Credit

discussed with Rayna Hata, Ethan Holand, and Arthur Bucker

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.

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

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! =====

Since $W = x + p$ and x is not a function of p, and elements of p are not dependent on eachother:

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

===== 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 full column rank for Δp to have a unique solution.

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

Q2.2: Lucas-Kanade (20 points)

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

In [23]:
```python
from scipy.interpolate import RectBivariateSpline
from numpy.linalg import lstsq
from scipy.ndimage import shift

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
    x1, y1, x2, y2 = rect
    It_rbs = RectBivariateSpline(np.arange(It.shape[0]), np.arange(It.shape
[1]), It)
    It1_rbs = RectBivariateSpline(np.arange(It1.shape[0]), np.arange(It1.shape
[1]), It1)
    x_grid_temp, y_grid_temp = np.meshgrid(np.arange(x1,x2), np.arange(y1,y2))
    T_t = It_rbs.ev(y_grid_temp, x_grid_temp)

    for _ in range(num_iters):
        x_range_shifted = x_grid_temp + p[0]
        y_range_shifted = y_grid_temp + p[1]
        It1_rect = It1_rbs.ev(y_range_shifted, x_range_shifted)
        grad_x = It1_rbs.ev(y_range_shifted, x_range_shifted, dy=1)
        grad_y = It1_rbs.ev(y_range_shifted, x_range_shifted, dx=1)

        A = np.vstack((grad_x.flatten(), grad_y.flatten())).T
        b = T_t - It1_rect
        b = b.flatten().T

        delta_p = np.linalg.lstsq(A, b, rcond=None)[0]
        if np.linalg.norm(delta_p) < threshold:
            break

        p += delta_p

    # ===== 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) pixe
l locations.
    # We suggest using RectBivariateSpline from scipy.interpolate to interpola
te 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 iterati
on)
    #
    # 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.R
ectBivariateSpline.ev.html#scipy.interpolate.RectBivariateSpline.ev
    # https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html


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

Debug Q2.2

A few tips to debug your implementation:

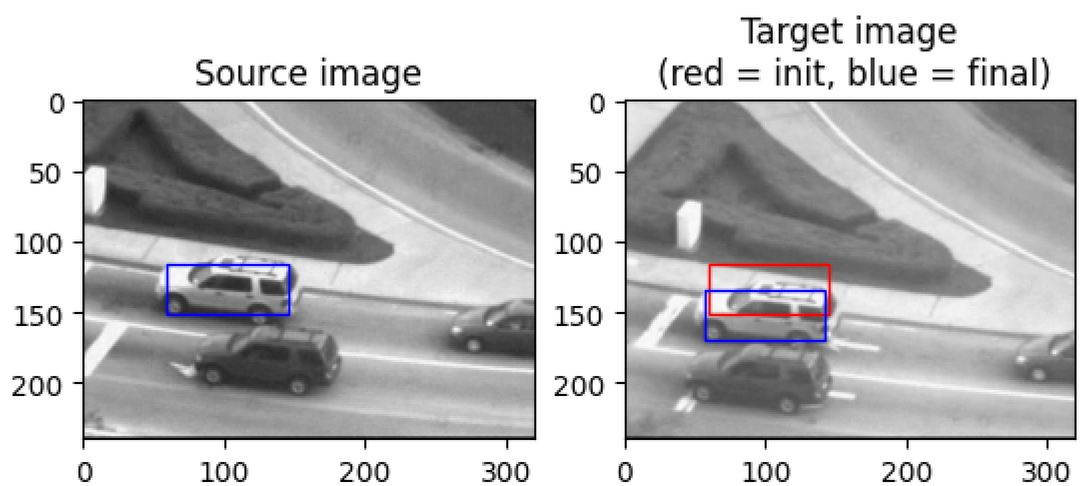
- Feel free to use and modify the following snippet to debug your implementation. The snippet simply visualizes the translation resulting from running LK on a single frame. You should be able to see a slight shift in the template.
- You may also want to visualize the image gradients you compute within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

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, linewidth=
1, edgecolor=color, facecolor='none'))
```

In [24]:
```
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 [15]:
```
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. t
op-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 =[rect]
    It = seq[:,:,0]

    # Iterate over the car sequence and track the car
    for i in range(N-1):

        # ===== your code here! =====
        # TODO: add your code track the object of interest in the sequence
        It = seq[:,:,i]
        It1 = seq[:,:,i+1]
        p = LucasKanade(It, It1, rect, threshold, num_iters, p0=np.zeros(2))
        rect = rect + np.concatenate((p,p))
        rects.append(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]:
```python
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 [25]:
```python
seq = np.load("./content/carseq.npy")
rect = [59, 116, 145, 151]

# 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, 79, 159, 279, 409])
```

Q2.3 (b) - Track Girl Sequence

Same as the car sequence.

In [17]:
```python
# 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])
```