Computer Vision, 16720A - Homework #3

Neeraj Basu
neerajb@andrew.cmu.edu

October 22, 2020

1 Q 1.1 - Lucas-Kanade Tracking

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

This term represents the Jacobian of the of the affine-warped image in respect to the transpose of p where p defines the parameters of our warp.

What is A and b?

The goal of the Lucas-Kanade algorithm is to minimize the sum of squared error between two images, the template and the image and warped back onto the coordinate frame of the template.
Source: https://www.ri.cmu.edu/pub$_f iles/pub3/baker_s imon_2003_3/baker_s imon_2003_3.pdf$

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

Figure 1: Steepest Descent of Image, Gradient times the Jacobian of the Warp

$$b = I_t(x')$$

Figure 2: The Template Image

What conditions must $A^T A$ meet so that a unique solution to Δ_p can be found?
In order for Δ_p to have a unique solution $A^T A$ much be invertible. It's crucial for this condition to be met as we need to invert $A^T A$ in order to compute our Hessian.

2 Q 1.3 - Implement Lucas-Kanade

The results for frame 1, 100, 200, 300 and 400 for the car sequence are as follows:

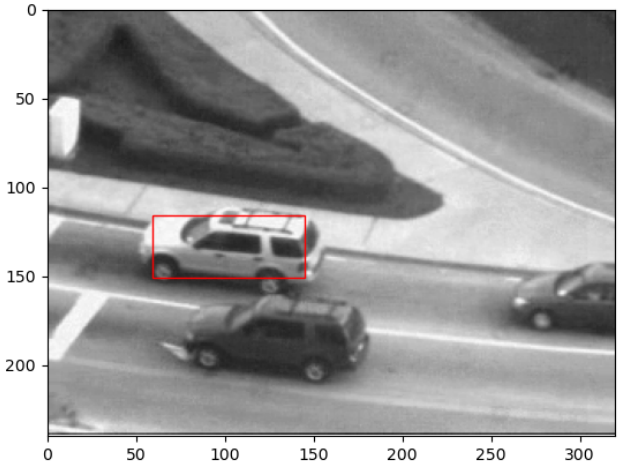

Figure 3: Frame 1

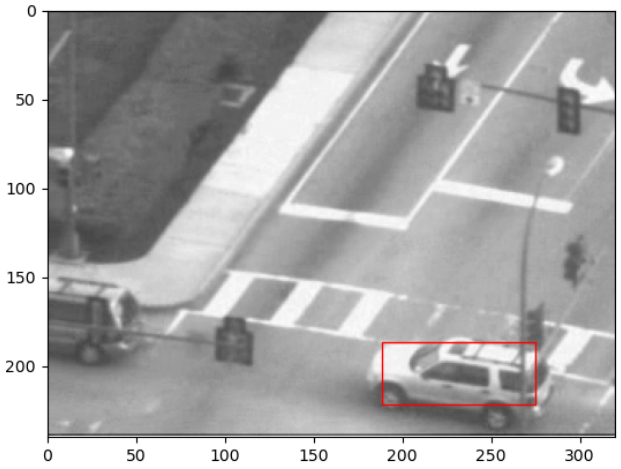

Figure 4: Frame 100

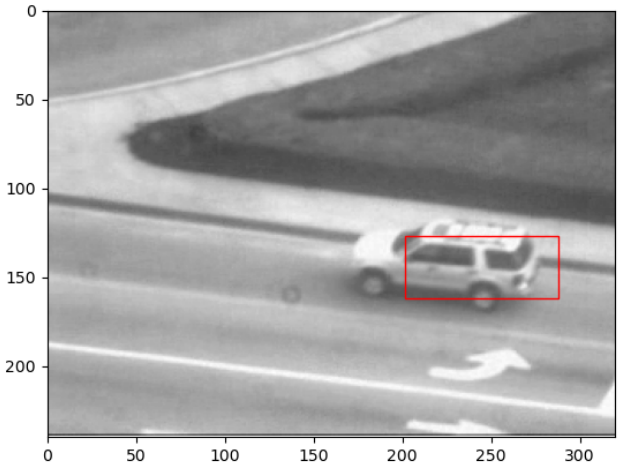

Figure 5: Frame 200

Figure 6: Frame 300

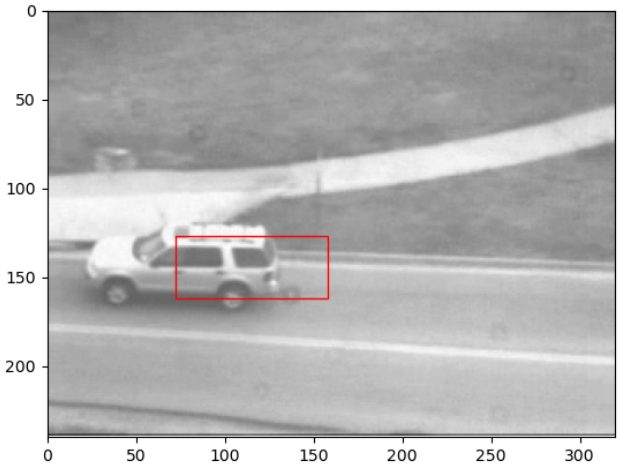

Figure 7: Frame 400

The results for frame 1, 20, 40, 60 and 80 for the girl sequence are as follows:

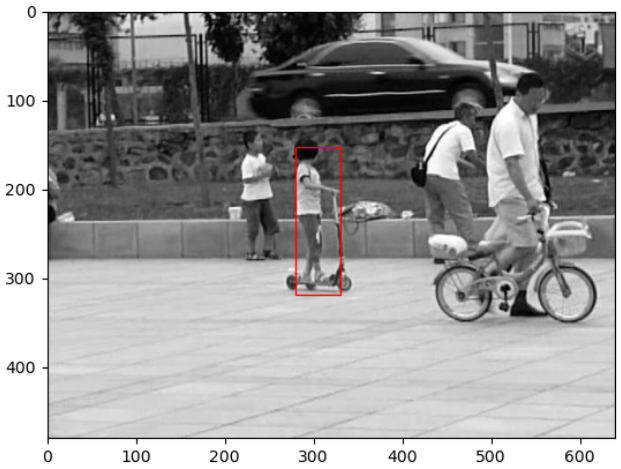

Figure 8: Frame 1

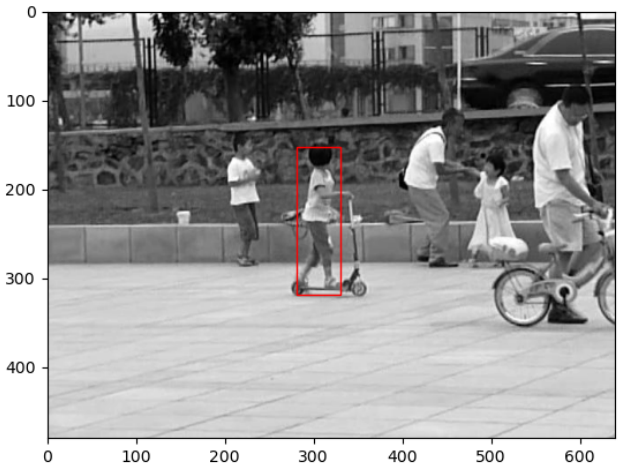

Figure 9: Frame 20

Figure 10: Frame 40

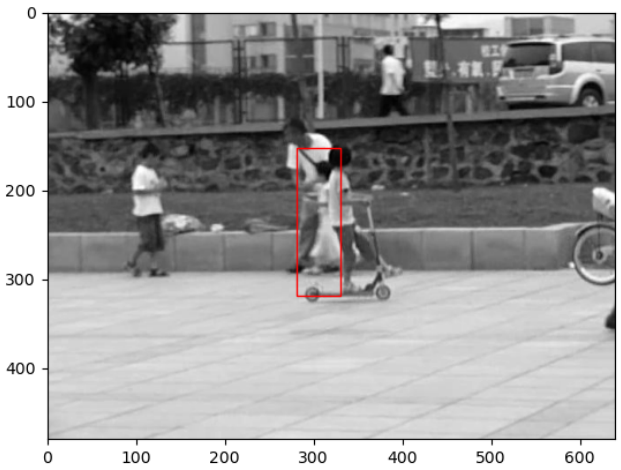

Figure 11: Frame 60

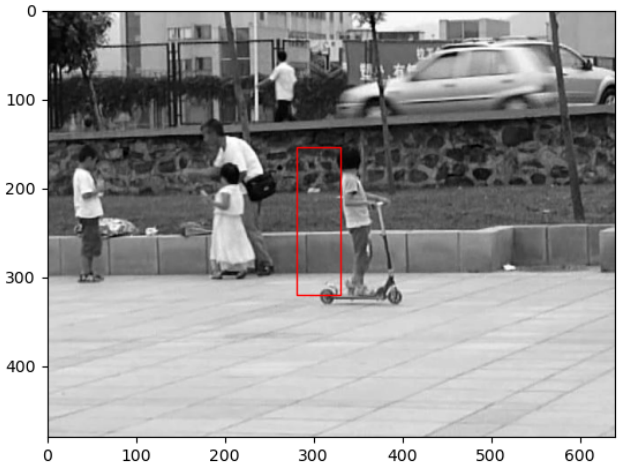

Figure 12: Frame 80

3 Q 1.4 - Implement Lucas-Kanade with Correction

We can see that once we update our template, our bounding box does a significantly better job of tracking the area of interest.

The results for frame 1, 100, 200, 300 and 400 for the car sequence with correction are as follows:

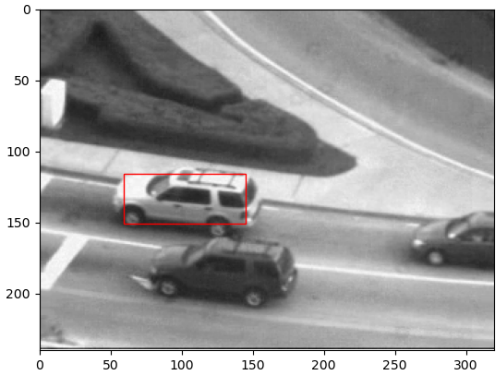

Figure 13: Frame 1

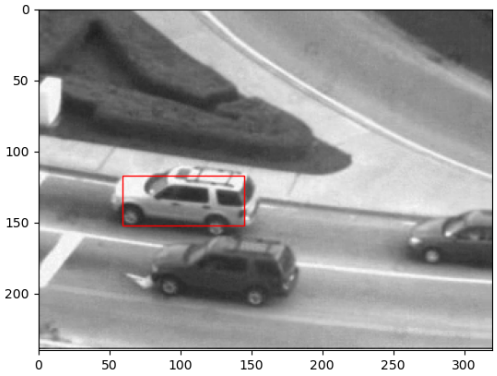

Figure 14: Frame 1 with Correction

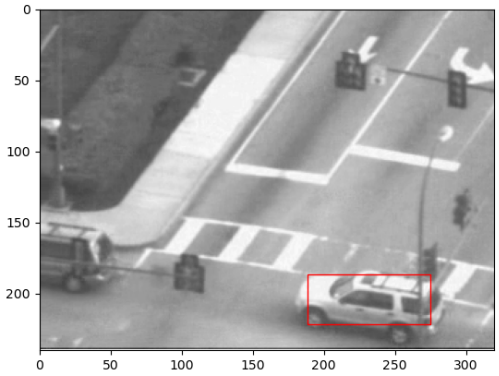

Figure 15: Frame 100

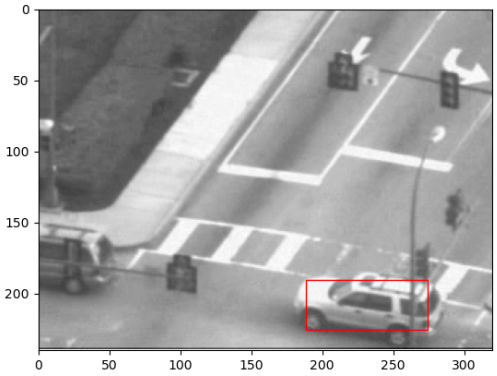

Figure 16: Frame 100 with Correction

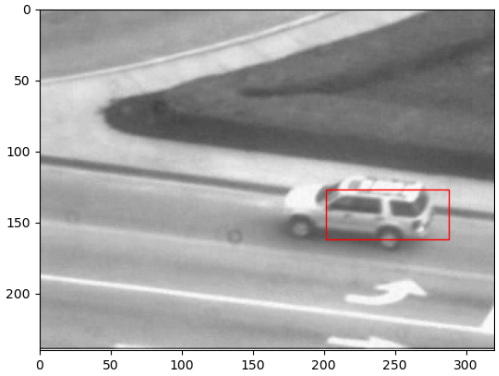

Figure 17: Frame 200

Figure 18: Frame 200 with Correction

Figure 19: Frame 300

Figure 20: Frame 300 with Correction

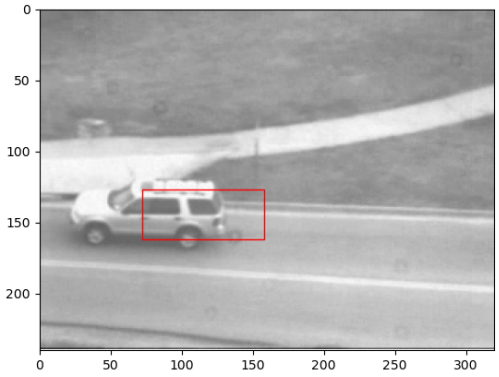

Figure 21: Frame 400

Figure 22: Frame 400 with Correction

The results for frame 1, 20, 40, 60 and 80 for the girl sequence with correction are as follows:

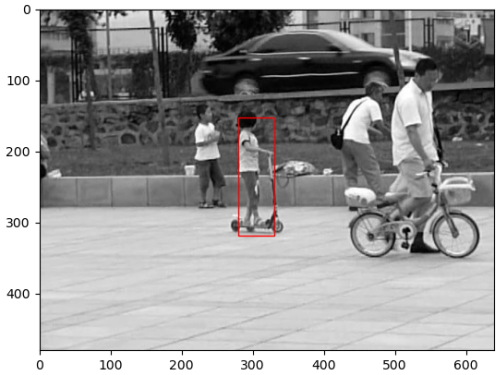

Figure 23: Frame 1

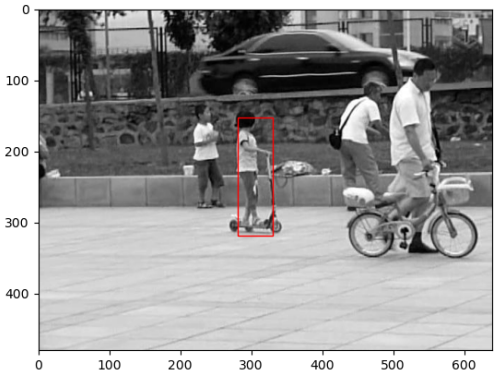

Figure 24: Frame 1 with Correction

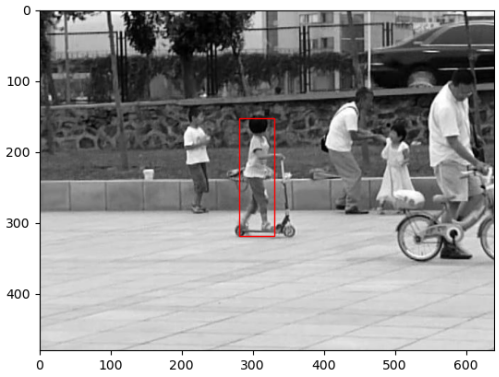

Figure 25: Frame 20

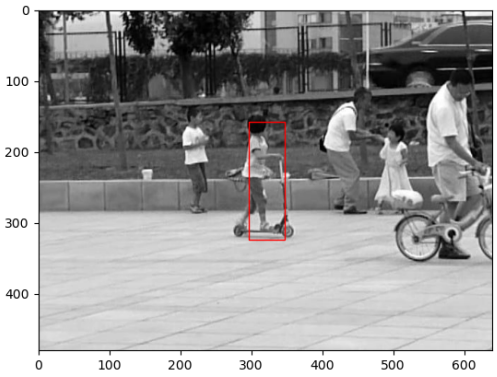

Figure 26: Frame 20 with Correction

Figure 27: Frame 40

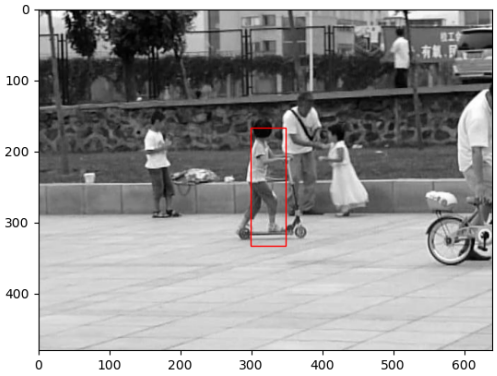

Figure 28: Frame 40 with Correction

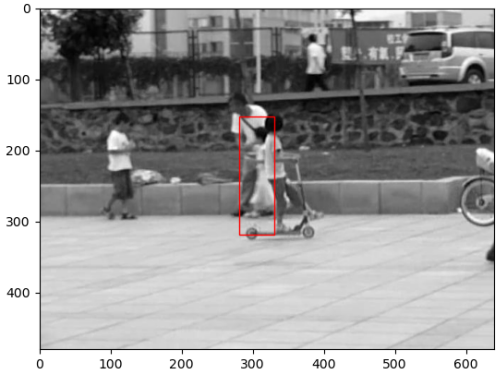

Figure 29: Frame 60

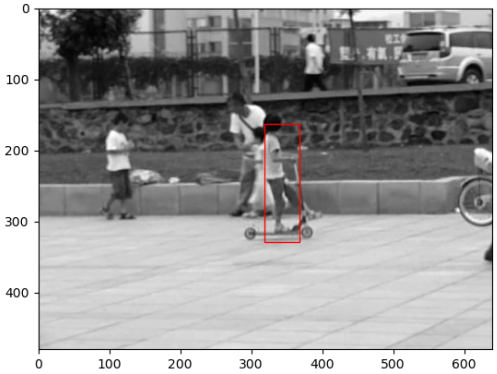

Figure 30: Frame 60 with Correction

Figure 31: Frame 80

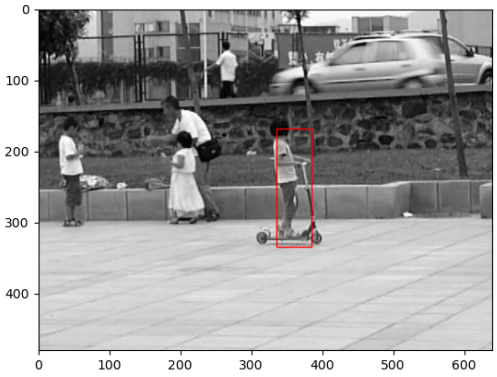

Figure 32: Frame 80 with Correction

4 Q 2.3 - Moving Object Detection

The results for frame 30, 60, 90, 120 for the ant sequence are as follows:

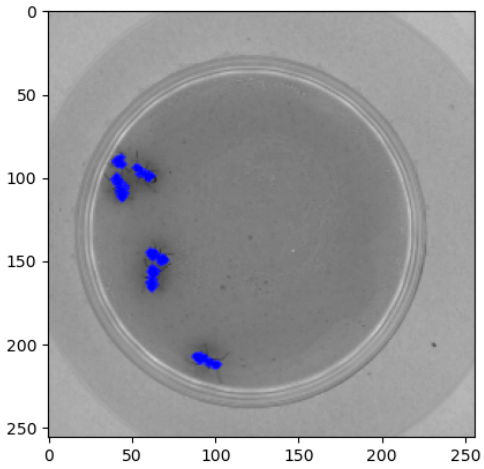

Figure 33: Frame 30

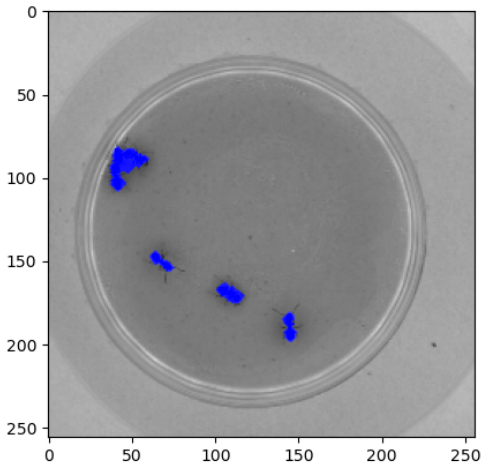

Figure 34: Frame 60

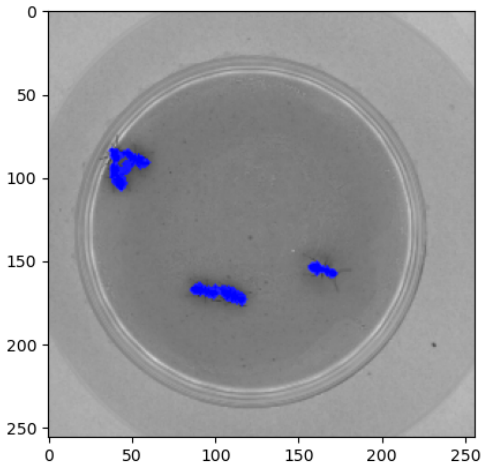

Figure 35: Frame 90

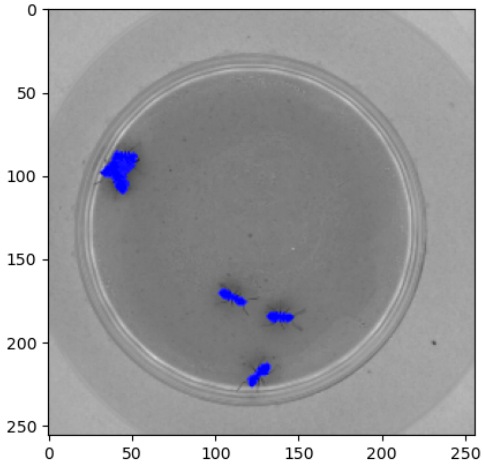

Figure 36: Frame 120

The results for frame 30, 60, 90, 120 for the aerial sequence are as follows:

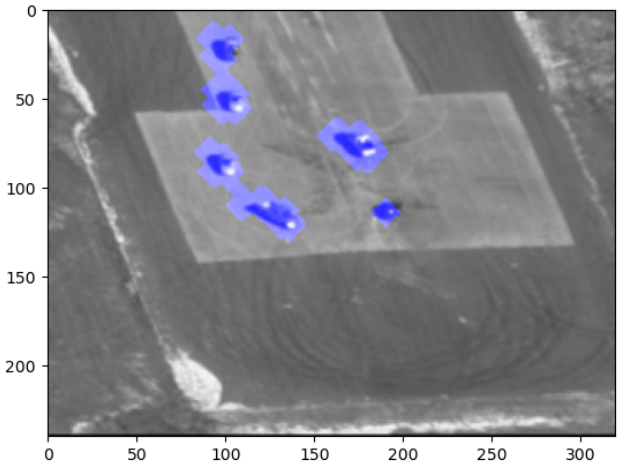

Figure 37: Frame 30

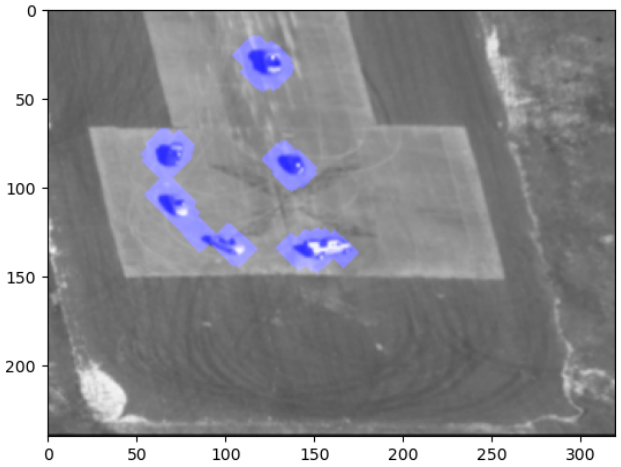

Figure 38: Frame 60

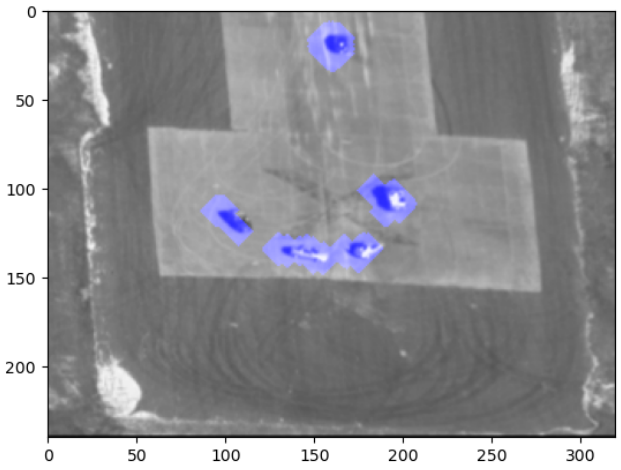

Figure 39: Frame 90

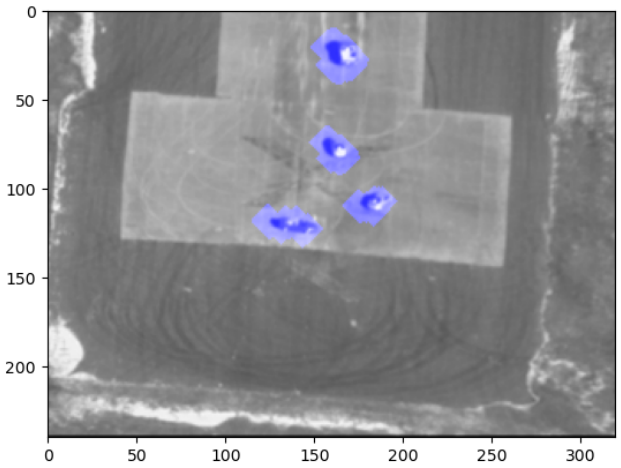

Figure 40: Frame 120

5 Q 3.1 - Inverse Composition

The beauty of the Inverse Compositional algorithm is that we can precompute a number of image parameters prior to entering our Lucas Kanade iteration loop. Since our Jacobian only depends on the x and y pixel coordinates, we can precompute this matrix for each frame. Since we are now looking at the gradient of the template vs. gradient of the image, we can also precompute this outside of the Lucas Kanade Iteration loop. Finally since our Hessian is dependent on the Jacobian and gradient for each pixel, we can calculate our Hessian outside of the loop. By precomputing these parameters, we save significant time on each of our frame calculations. I was able to reduce the speed of each of my frame calculations by nearly half while using Python's time library.