HW6 Computer Vision
Luka Eerens: MRSD 2018

Q1.1.1
Answer:

Affine warps are compose of 6 variables. These are denoted in subscripts below:

$$\frac{\partial W(x;p)}{\partial p^T} = \begin{bmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \frac{\partial W_x}{\partial p_3} & \frac{\partial W_x}{\partial p_4} & \frac{\partial W_x}{\partial p_5} & \frac{\partial W_x}{\partial p_6} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \frac{\partial W_y}{\partial p_3} & \frac{\partial W_y}{\partial p_4} & \frac{\partial W_y}{\partial p_5} & \frac{\partial W_y}{\partial p_6} \end{bmatrix}$$

Assuming that that the transpose of the product of x and y is equal to x:

$$\frac{\partial W(x;p)}{\partial p^T} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

Q1.1.2
Answer:

We want to reduce the mismatch between the object in question and the rest. Here we rely on the Euclidean norm to gauge what kind of a mismatch we are expecting and this can be approximated linearly for small changes in the position of the object. So for small change in the position of p (ΔP):

$$\Delta P = argmin\left|\left|L_{t+1}(x' + \Delta P) - L_t(x')\right|\right|_2^2$$

Apply the distributive property to yield:

$$\Delta P = argmin\left|\left|L_{t+1}(x') + L_{t+1}(\Delta P) - L_t(x')\right|\right|_2^2$$

From the documentation:

$$\Delta P = argmin \left|\left| L_{t+1}(x') + \frac{\partial L_{t+1}(x')}{\partial x'^T} * \frac{\partial W(x;P)}{\partial p^T} \Delta P - L_t(x') \right|\right|_2^2$$

And from the same documentation:

$$\Delta P = argmin\left|\left|A * \Delta P - b\right|\right|_2^2$$

Therefore:

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

$$L_{t+1}(x') - L_t(x') = -b$$
$$\therefore L_t(x') - L_{t+1}(x') = b$$

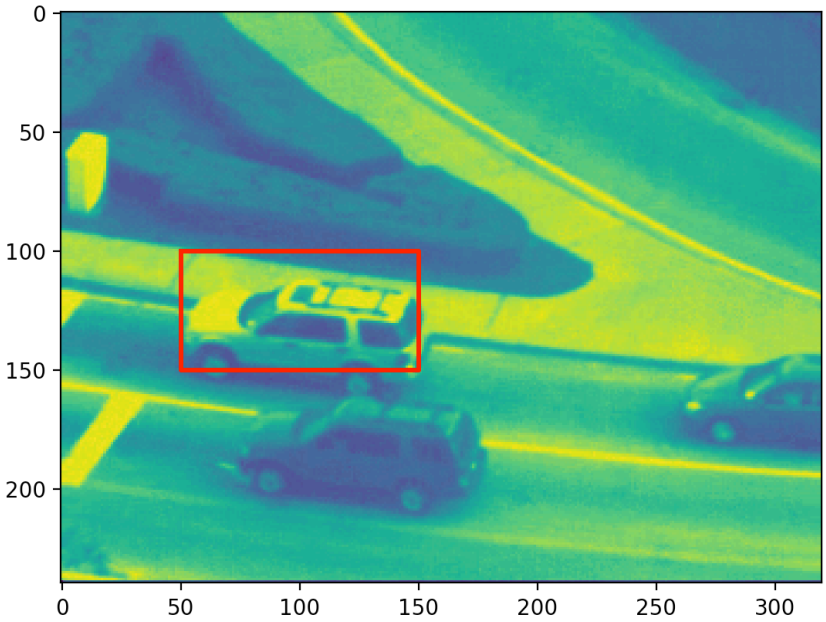
Q1.1.3
Answer:

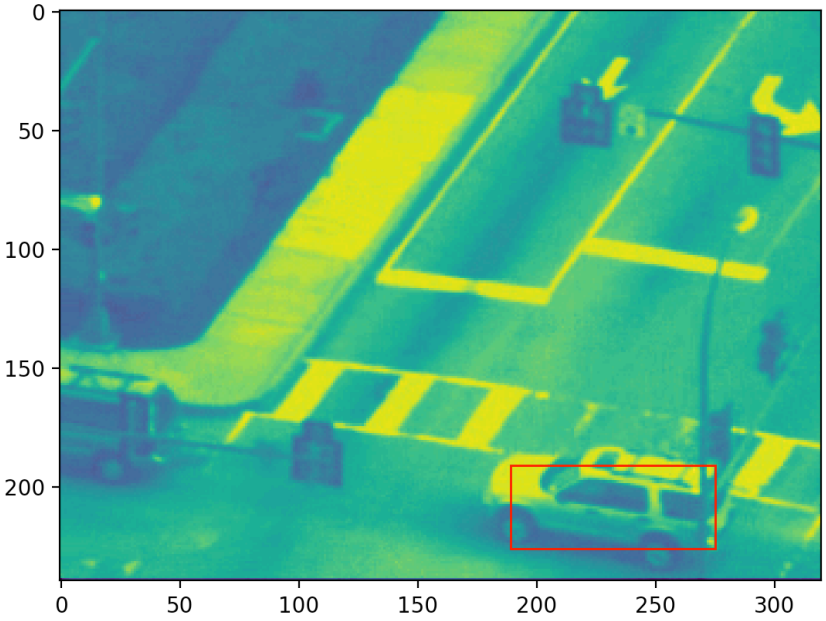
$A^T A$ has to be invertible otherwise a unique solution to the change in p cannot be found. So it should be full column rank.

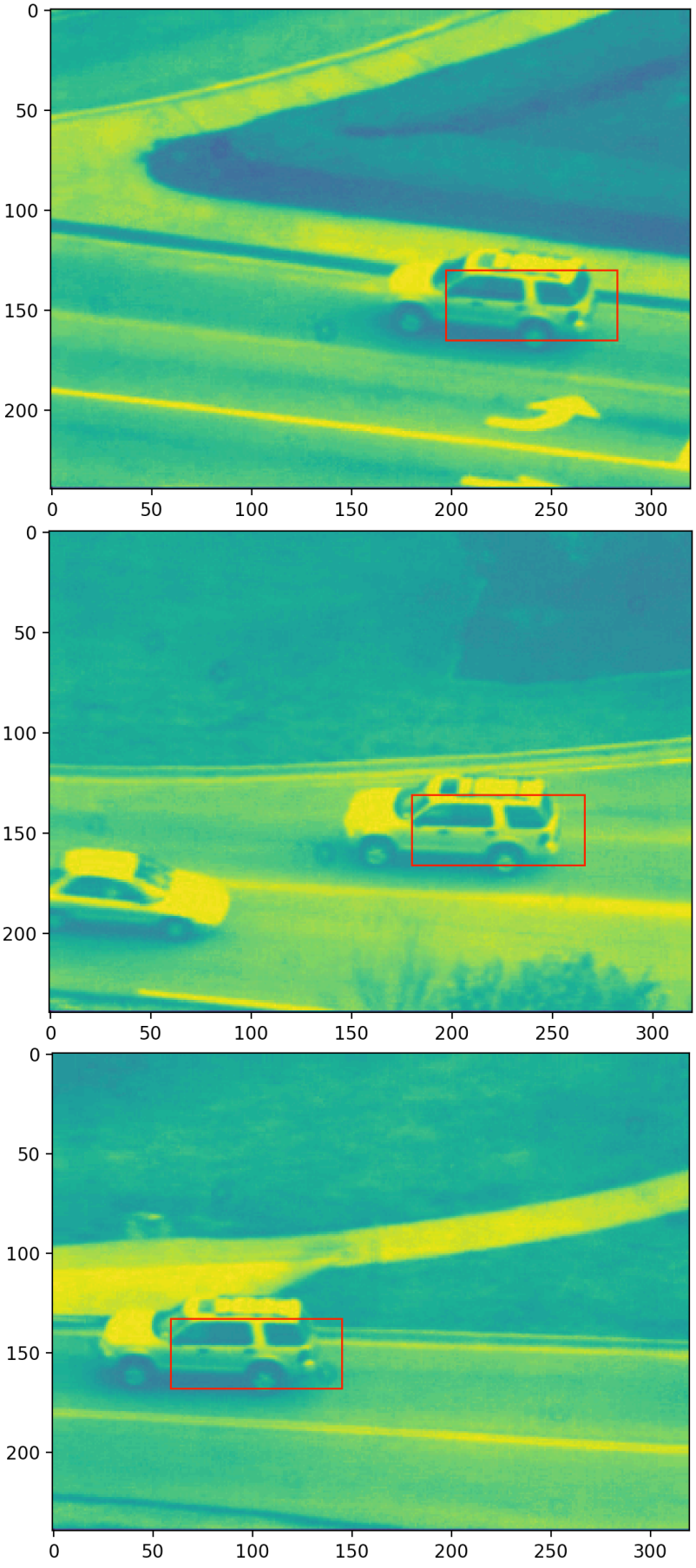
Q1.2
Answer:

Done in code

Q1.3
Answer:

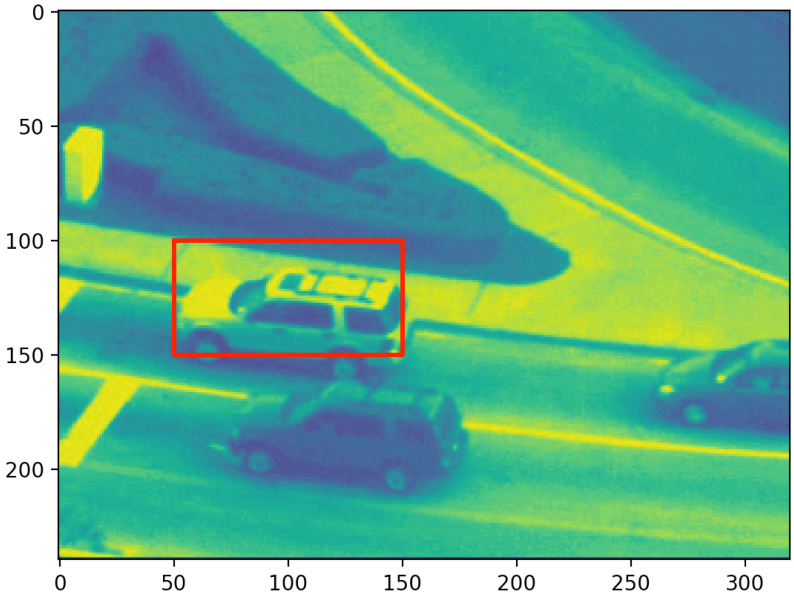


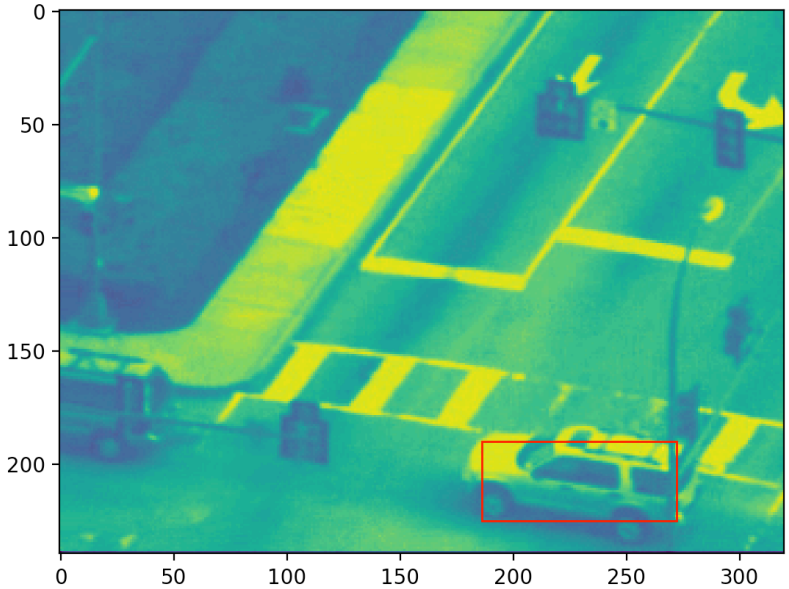


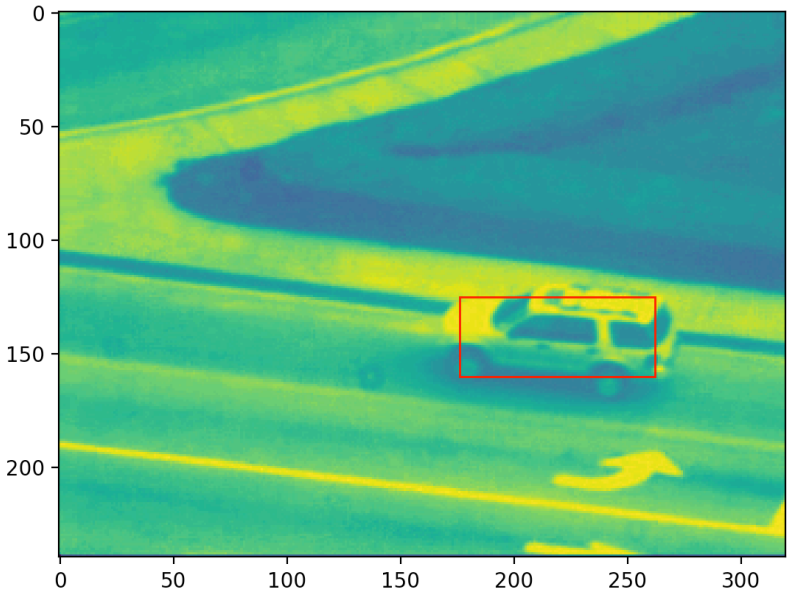

0
50
100
150
200
0
50
100
150
200
250
300

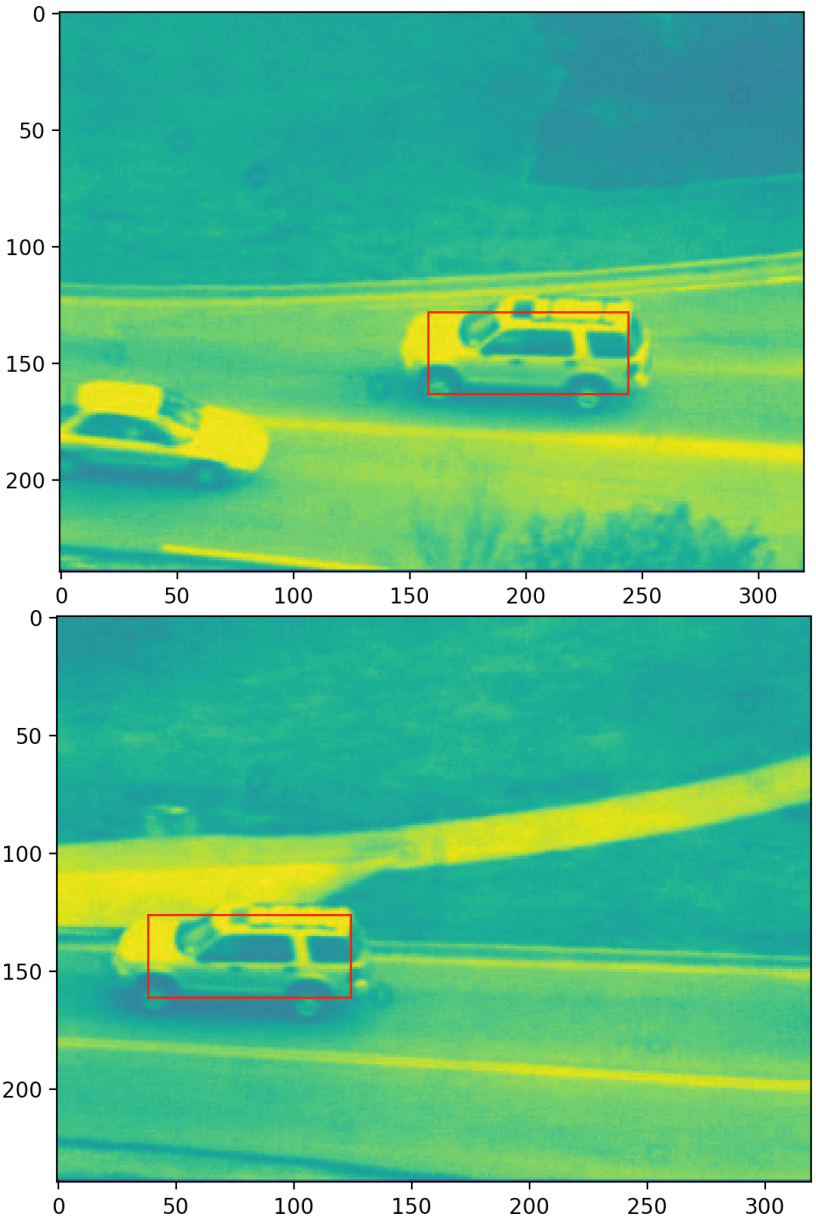
Q1.4

Answer:


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


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


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


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

Q2.1

Answer:

Starting with the equation that is provided (equation 6):

$$L_{t+1}(x) = L_t(x) + \sum_{k=1}^{K} w_k \, B_k(x)$$

We then compute the dot product across the entire equation wrt to $B_i(x)$ to get:

$$B_i(x) \cdot L_{t+1}(x) = B_i(x) \cdot L_t(x) + B_i(x) \cdot \sum_{k=1}^{K} w_k \, B_k(x)$$

We then bring Ls to one side to get:

$$B_i(x) \cdot L_{t+1}(x) - B_i(x) \cdot L_t(x) = B_i(x) \cdot \sum_{k=1}^{K} w_k \, B_k(x)$$

$$B_i(x) \cdot [L_{t+1}(x) - L_t(x)] = B_i(x) \cdot \sum_{k=1}^{K} w_k \, B_k(x)$$

And assuming that B_k are orthogonal vectors and normalised to be units vectors:

$$\sum_{x} B_i(x) \cdot [L_{t+1}(x) - L_t(x)] = w_i$$

Q2.2

Answer:

Done in code

Q2.3

Answer:

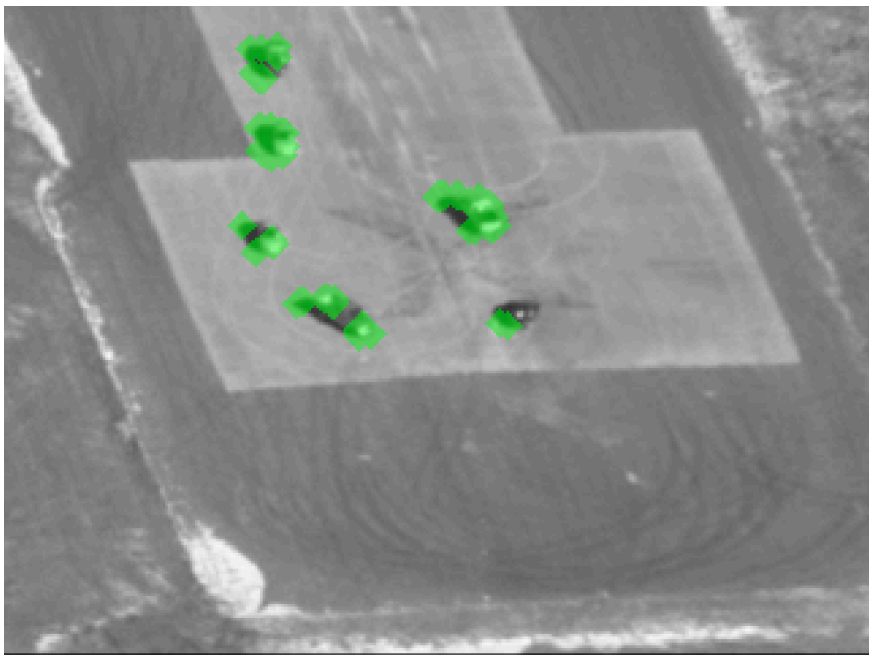
Blue box is LucasKanadeBasis, red box is LucasKanade. Tracking is very similar for the first 100 or iterations but then LucasKanade starts to drift, and offshoot the correction as the toy comes within the field of view of the red box.

Q3.1
Answer:
Done in code

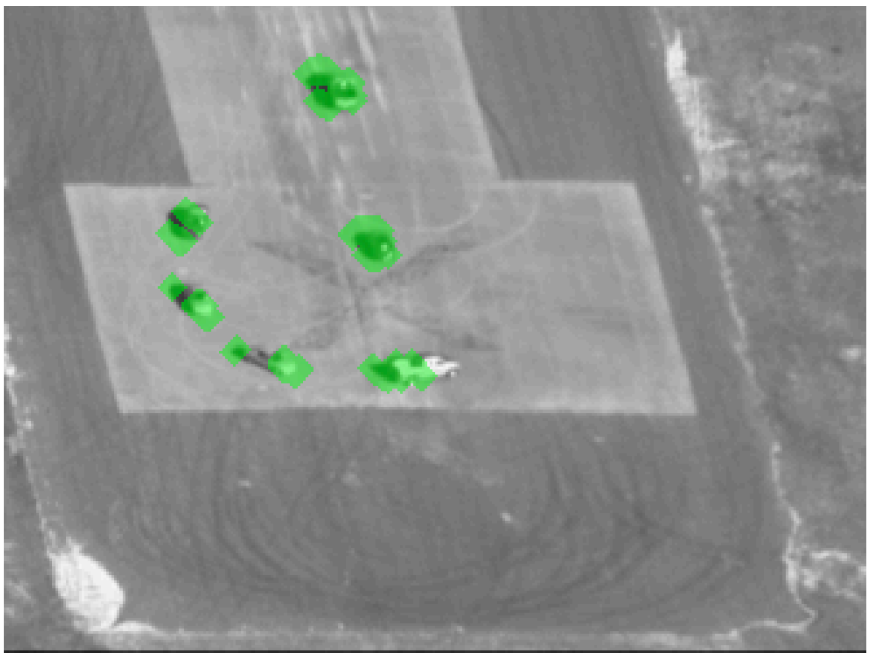
Q3.2
Answer:
Done in code

Q3.3
Answer:

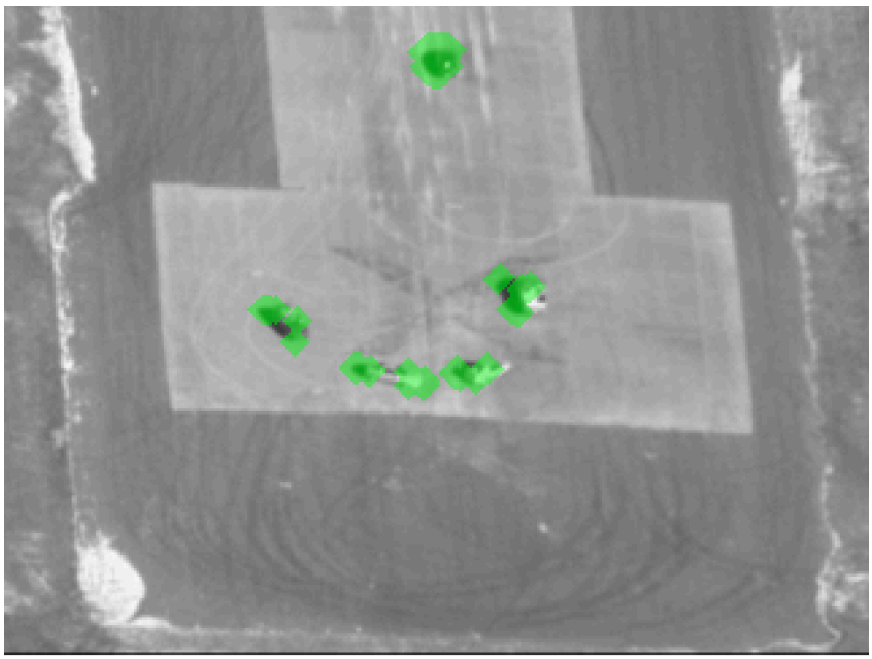
Frame 30



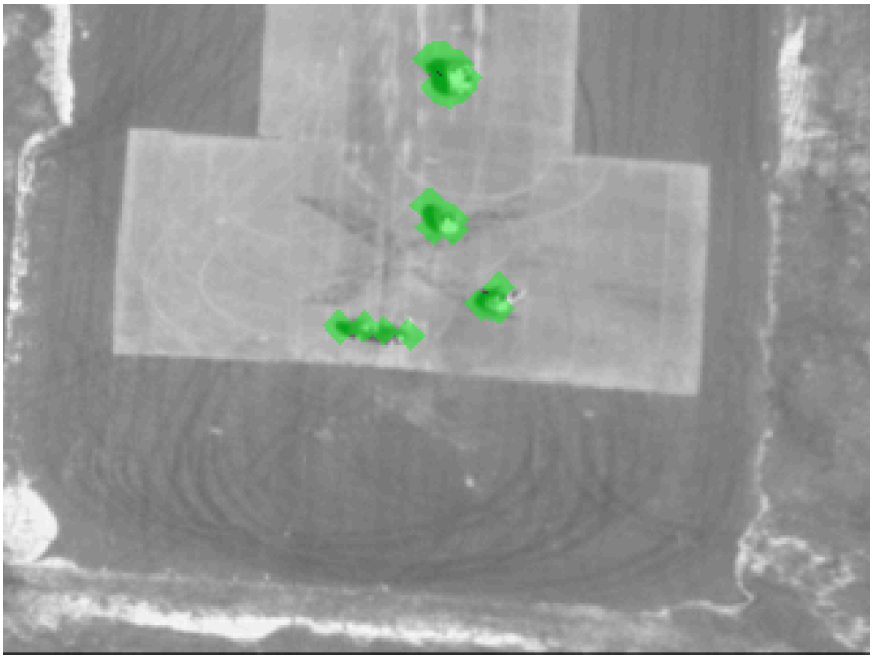
Frame 60



Frame 90

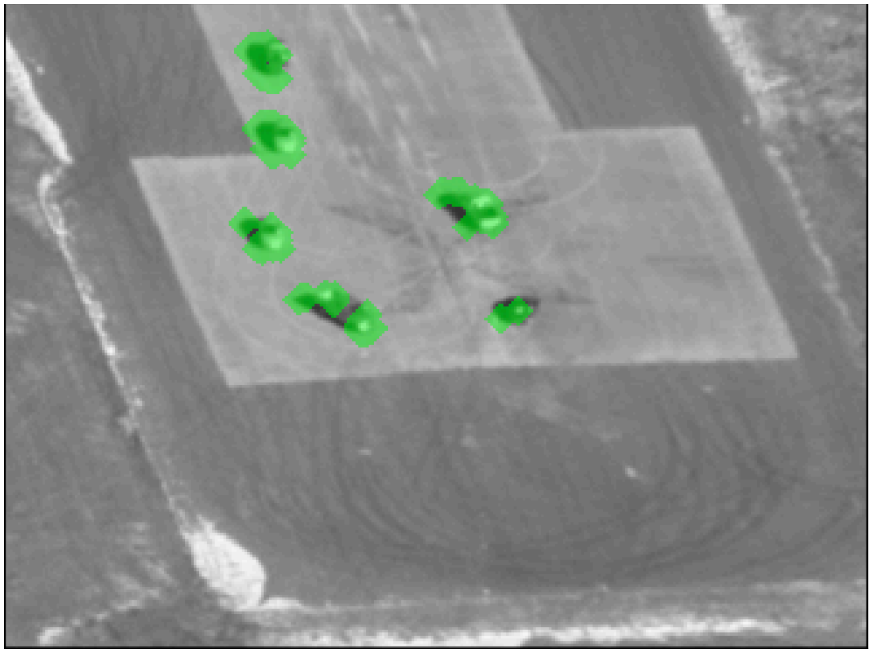


Frame 120

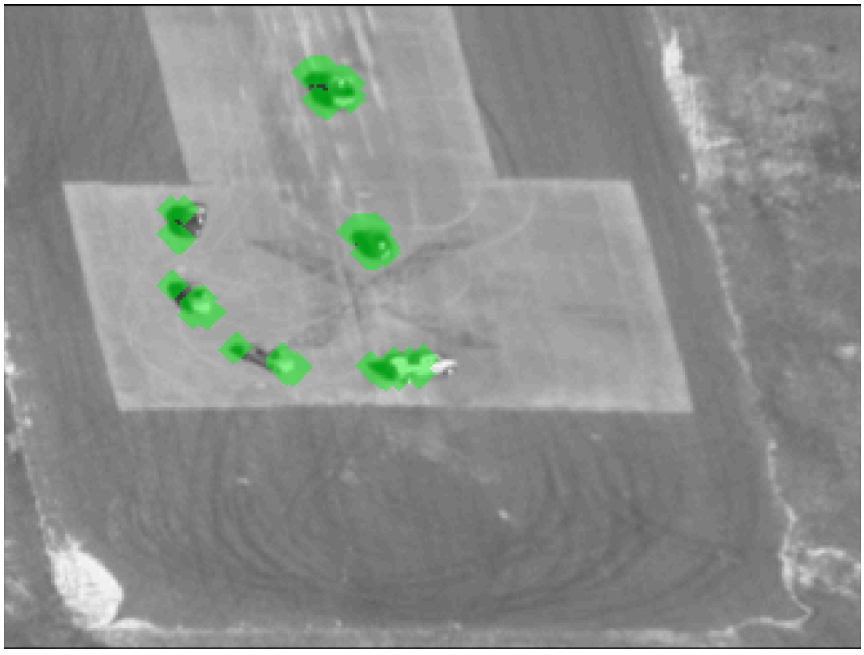


Q4.1
Answer:

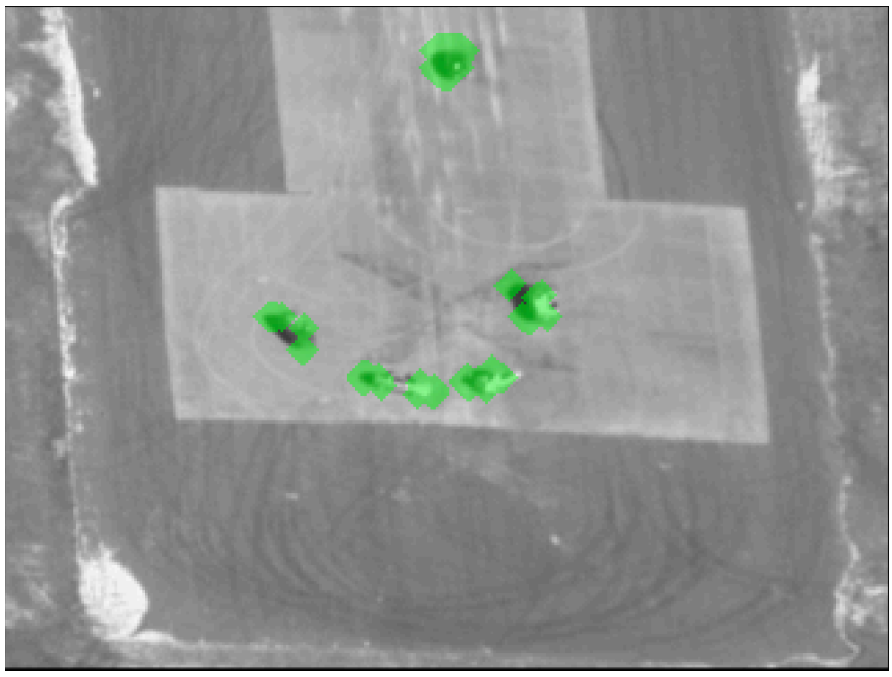
Frame 30



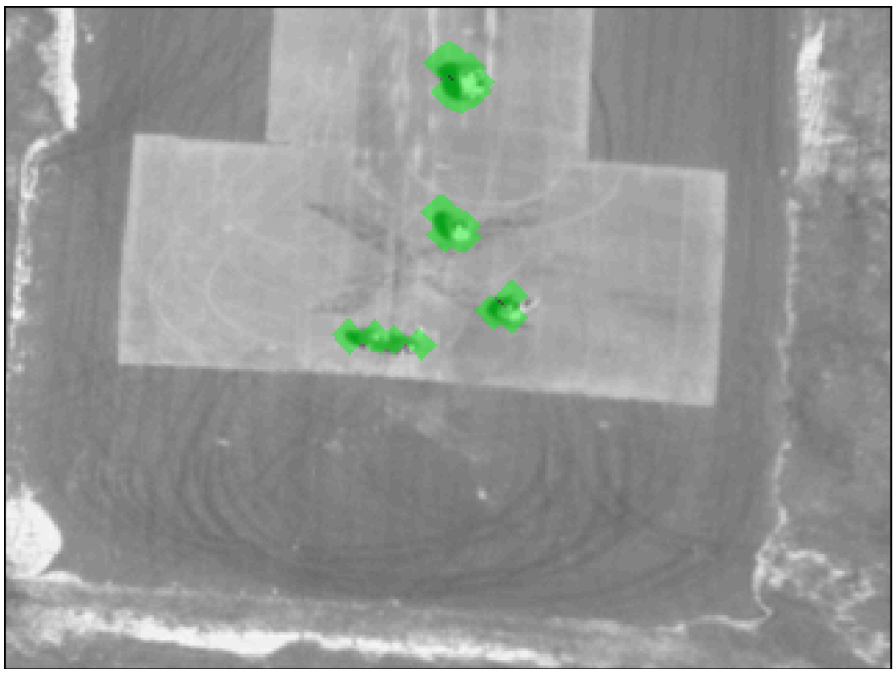
Frame 60



Frame 90



Frame 120



These images do not differ from those of the Lucas-Kanade Affine (LKA) algorithm, however what separates the two approaches is speed. These are the same algorithms, except that some steps are rearranged. The loops that run inside LKA are much slower because computing the Hessian is done with each loop, whereas Inverse Composition calculates the Hessian outside the loop. Furthermore, a marginal speed and space tradeoff comes about from the reduction in the need for some lines of code, namely a flattening step after computing affine transforms on gradients of the image.

Q4.2
Answer:

The solution to this problem is arrived by finding the derivative of equation 15 and then finding the min (derivative = 0) and then solving. So let's begin. We start of by writing this equation:

$$f(g) = arg\min_{g} \frac{1}{2}\left|\left|y - X^T g\right|\right|_2^2 + \frac{\lambda}{2}\left|\left|g\right|\right|_2^2$$

As:

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

Solve and use a clever matrix property to simplify

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

Expand, using the common distributive property:

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

We now need to compute the derivative to get:

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

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

Now from the instructions, we see that S = XX^T, therefore:

$$\frac{\partial f(g)}{\partial g} = -(y^T)(X^T) + g^T S + \lambda g^T$$

Now let's find the minimum of this derivative with respect to g since after all we are computing the argmin with respect to g:

$$0 = -(y^T)(X^T) + g^T S + \lambda g^T$$

$$(y^T)(X^T) = g^T(S + \lambda I)$$

$$yX = (S + \lambda I)^T g$$

$$\boldsymbol{(S + \lambda I)^{-1} yX = g}$$

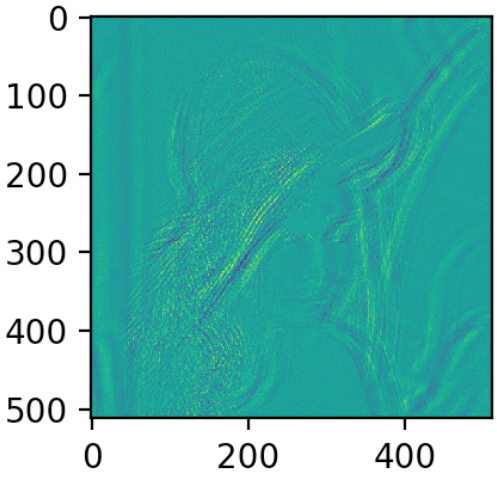
Q4.3 and Q4.4

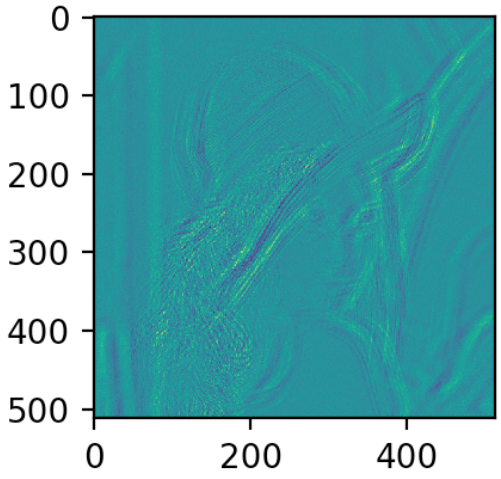
Answer:

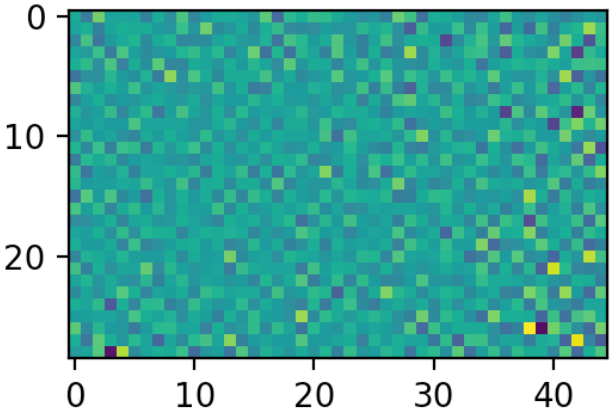
I have combined both question 3 and question 4 together.

The code in question that I wrote, creates subplots as follows. 2 images above, 1 image below, where the three images are:

- convolve (top left)
- correlation (top right)
- resultant linear discriminant weight vector (low right)



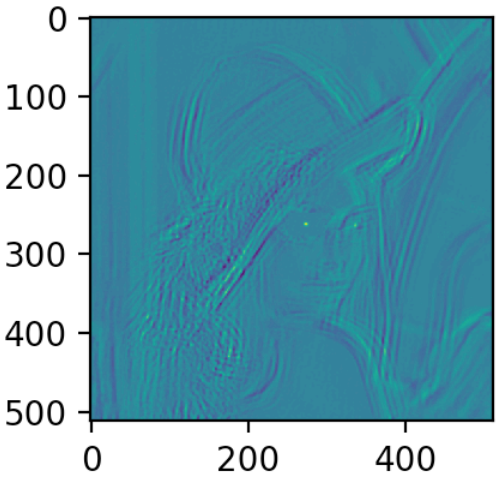


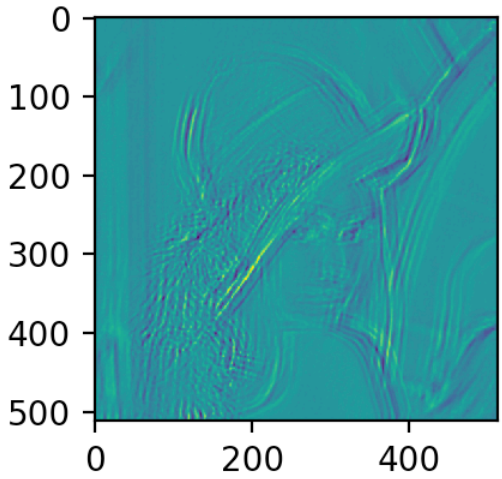


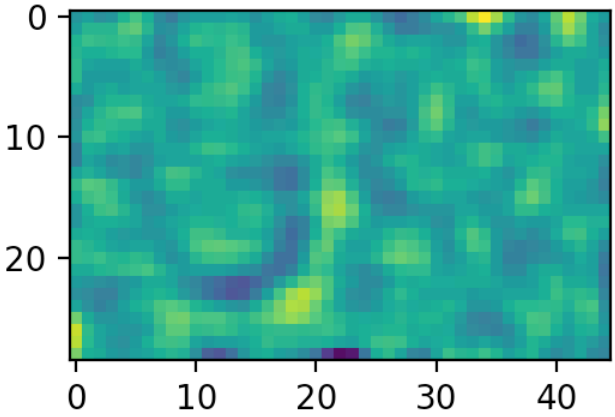
The pictures that you see above are the output from setting λ (gamma) to 0.

Notice the level of granularity in the resultant linear discriminant weight vector. It actually looks like random noise and hints at the filter not being learned as there is no structure that emerges from it.

The same is done below for $\lambda = 1$ with the same organization of subplots below:







In this one, notice that the resultant linear discriminant weight vector does look more structurally coherent. It does also look like there is a better performance in the resulting filters from having a λ that is equal to 1. The reason why this is, more formally is as follows:

Firstly, looking back at the final equation from question 4.2, we have:

$$(\boldsymbol{S} + \boldsymbol{\lambda I})^{-1}\boldsymbol{yX} = \boldsymbol{g}$$

Now if we plug $\lambda = 1$ into this equation, the solution becomes unique, whereas if we choose to go with $\lambda = 0$, we end up getting a total mess that the computer can't boil down into a single unique solution. As a result of the resultant linear discriminant weight vector is able to learn the filters better.

On the topic of how we might be able to get to something similar, one must remember that convolution filters and correlation filters are the exact same thing, except that a convolution filter is a correlation filter rotated by 180 degrees. So how to get similar results would entail rotating the filter or the image that passes through the image by 180 degrees, and then rotating the result again by 180 degrees so that it assumes the same orientation as it did originally.