scipy.interpolate.RectBivariateSpli
ne.ev
    # https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.
html

    x1, y1, x2, y2 = rect
    x, y = np.meshgrid(np.arange(x1, x2), np.arange(y1, y2 ))
    spline_It = RectBivariateSpline(np.arange(It.shape[0]), np.arange
(It.shape[1]), It)
    spline_It1 = RectBivariateSpline(np.arange(It1.shape[0]), np.arang
e(It1.shape[1]), It1)

    It_warped = spline_It.ev(y, x)
    for _ in range(num_iters):
        x_shifted = x + p[0]
```

```
        y_shifted = y + p[1]
        # Warp It1
        It1_warped = spline_It1.ev(y_shifted, x_shifted)
        It1_grad_x = spline_It1.ev(y_shifted, x_shifted, dx=0, dy=1)
        It1_grad_y = spline_It1.ev(y_shifted, x_shifted, dx=1, dy=0)
        # Construct the A matrix and b vector.
        A = np.vstack((It1_grad_x.flatten(), It1_grad_y.flatten())).T
        b = It_warped.flatten() - It1_warped.flatten()
        # Solve for delta_p using least squares.
        delta_p, _, _, _ = np.linalg.lstsq(A, b, rcond=None)
        # Update p and check for convergence.
        p += delta_p
        a = np.linalg.norm(delta_p)
        if a < threshold:
            break

    return p
```

In [3]:

```
```

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

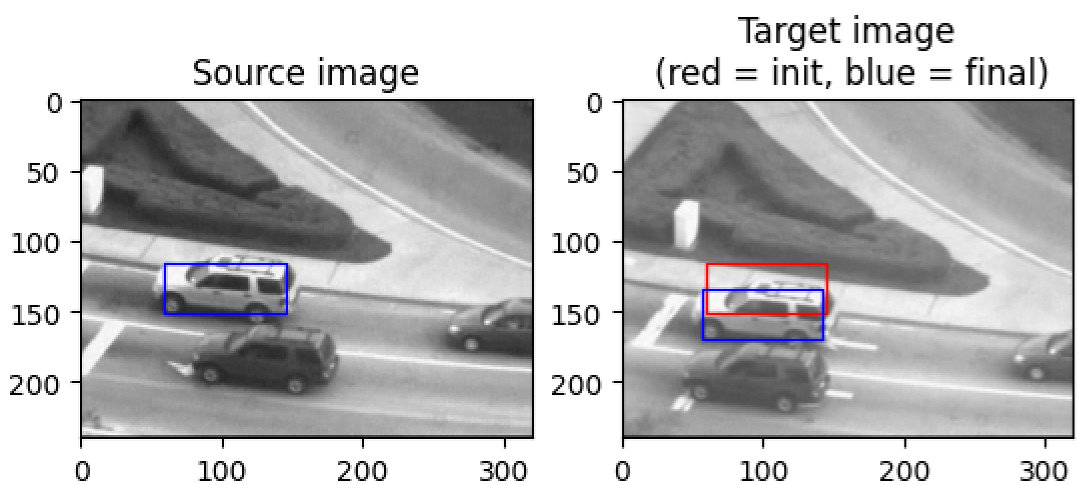
In [4]:

```
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, lin
ewidth=1, edgecolor=color, facecolor='none'))
```

In [5]:
```
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)

In [6]:
```
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 optim
ization
    :param threshold : float, threshold for terminating the LK optimiz
ation
    :return: rects   : (T, 4) tracked rectangles for each frame
    """
    H, W, N = seq.shape
    rects =[]
    It = seq[:,:,0]
    rects.append(rect)

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

        # ===== your code here! =====
        # TODO: add your code track the object of interest in the sequ
ence
      It = seq[:,:,i]
      It1 = seq[:,:,i+1]
      rect = rects[i]
      p = LucasKanade(It, It1, rect,threshold, num_iters)
      x1,y1,x2,y2=rect
      updated_rect=updated_rect=[x1+p[0],y1+p[1],x2+p[0],y2+p[1]]
      rects.append(updated_rect)
        # ===== 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.

In [7]:
```
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');
```

In [8]:
```
seq = np.load("/content/carseq.npy")
rect = [59, 116, 145, 151]

# NOTE: feel free to play with these parameters
num_iters = 100
threshold = 0.01

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

visualize_track(seq,rects,[0, 79, 159, 279, 409])
#visualize_track(seq,rects,[0, 1, 2, 3, 4])
```

Q2.3 (b) - Track Girl Sequence

Same as the car sequence.

In [9]:
```
# Loads the squence
seq = np.load("/content/girlseq.npy")
rect = [280, 152, 330, 318]

# NOTE: feel free to play with these parameters
num_iters = 10000
threshold = 0.01

rects = TrackSequence(seq, rect, num_iters, threshold)
visualize_track(seq,rects,[0, 14, 34, 64, 84])
```