16385-HW6

yunchuc

December 2023

Q2.1

Given

$$x = \begin{bmatrix} u \\ v \end{bmatrix}, W(p) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, we get

$$W(p) \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} (1+p_1)x + p_3 y + p_5 \\ p_2 x + (1+p_4)y + p_6 \\ 1 \end{bmatrix}$$

If we convert it back to heterogeneous coordinate

$$W = \begin{bmatrix} W_x\{u,v\} \\ W_y\{u,v\} \end{bmatrix} = \begin{bmatrix} (1+p_1)u + p_3 v + p_5 \\ p_2 u + (1+p_4)v + p_6 \end{bmatrix}$$

By definition of Jacobian,

$$J = \frac{\partial W}{\partial p} = \begin{bmatrix} \frac{\partial W_x\{u,v\}}{p_1} & \frac{\partial W_x\{u,v\}}{p_2} & .. & \frac{\partial W_x\{u,v\}}{p_6} \\ \frac{\partial W_y\{u,v\}}{p_1} & \frac{\partial W_y\{u,v\}}{p_2} & .. & \frac{\partial W_y\{u,v\}}{p_6} \end{bmatrix}$$

$$= \begin{bmatrix} u & 0 & v & 0 & 1 & 0 \\ 0 & u & 0 & v & 0 & 1 \end{bmatrix}$$

Q2.2

The big-O for initialization goes as follows:
First, computing ∇T is $O(n)$ because to evaluate the derivative of template image, we have to iterate over every pixel.
Then, we need to calculate $\frac{\partial W}{\partial p}$, which is the Jacobian of the warp image. The big O would be $O(pn)$, because for each pixel in template image, we need to apply each parameter to obtain derivative.

Multiplying the two matrices is $O(pn)$
Then, to compute hessian matrix, we need to iterate over x, whiich results in $O(p^2n)$
Overall, the complexity is $O(p^2n)$

In each iteration,
Warp the image requires $O(pn)$ computation for iterating through every pixel
Compute Δp is $O(p^3)$
Finally, updating p is $O(p^2)$
So overall complexity of each iteration is $O(pn + p^3)$

Compare with Lucas-Kanade method, Mattews-baker method is less computational expensive as W and hessian matrix are all pre-computed, which saves $O(pn)$ and $O(p^2n)$ respectively.

3.4

How do your algorithms perform on each video? Which are the differences of three algorithms in terms of performance and why do we have those differences? At what point does the algorithm break down and why does this happen?

Lucas-Kanade with translation only works really well in tracking the object; it successfully tracks the object in all three test cases. However, its disadvantage lies in computing only the translation, meaning the size of the square will not change if the object moves closer or further away.

Lucas-Kanade with affine works well in adapting to changing shapes to enclose the moving object but failed to track car2 after it passed the traffic light. The square extended beyond the boundary, seemingly recognizing the traffic light as part of the car. It would work better if the script included bounding the square within limits, but I only included the original version of the script.

Finally, Inverse Compositional succeeded only in land and did not perform well in car1. When the car drove into the shady region marking the intensity change, it failed to recognize the car. It also didn't perform well in car2. Similar to Lucas-Kanade with affine, it also recognized the second traffic light as part of the car.

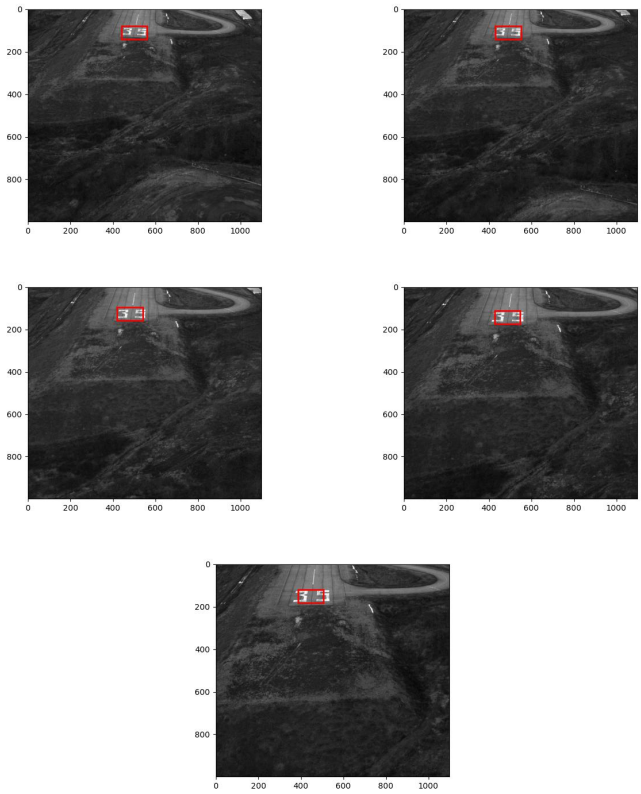

Figure 1: LucasKanade with translation only. Land with frame (1,10,20,30,40)

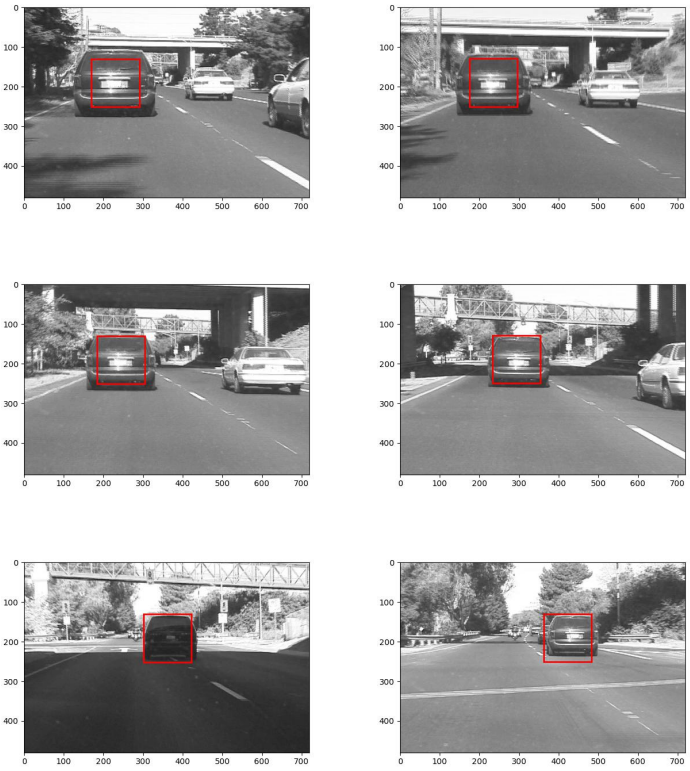

Figure 2: LucasKanade with translation only.
car1 with frame (1,50,100,150,200,250)

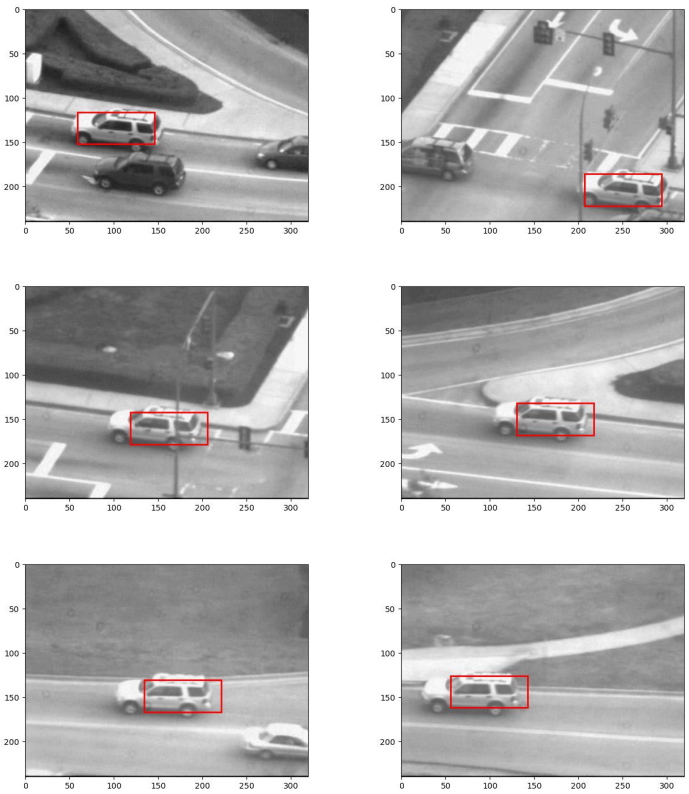

Figure 3: LucasKanade with translation only.
car2 with frame (1,80,160,240,320,400)

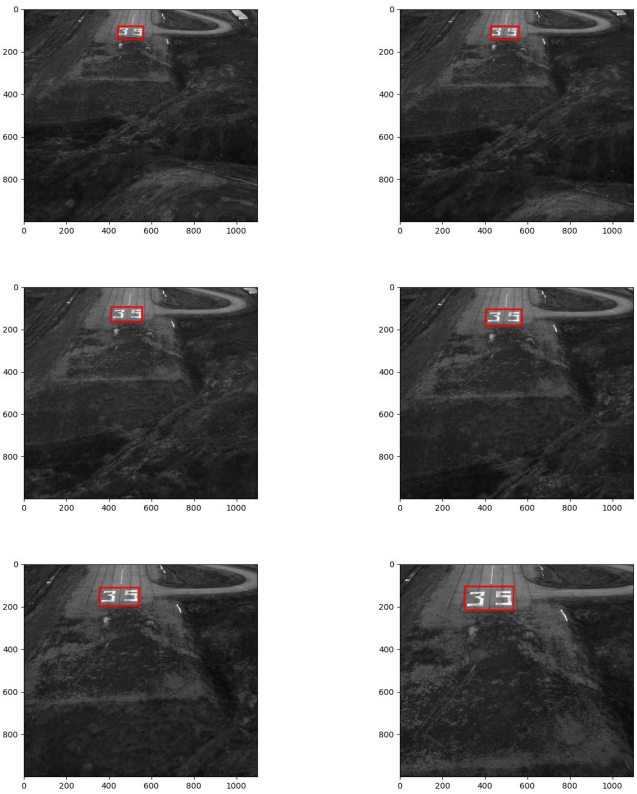

Figure 4: LucasKanade with affine.land with frame (1,10,20,30,40)

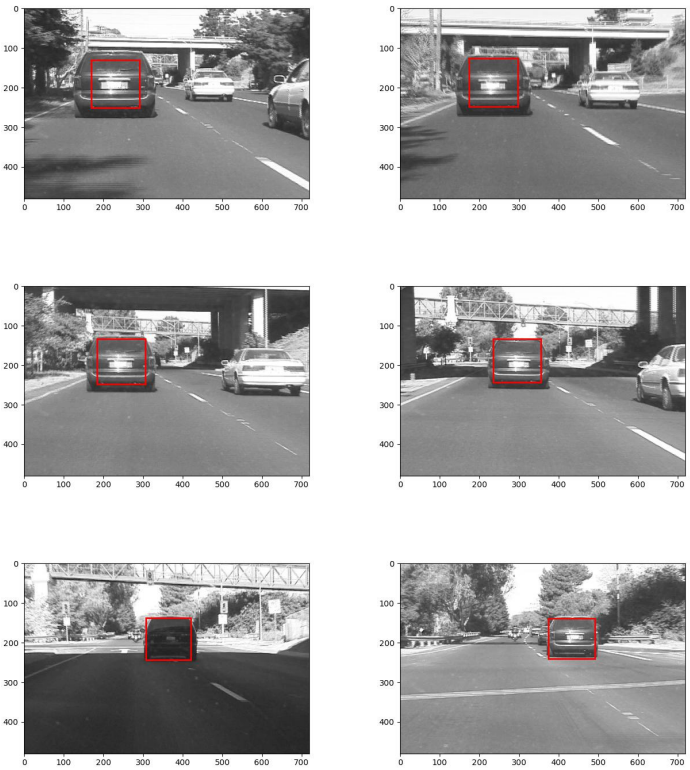

Figure 5: LucasKanade with affine.car1 with frame (1,50,100,150,200,250)

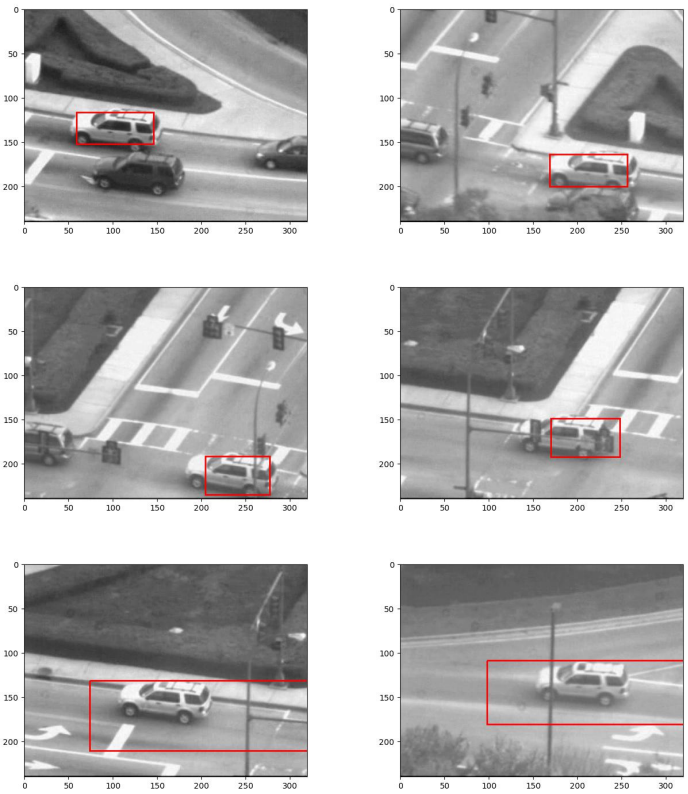

Figure 6: LucasKanade with affine.car2 with frame (1,54,97,135,172,269)

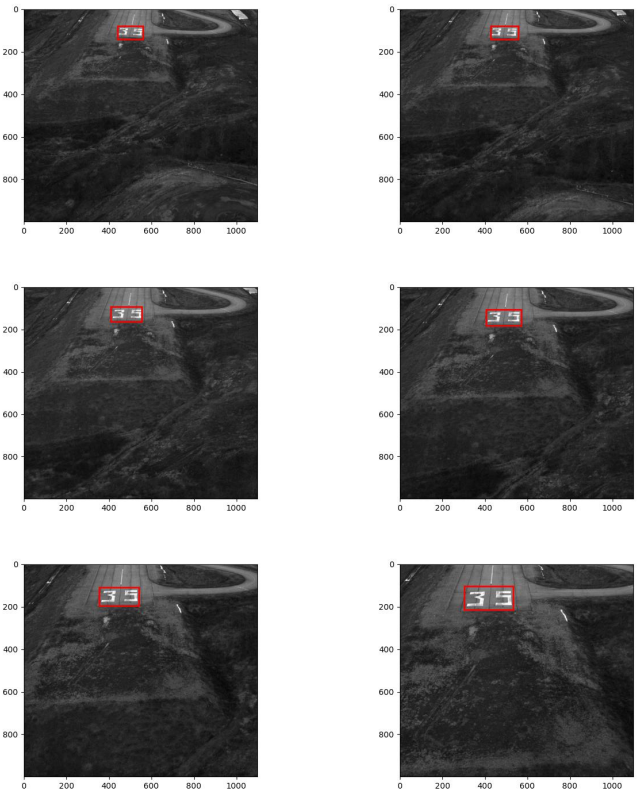

Figure 7: Inverse Compositionalland with frame (1,10,20,30,40,50)

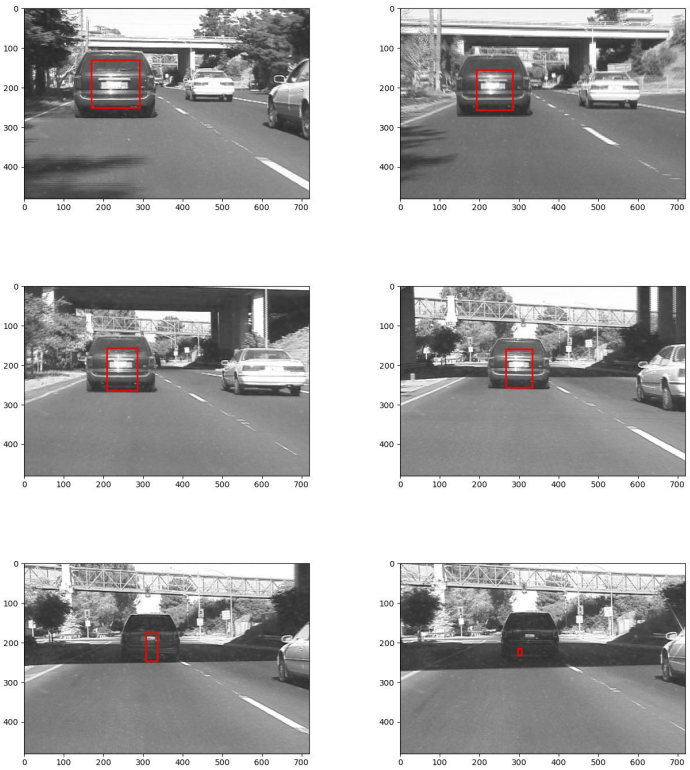

Figure 8: Inverse Compositional car1 with frame (1,50,100,150,167,173)

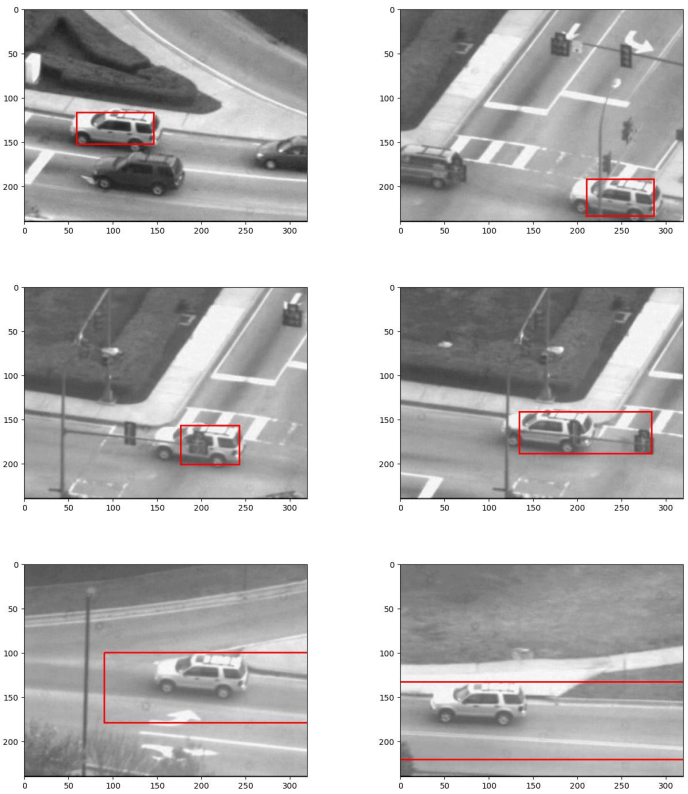

Figure 9: Inverse Compositional car2 with frame (1,87,126,147,256,410)