CS 83: Computer Vision

Optical Flow

Amittai Siavava

03/03/2024

Abstract

This project implements Lucas Kanade, Lucas Kanade Affine, and Inverse Compositional algorithms for tracking optical flow, helping keep track of objects in a video.

Credit Statement

I worked on this PSET alone, with reference to class notes and the three references listed in the assignment. [1, 2, 3]

Contents

1. Theory Questions

Problem 1.

Assuming the affine warp model $\mathbf{W} = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$, derive the expression for the Jacobian matrix $\mathbf{J}$ in terms of the warp parameters $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5, p_6]^\top$.

The Jacobian matrix $\mathbf{J}$ is the matrix of partial derivatives of the warp $\mathbf{W}$ with respect to the warp parameters $\mathbf{p}$. We have

$$\mathbf{W} = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{W}}{\partial p_1} & \frac{\partial \mathbf{W}}{\partial p_2} & \frac{\partial \mathbf{W}}{\partial p_3} & \frac{\partial \mathbf{W}}{\partial p_4} & \frac{\partial \mathbf{W}}{\partial p_5} & \frac{\partial \mathbf{W}}{\partial p_6} \end{bmatrix}$$

We can compute the partial derivatives of $\mathbf{W}$ with respect to the warp parameters $\mathbf{p}$ as follows:

$$\frac{\partial \mathbf{W}}{\partial p_1} = \begin{bmatrix} x \\ 0 \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_2} = \begin{bmatrix} 0 \\ x \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_3} = \begin{bmatrix} y \\ 0 \end{bmatrix},$$

$$\frac{\partial \mathbf{W}}{\partial p_4} = \begin{bmatrix} 0 \\ y \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_5} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_6} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Therefore, the Jacobian matrix $\mathbf{J}$ is:

$$\mathbf{J} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

Note: Since the last row of $\mathbf{W}$ is always $[0, 0, 1]$ and therefore leaves the x-coordinate (in homogeneous coordinates) unchanged, we do not need to include the partial derivatives of the last row of $\mathbf{W}$ with respect to the warp parameters $\mathbf{p}$ since the last row would be all zeros.

Problem 2.

Find the computational complexity (Big O notation) for the initialization step (pre-computing $\mathbf{J}$ and $\mathbf{H}^{-1}$) and for each runtime iteration (Equation 13) of the Matthews-Baker method:

$$\Delta \mathbf{p}^* = \mathbf{H}^{-1}\mathbf{J}^\top [\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p})) - \mathbf{T}]$$

Express your answers in terms of n, m, and p, where

(i) n is the number of pixels in the template $\mathbf{T}$,

(ii) m is the number of pixels in an input image $\mathbf{I}$, and

(iii) p is the number of parameters used to describe the warp $\mathbf{W}$.

How does this compare to the runtime of the regular Lucas-Kanade method?

Matthews-Baker method:

1. **Pre-computing J and H^{-1}:**
   - J is the Jacobian of the image warp with respect to the parameters, which involves computing the gradient at each pixel in the template T for each parameter. This gives a computational complexity of $O(np)$.
   - The inversion of the Hessian matrix H^{-1} has a complexity of $O(p^3)$ using standard matrix inversion methods such as LU-decomposition.
   - In total, the computational complexity of the initialization step is $O(np + p^3)$.
2. **Runtime Iteration (Equation 13):**
   - Computing $[I(W(x;p)) - T]$ involves warping points from the input image ($O(np)$ since we only warp the template region of the input image) and then computing the difference from the template, which has a complexity of $O(n)$, for a total of $O(np)$.
   - The computation of $J^T[I(W(x;p)) - T]$ involves multiplying a $p \times n$ matrix with an $n \times 1$ vector, which gives complexity of $O(np)$.
   - The matrix-vector multiplication $H^{-1}J^T[I(W(x;p)) - T]$ has a computational complexity of $O(p^2)$ since it involves multiplying a $p \times p$ matrix with a $p \times 1$ vector.
   - In total, this gives a computational complexity of $O(np + p^2)$. Given that $n \gg p > 1$, this complexity is dominated by $O(np)$.

Lucas-Kanade method:

In comparison, the regular Lucas-Kanade method has a computational complexity of $O(np^2 + p^3)$ per iteration. This is larger than the complexity of the Matthews-Baker method because the Matthews-Baker method is able to factor out the computation of the Jacobian and Hessian matrixes out of the iteration, so they are only computed once in the initialization step.

3. Video Tracking Results: Lucas Kanade

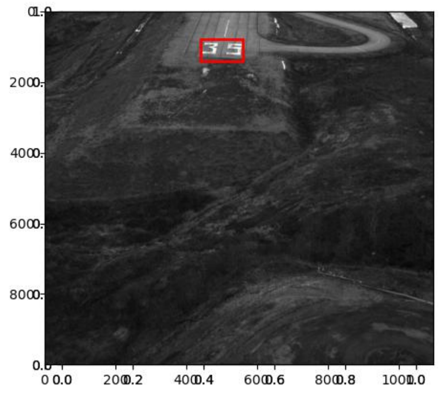

Figure 1. Frame 1

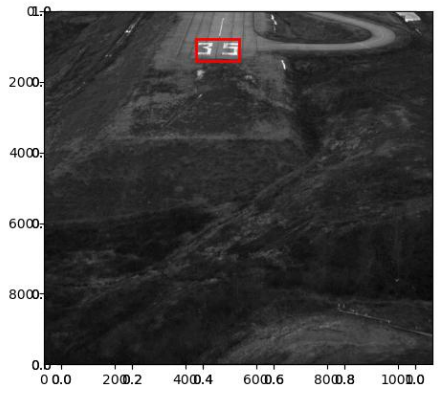

Figure 2. Frame 10

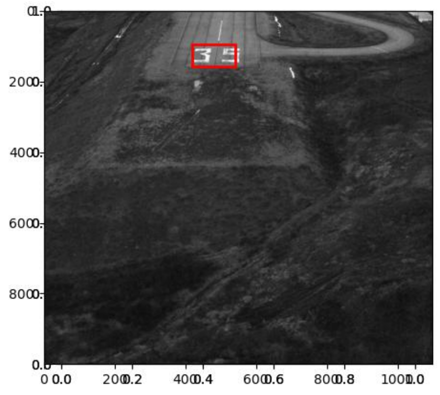

Figure 3. Frame 20

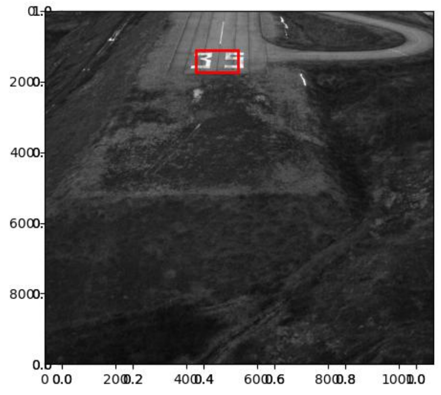

Figure 4. Frame 30

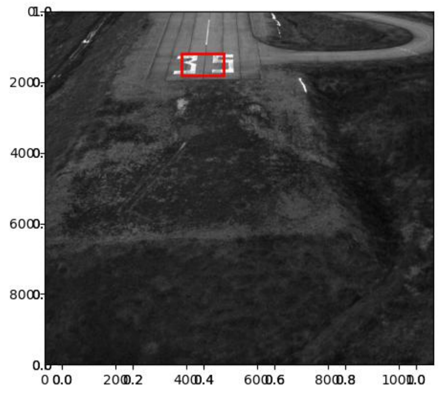

Figure 5. Frame 40

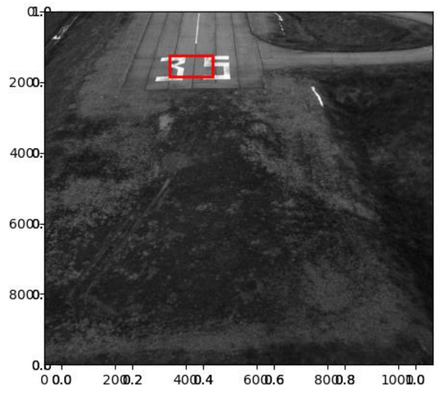

Figure 6. Frame 49

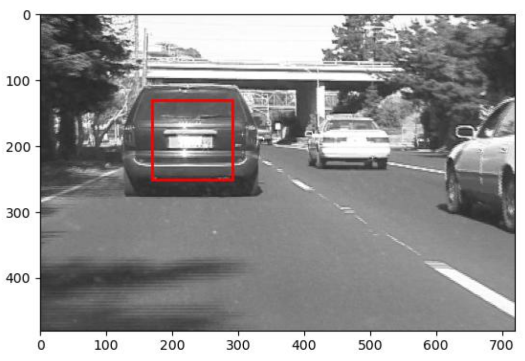
FIGURE 7. Frame 1

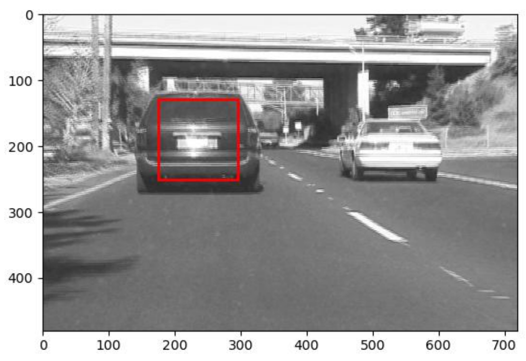
FIGURE 8. Frame 50

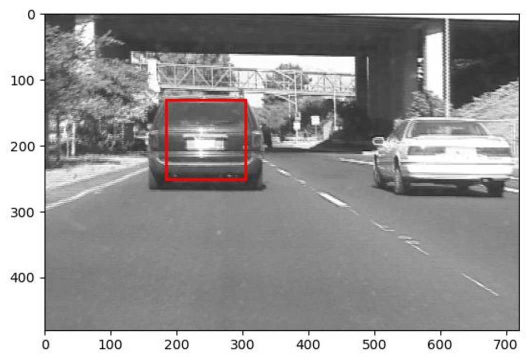
FIGURE 9. Frame 100

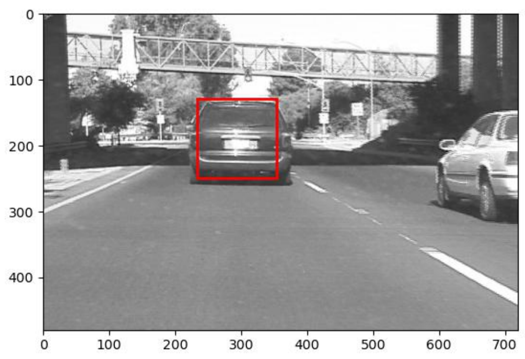
FIGURE 10. Frame 150

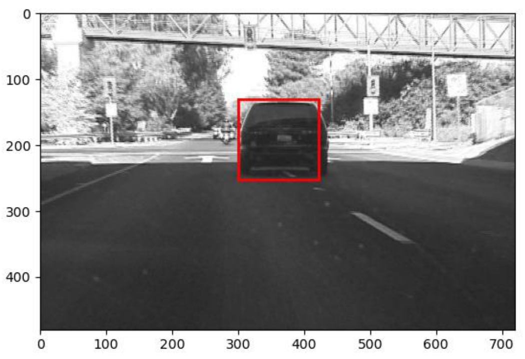
FIGURE 11. Frame 200

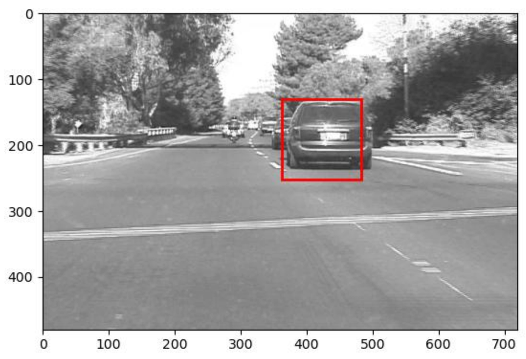
FIGURE 12. Frame 250

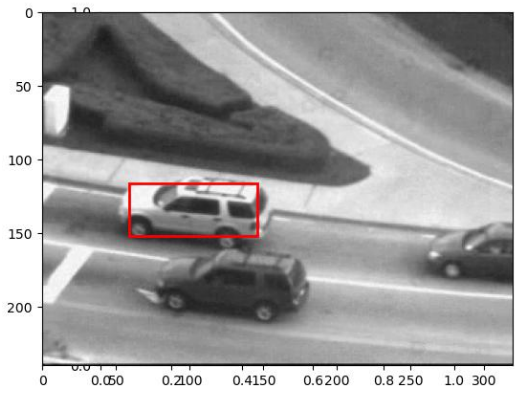

FIGURE 13. Frame 1

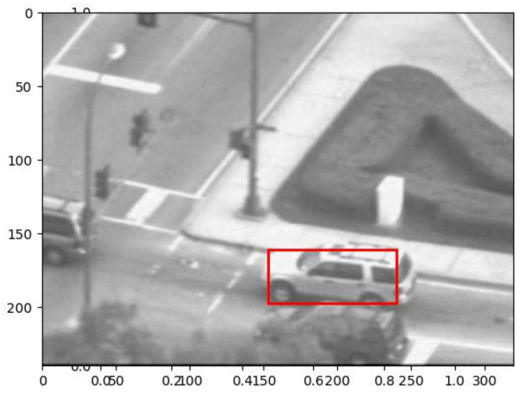

FIGURE 14. Frame 50

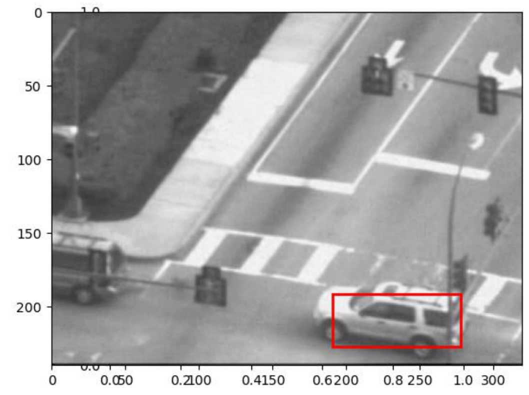

FIGURE 15. Frame 100

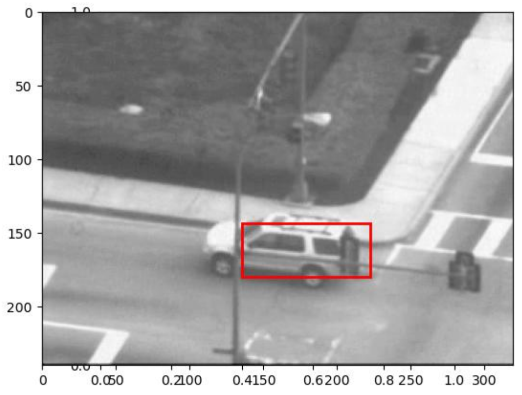

FIGURE 16. Frame 150

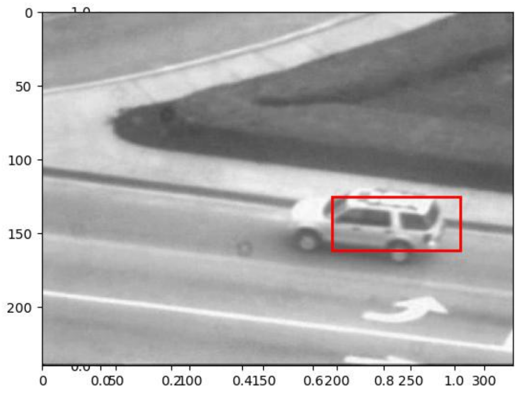

FIGURE 17. Frame 200

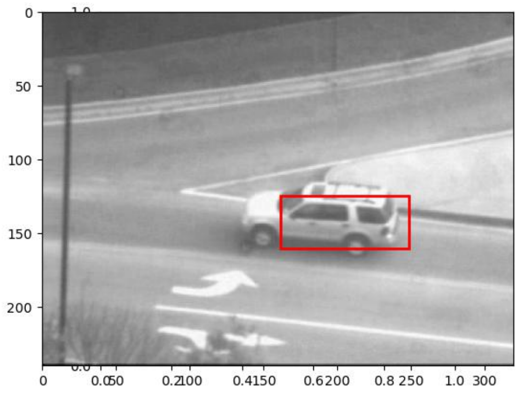

FIGURE 18. Frame 250

4. Video Tracking Results: Lucas Kanade Affine

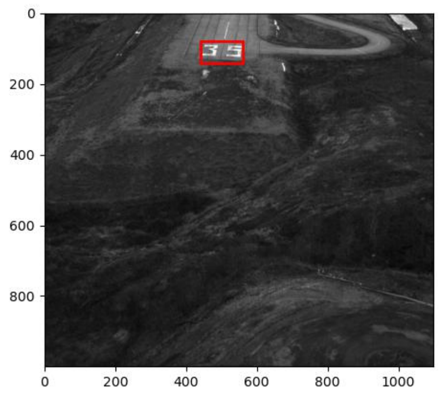

Figure 19. Frame 1

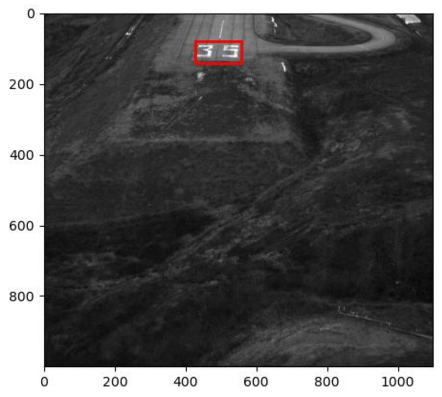

Figure 20. Frame 10

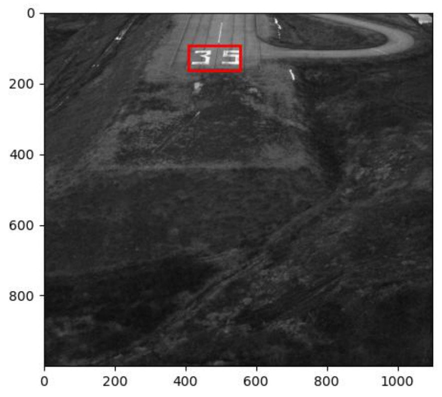

Figure 21. Frame 20

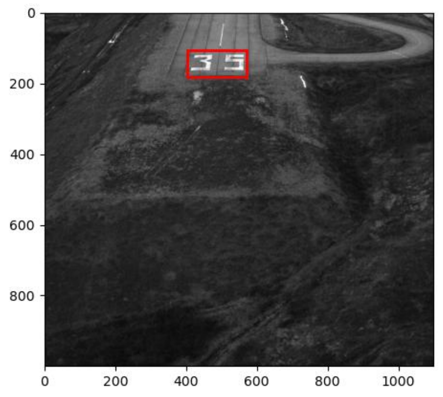

Figure 22. Frame 30

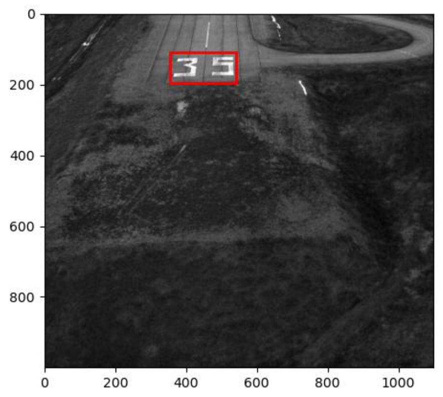

Figure 23. Frame 40

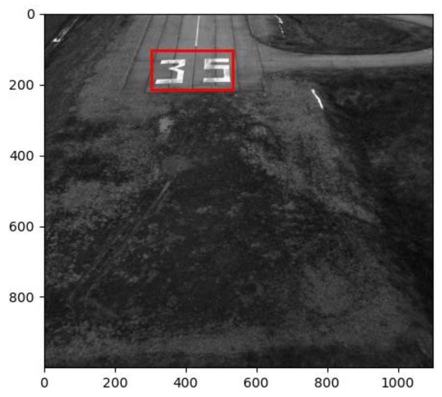

Figure 24. Frame 49

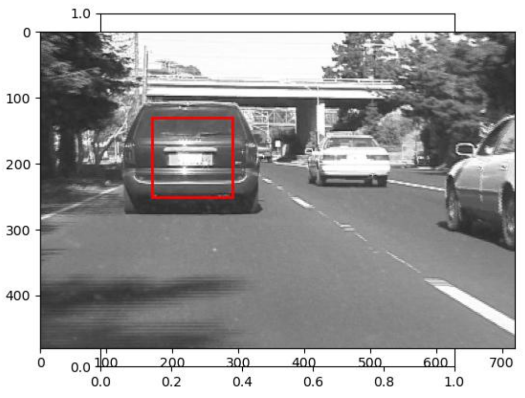

FIGURE 25. Frame 1

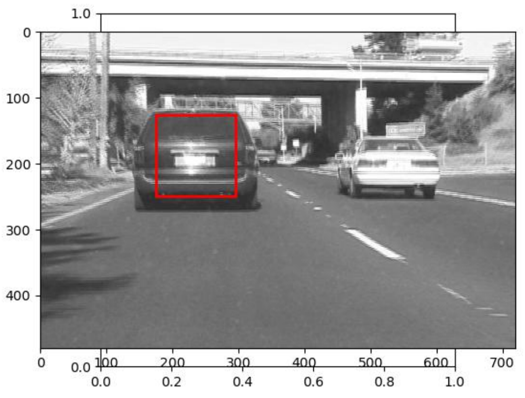

FIGURE 26. Frame 50

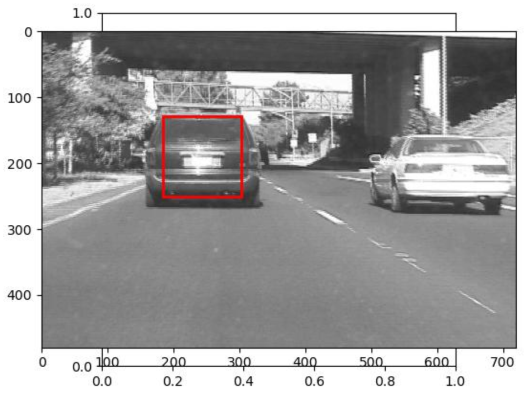

FIGURE 27. Frame 100

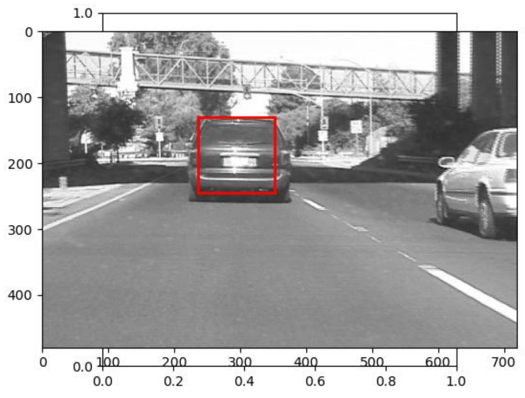

FIGURE 28. Frame 150

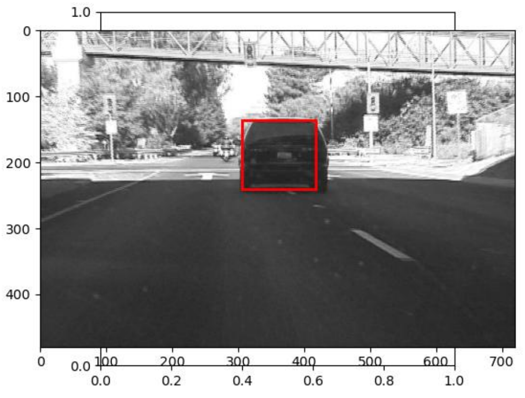

FIGURE 29. Frame 200

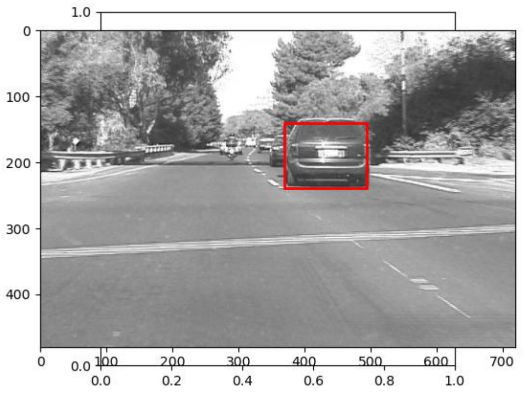

FIGURE 30. Frame 250

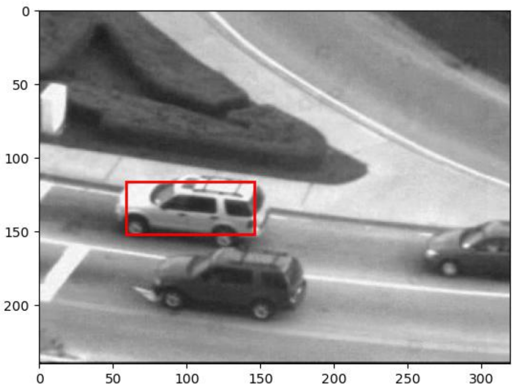

Figure 31. Frame 1

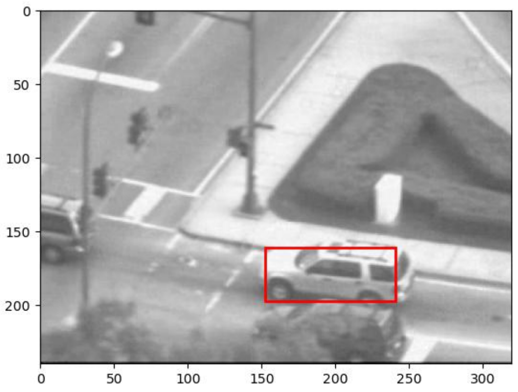

Figure 32. Frame 50

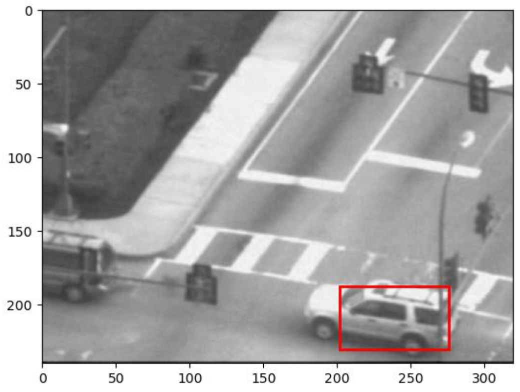

Figure 33. Frame 100

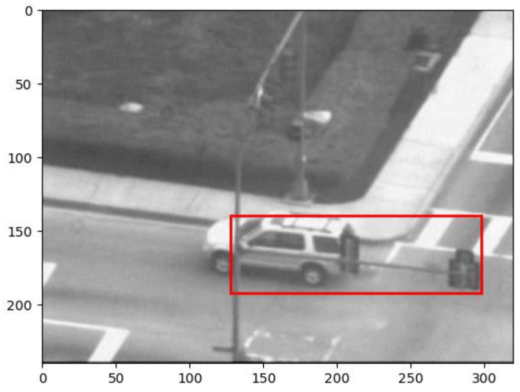

Figure 34. Frame 150

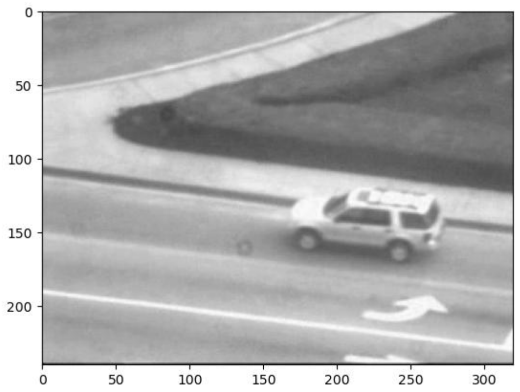

Figure 35. Frame 200

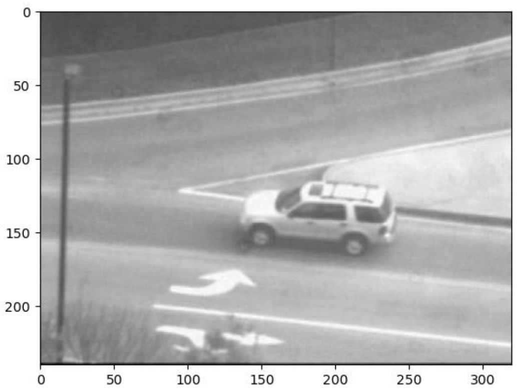

Figure 36. Frame 250

I noticed that the affine transformations (Lucas-Kanade affine and Inverse-Compositional Affine) were thrown off in the third video and seemed to stick to one of the street lights as the car moved. Here's the exact place where that happens in Lucas-Kanade:

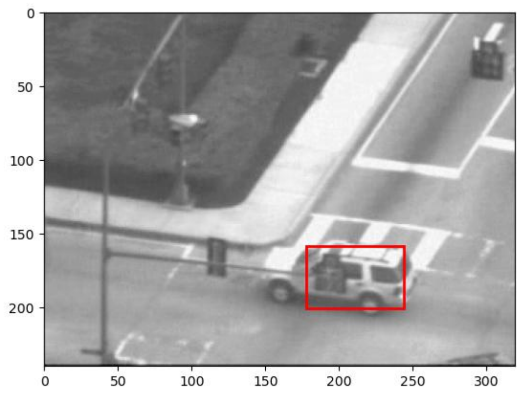

Figure 37. Frame 125

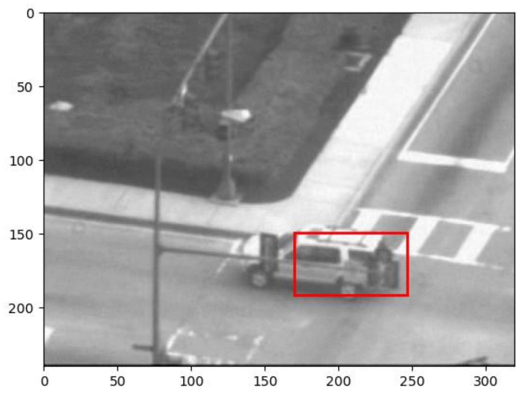

Figure 38. Frame 135

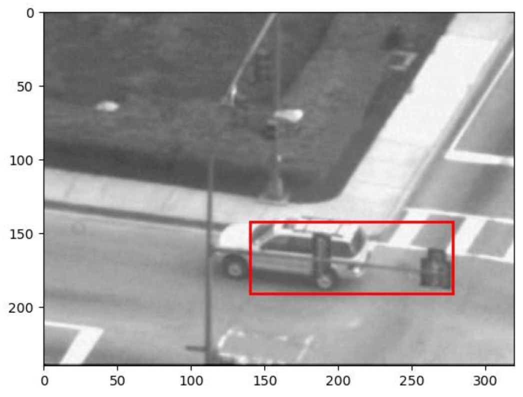

Figure 39. Frame 145

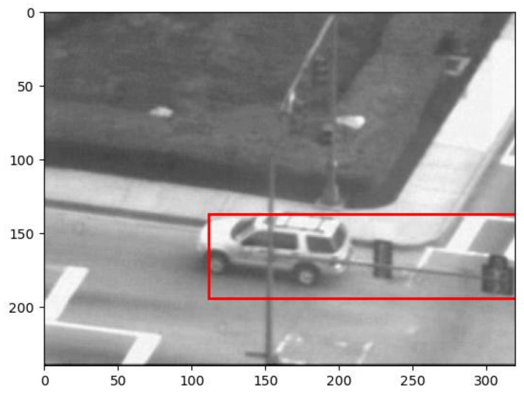

Figure 40. Frame 155

5. Video Tracking Results: Inverse Compositional Affine

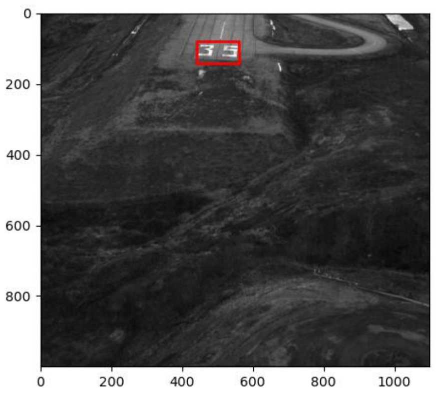

Figure 41. Frame 1

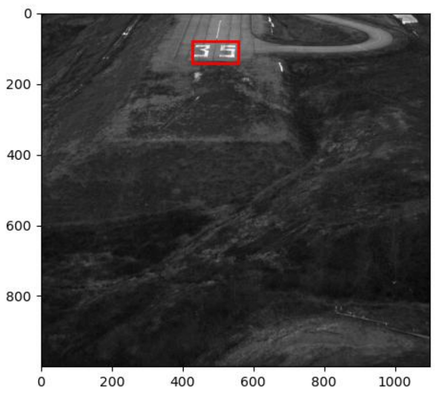

Figure 42. Frame 10

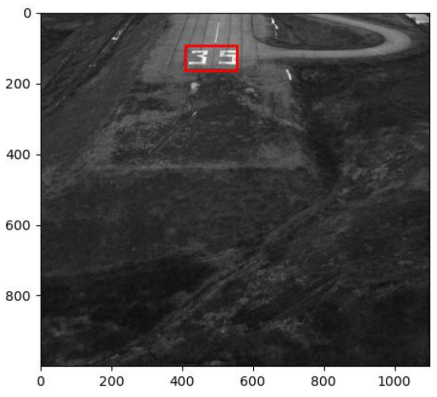

Figure 43. Frame 20

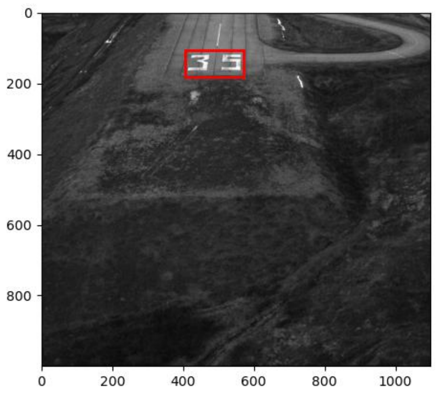

Figure 44. Frame 1

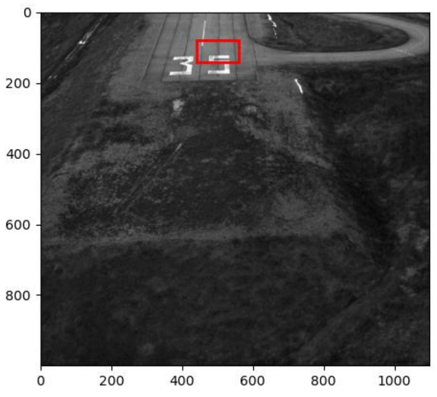

Figure 45. Frame 10

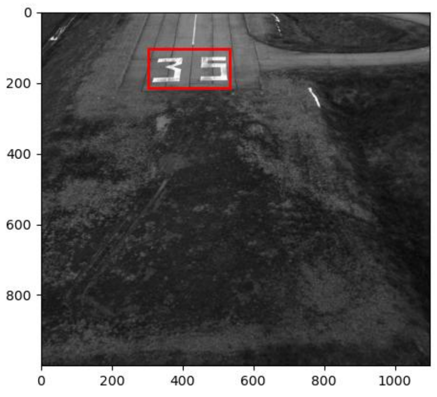

Figure 46. Frame 20

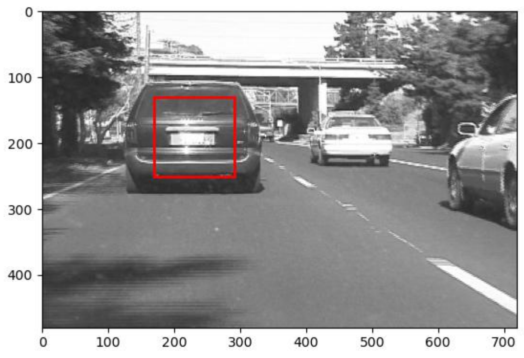

Figure 47. Frame 1

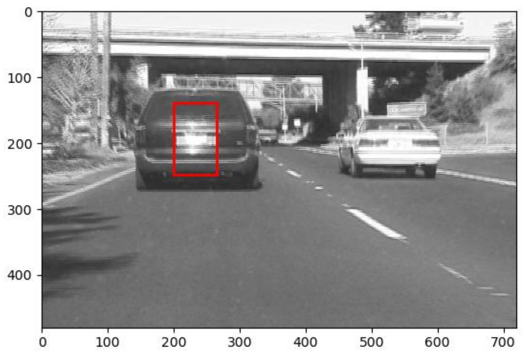

Figure 48. Frame 50

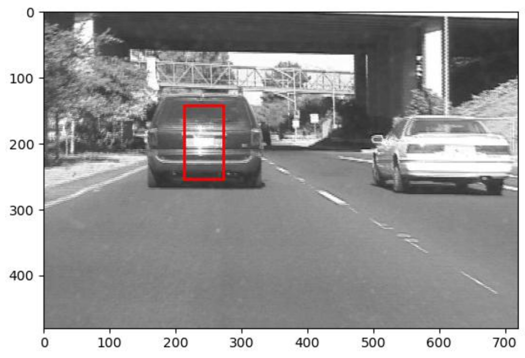

Figure 49. Frame 100

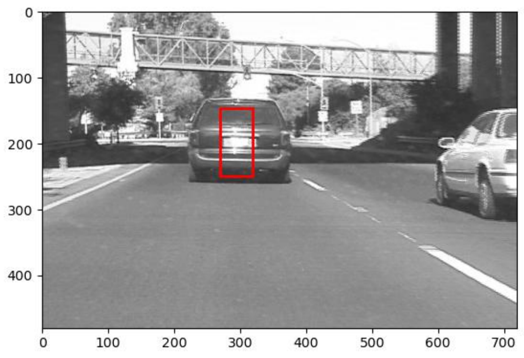

Figure 50. Frame 150

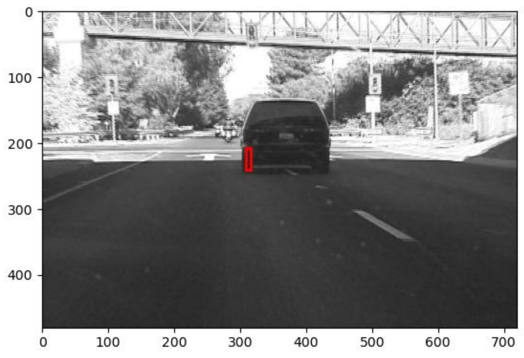

Figure 51. Frame 200

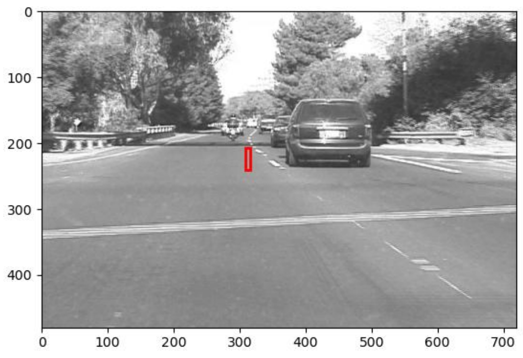

Figure 52. Frame 250

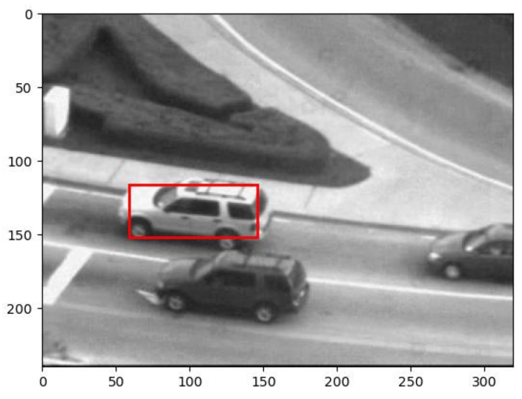

Figure 53. Frame 1

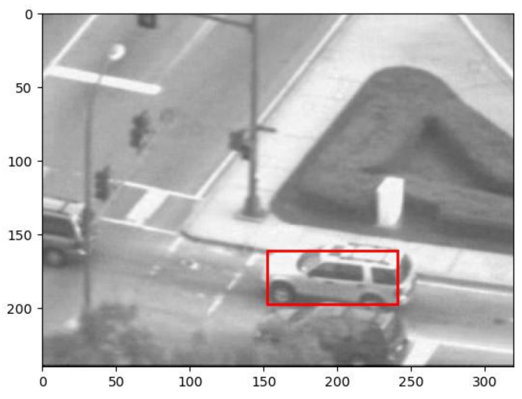

Figure 54. Frame 50

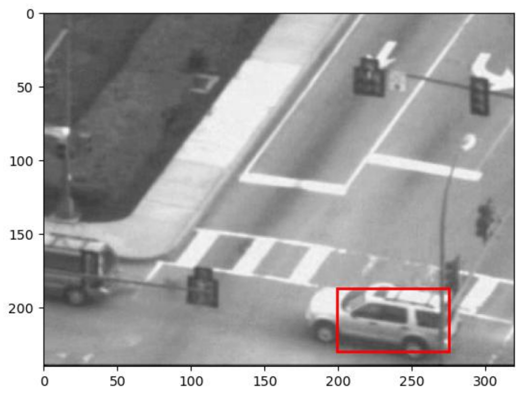

Figure 55. Frame 100

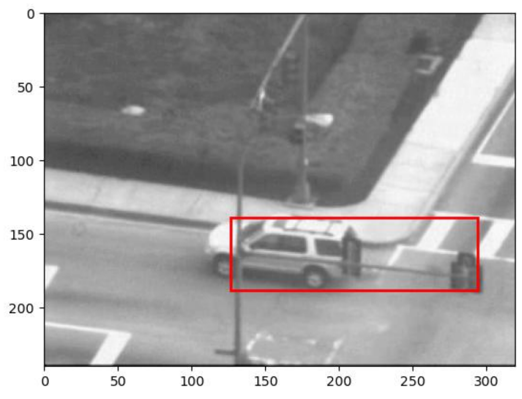

Figure 56. Frame 150

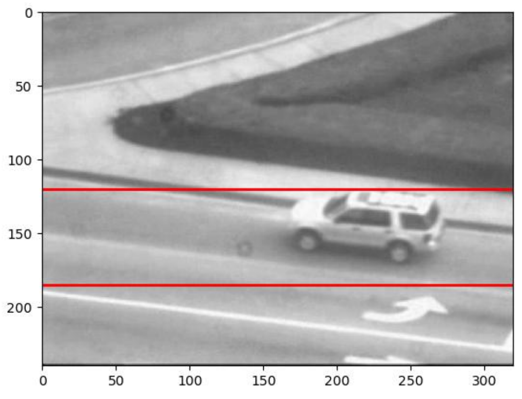

Figure 57. Frame 200

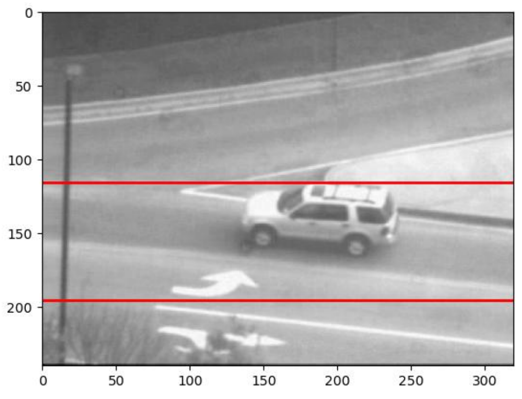

Figure 58. Frame 250

I noticed that the affine transformations (Lucas-Kanade affine and Inverse-Compositional Affine) were thrown off in the third video and seemed to stick to one of the street lights as the car moved. Here's the exact place where that happens in Inverse-Compositional Affine:

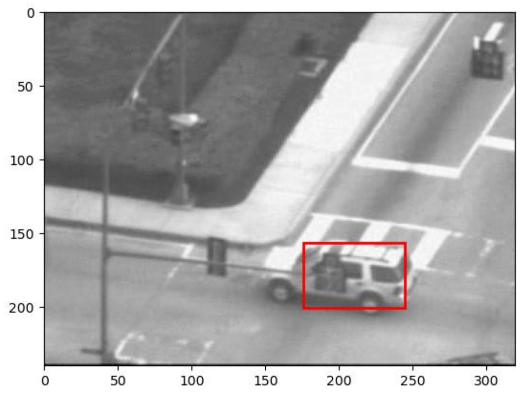

FIGURE 59. Frame 125

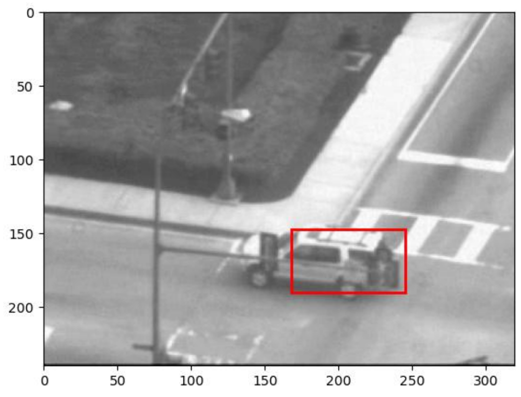

FIGURE 60. Frame 135

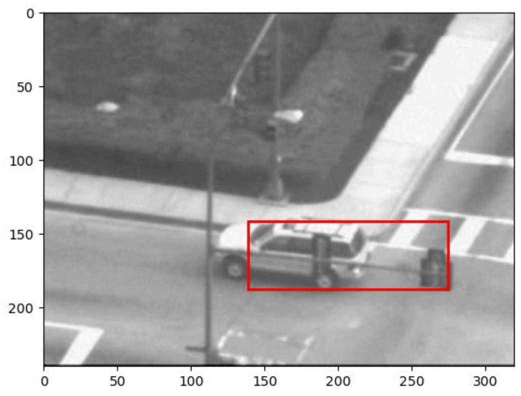

FIGURE 61. Frame 145

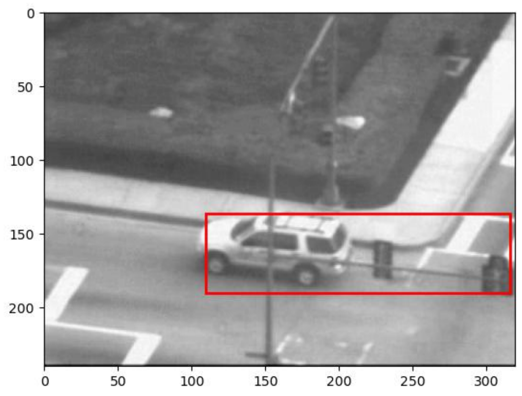

FIGURE 62. Frame 155

Furthermore, I noticed that the bounding box shrinks in `car1` when using Inverse-Compositional Affine, an issue which does not occur with the other two algorithms. These errors could be due to an error in the estimation of $\mathbf{H}^{-1}$

6. Analysis of Results

6.1. **Accuracy of Tracking.** When it worked, Lucas-Kanade Affine seemed most accurate:

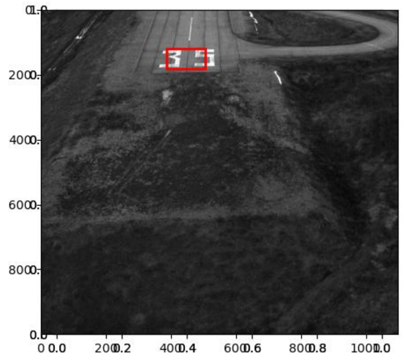

Figure 63. LK, Frame 40

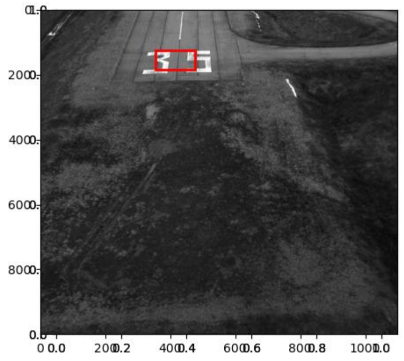

Figure 64. LK, Frame 49

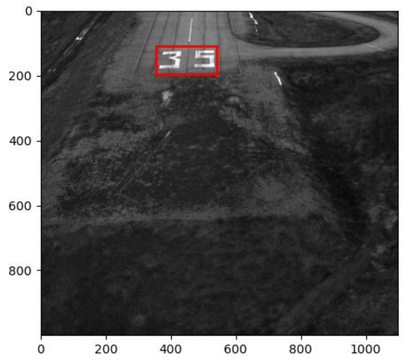

Figure 65. LK Affine, Frame 40

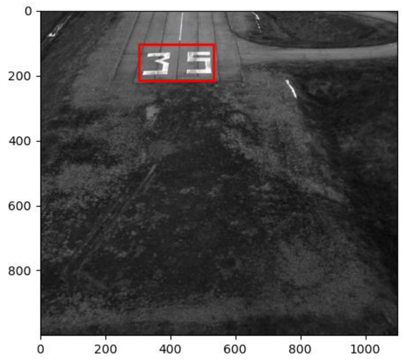

Figure 66. LK Affine, Frame 49

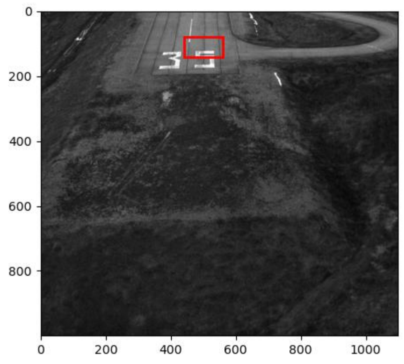

Figure 67. IC Affine, Frame 40

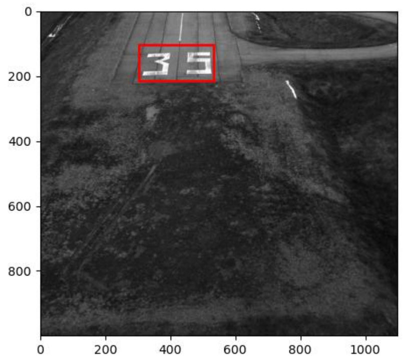

Figure 68. IC Affine, Frame 49

6.2. **Robustness of Tracking.** Both of the affine transformations (Lucas-Kanade affine and Inverse-Compositional Affine) were thrown off in the third video and seemed to stick to one of the street lights as the car moved. Here's the exact place where that happens in Lucas-Kanade (same place for Inverse-Compositional Affine, but not shown here):

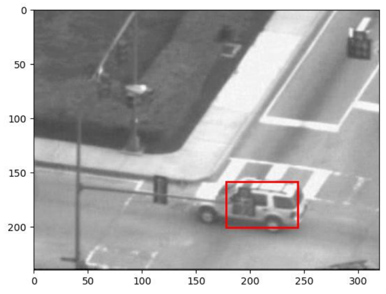

FIGURE 69. Frame 125

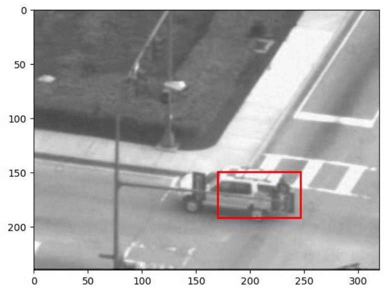

FIGURE 70. Frame 135

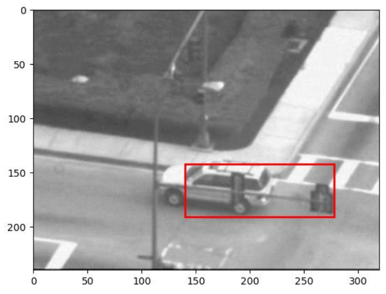

FIGURE 71. Frame 145

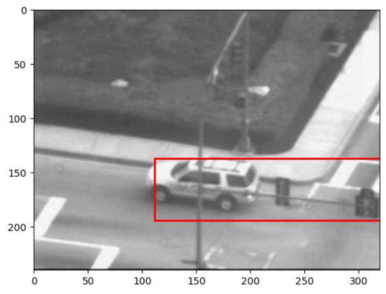

FIGURE 72. Frame 155

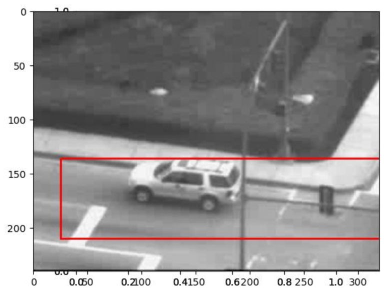

FIGURE 73. Frame 165

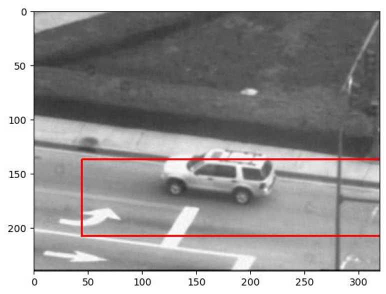

FIGURE 74. Frame 175

This suggests that the Lucas-Kanade affine and Inverse-Compositional affine optical flow algorithms might be more sensitive to changes in the scene than the Lucas-Kanade transformation.

References

[1] Simon Baker et al. *Lucas-Kanade 20 Years On: A Unifying Framework: Part 1*. Tech. rep. CMU-RI-TR-02-16. Carnegie Mellon University, Robotics Institute, 2002.

[2] Simon Baker et al. *Lucas-Kanade 20 Years On: A Unifying Framework: Part 2*. Tech. rep. CMU-RI-TR-03-35. Carnegie Mellon University, Robotics Institute, 2003.

[3] Jean-Yves Bouguet. *Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the algorithm*. Tech. rep. Intel Corporation, 2001.