Q1.1

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

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

- What is A and b?

$$A = \frac{\partial I(x')}{\partial x'^T} \frac{\partial W(x;p)}{\partial p^T} = \begin{pmatrix} \frac{\partial I(x'_1)}{\partial x'_1{}^T} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial I(x'_n)}{\partial x'_n{}^T} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\partial I(x'_1)}{\partial x'_1{}^T} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial I(x'_n)}{\partial x'_n{}^T} \end{pmatrix}$$

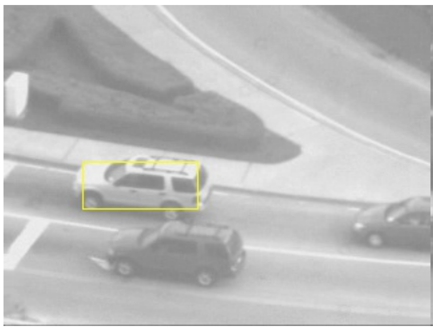
Note that in my implementation, I transform the gradient of image into a Nx2 matrix which contains the x-direction gradient and y-direction gradient, respectively.

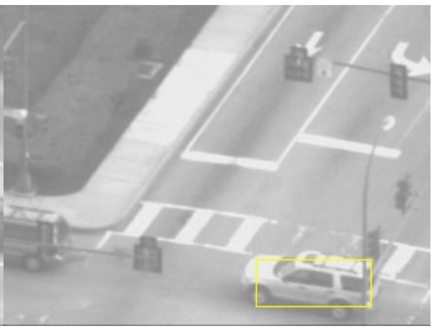
$$b = \begin{pmatrix} I_t(x_1) - I_{t+1}(W(x_1:p)) \\ \vdots \\ I_t(x_n) - I_{t+1}(W(x_n:p)) \end{pmatrix}$$

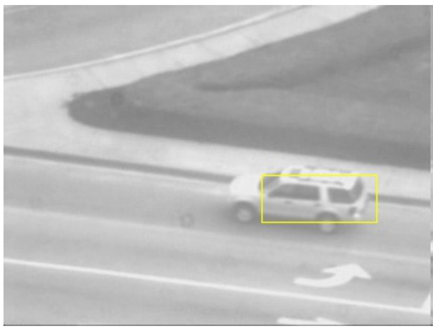
- What conditions must $A^T A$ meet so that a unique solution to Δp can be found?

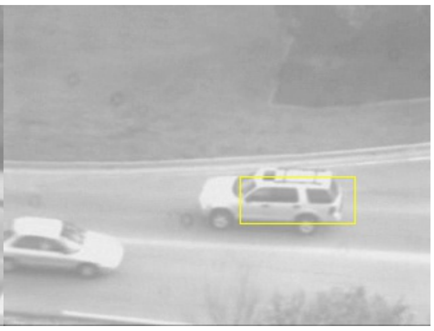
Due to the close-form solution: $\Delta p = (A^T A)^{-1} A^T b$, $A^T A$ should be invertible so that an unique solution can be found.

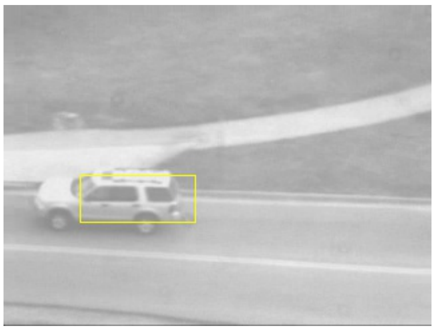
Q1.3 Report your tracking performance (image + bounding rectangle) at frames 1, 100, 200, 300 and 400.


Frame#1

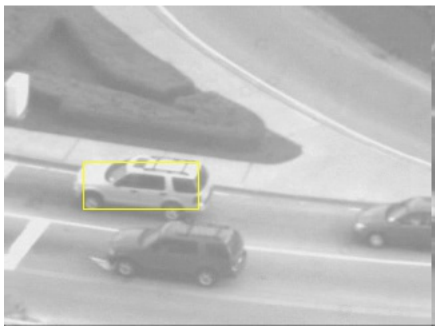

Frame#100

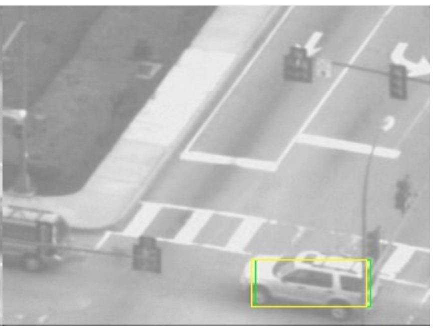

Frame#200

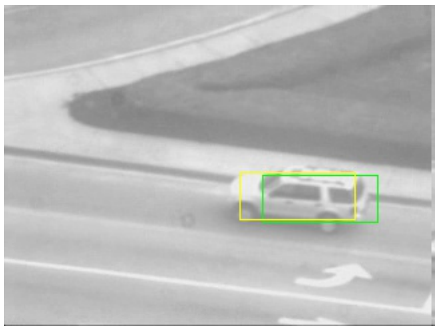

Frame#300

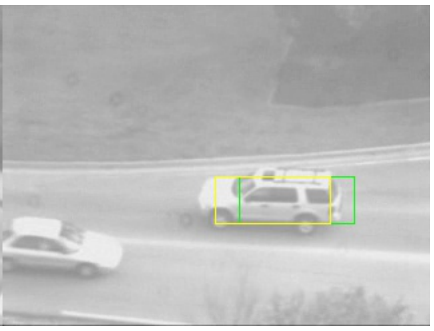

Frame#400

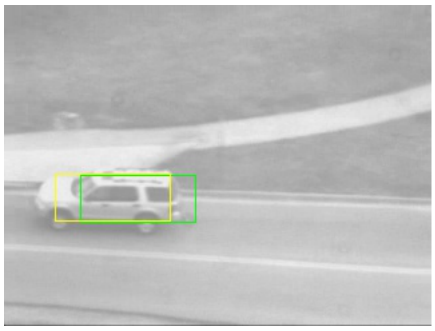
Q1.4 Report your tracking performance (image + bounding rectangle) at frames 1, 100, 200, 300 and 400. Note that the green rectangles are created with the baseline tracker in Q1.3, the yellow ones with the tracker in Q1.4.


Frame#1


Frame#100


Frame#200


Frame#300


Frame#400

Q2.1 (5 points) Express w as a function of It+1, It, and $\{B_k\}^K_{k=1}$, given Equation 6. Note that since the B_k's are orthobases, they are orthogonal to each other.

$$I_{t+1}(x) = I_t(x) + \sum_{k=1}^{K} w_k B_k(x)$$

Subtract $I_t(x)$ on both sides:

$$\begin{pmatrix} B_1(x) & \cdots & B_K(x) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_K \end{pmatrix} = I_{t+1}(x) - I_t(x)$$

Multiply transpose of $\{Bk\}^K_{k=1}$ on both sides:

$$\begin{pmatrix} B_1^T(x) \\ \vdots \\ B_K^T(x) \end{pmatrix} \begin{pmatrix} B_1(x) & \cdots & B_K(x) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_K \end{pmatrix} = \begin{pmatrix} B_1^T(x) \\ \vdots \\ B_K^T(x) \end{pmatrix} \big(I_{t+1}(x) - I_t(x)\big)$$

Since B_k's are orthobases, we have:

$$\begin{pmatrix} B_1^T(x)B_1(x) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & B_K^T(x)B_K(x) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_K \end{pmatrix} = \begin{pmatrix} B_1^T(x) \\ \vdots \\ B_K^T(x) \end{pmatrix} \big(I_{t+1}(x) - I_t(x)\big)$$

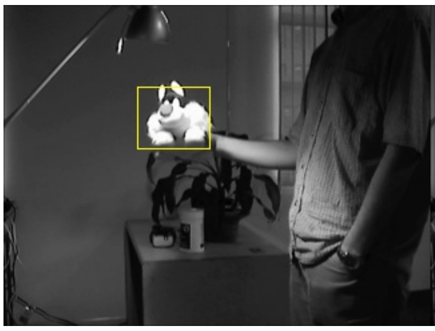
Multiply the inverse matrix of $\{B_k^T(x)B_k(x)\}$ and note that the inverse matrix of diagonal matrix equals to reciprocal of each diagonal elements:

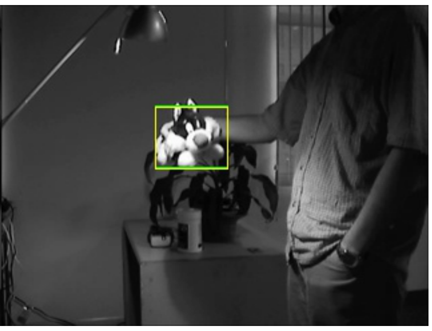
$$\begin{pmatrix} w_1 \\ \vdots \\ w_K \end{pmatrix} = \begin{pmatrix} \frac{1}{\|B_1(x)\|_2^2} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{\|B_k(x)\|_2^2} \end{pmatrix} \begin{pmatrix} B_1^T(x) \\ \vdots \\ B_K^T(x) \end{pmatrix} \big(I_{t+1}(x) - I_t(x)\big)$$

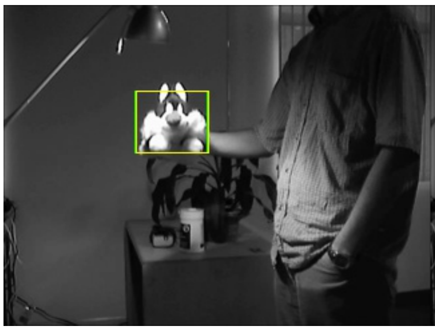
And combine all the stuffs we have:

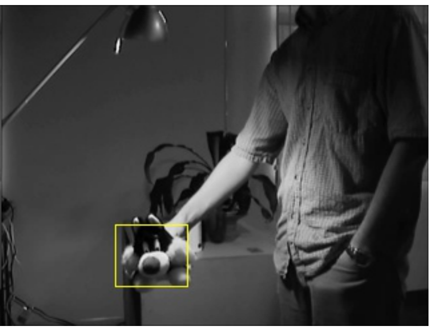
$$w = \begin{pmatrix} \frac{B_1^T(x)}{\|B_1(x)\|_2^2} \\ \vdots \\ \frac{B_K^T(x)}{\|B_k(x)\|_2^2} \end{pmatrix} \big(I_{t+1}(x) - I_t(x)\big)$$

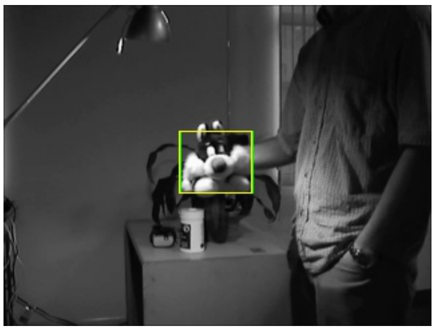
Q2.3 Please report the performance of this tracker at frames 1, 200, 300, 350 and 400 (the frame + bounding box), in comparison to that of the tracker in the first section. Note that the green rectangles are created with the baseline tracker in Q1.3, the yellow ones with the tracker in Q2.3.


Frame#1


Frame#200

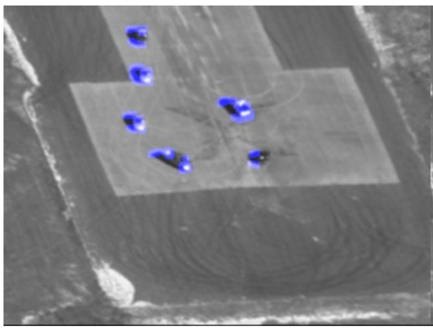

Frame#300

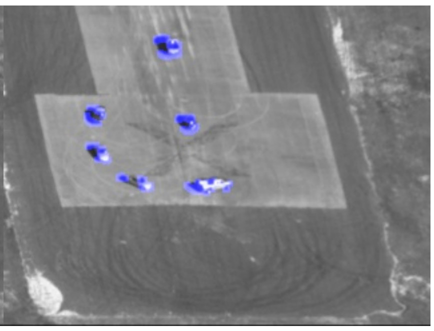

Frame#350

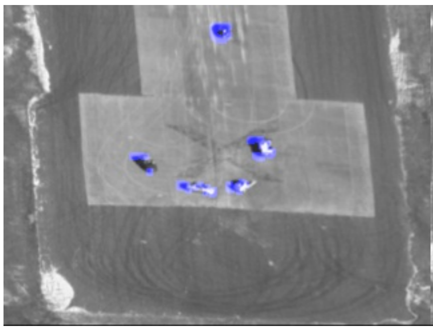

Frame#400

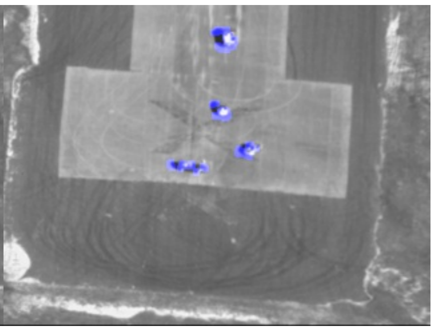
Note that the baseline tracker in Q1.3 works well if I choose a small threshold and that’s the reason why the baseline tracker in Q1.3 has the similar result with the one in Q2.3.

Q3.3 Report the performance at frames 30, 60, 90 and 120 with the corresponding binary masks superimposed.


Frame#30


Frame#60


Frame#90


Frame#120

Q4.1 In your own words please describe why the inverse compositional approach is more computationally efficient than the classical approach?

According to the formula in the original paper, the close-form solution of classical approach is:

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right] \qquad H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right].$$

In this case, for every iteration, we have to compute the gradient, warping and the Jacobian matrix. On the other hand, the close-form of inverse compositional approach can be expressed as below:

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \right] \qquad H = \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

We can notice that since the template image keep the same in every iteration. As a result, the inverse compositional only have to calculate the warping part iteratively and other parts (including gradient and Jacobian matrix) can be pre-computed.

Q4.2 Please write down the solution to Equation 15 in terms of the matrices S = XX^T , X and y.

First, we need to simplify the original expression and define it as the loss function:

$$\mathcal{L} = \frac{1}{2}\|y - X^T g\|_2^2 + \frac{1}{2}\lambda\|g\|_2^2 = \frac{1}{2}(y - X^T g)^T(y - X^T g) + \frac{1}{2}\lambda g^T g$$

$$= \frac{1}{2}(y^T - g^T X)(y - X^T g) + \frac{1}{2}\lambda g^T g = \frac{1}{2}(y^T y - 2y^T X^T g + g^T X X^T g) + \frac{\lambda}{2} g^T g$$

In order to get the close-form solution we need to calculate:

$$\frac{\partial \mathcal{L}}{\partial g} = 0$$

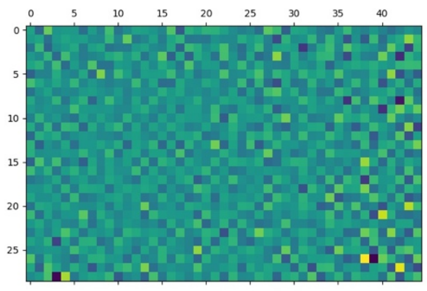
$$\frac{\partial}{\partial g}\left(\frac{1}{2}(y^T y - 2y^T X^T g + g^T X X^T g) + \frac{\lambda}{2} g^T g\right) = \frac{1}{2}(-2y^T X^T + 2XX^T g) + \lambda g$$

$$= -Xy + XX^T g + \lambda g = -Xy + (XX^T + \lambda I)g = -Xy + (S + \lambda I)g$$

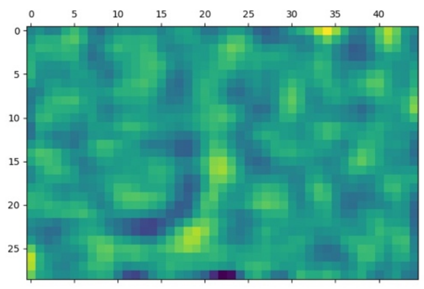
At the end, we can conclude and get the solution of g:

$$g = (S + \lambda I)^{-1} X y$$

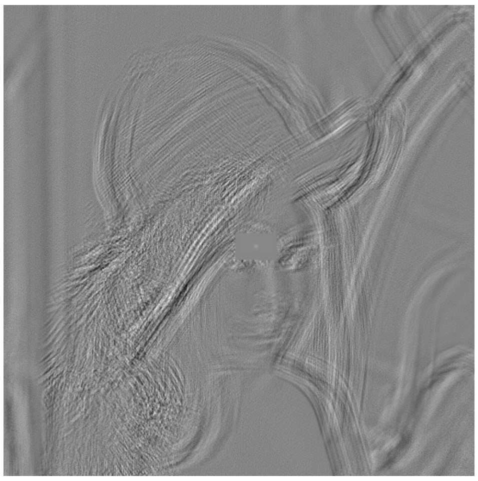
Q4.3 Visualize the resultant linear discriminant weight vector g for the penalty values λ = 0 and λ = 1 (remember to use the reshape function to convert g back into a 2D array). Apply the filter to the entire Lena image using the scipy.ndimage.correlate function. Visualize the responses for both values of λ. Include these visualizations in your written response document. Can you comment on which value of λ performs best and why? Place your answers and figures in your written response.



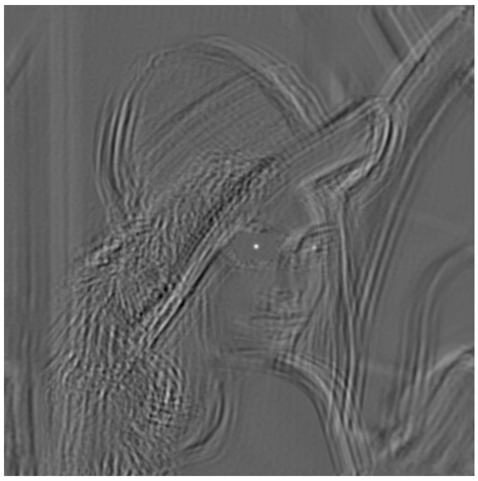
g for penalty λ = 0



g for penalty λ = 1



response for penalty λ = 0

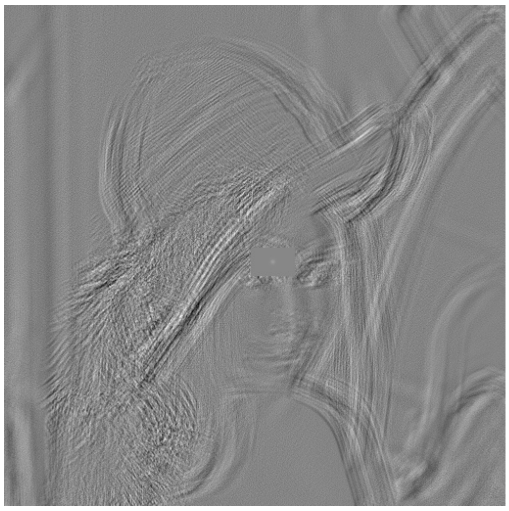


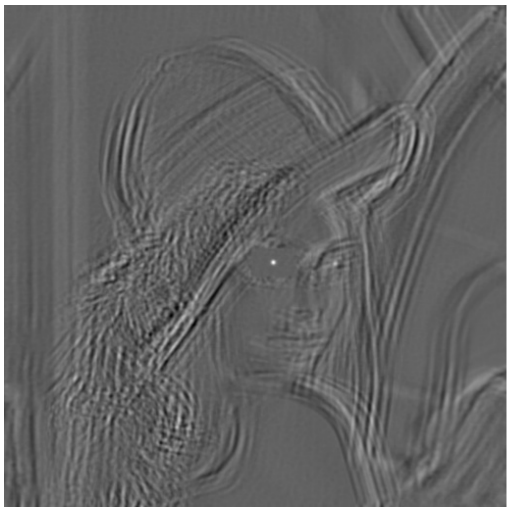
response for penalty λ = 1

The performance for penalty λ = 1 is better because it adopts the regularization term to make sure each element in g would not have large difference, that is, make sure the correlation filter g is smoother.

(Reference: https://optimization.mccormick.northwestern.edu/index.php/Trust-region_methods)

Q4.4 Visualize the response you get if you attempt to use the 2D convolution function scipy.ndimage.convolve. Why does this get a different response to the one obtained using correlate? How could you use the numpy indexing operations to get a response more similar to the one obtained using correlate? Place the answer in your written response.


response for penalty λ = 0


response for penalty λ = 1

Since we have to flip the kernel when using convolve to get the similar result with correlation, I first adopt np.flip() which is similar to the convolution part in HW1. As for solving by indexing, I refer to the document of python and apply in 2D case:
(Ref.: https://docs.scipy.org/doc/numpy/reference/generated/numpy.flip.html)

```
# numpy indexing for flipping the kernel
g_lamda1_flip = g_lamda1[::-1,::-1]
```