Q1.1)

Given warp function :

$$W(\mathbf{x};\mathbf{p}) \;=\; \mathbf{x} \;+\; \mathbf{p} \;=\begin{bmatrix} w_x \\ w_y \end{bmatrix} \;=\; \begin{bmatrix} x+p_x \\ y+p_y \end{bmatrix}$$

$$\frac{\partial W(x;p)}{\partial p^T} \;=\; \begin{bmatrix} \frac{\partial w_x}{\partial p_x} & \frac{\partial w_x}{\partial p_y} \\ \frac{\partial w_y}{\partial p_x} & \frac{\partial w_y}{\partial p_y} \end{bmatrix} \;=\; \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \;=\; I_{2\mathbf{x}2} \text{ identity matrix}$$

$$p^* \;=\; argmin_p \sum || \, I_{t+1}(x+p) \;-\; I_t(x)||_2^2$$

$$I_{t+1}(x'+\Delta p) \;\approx\; I_{t+1}(x') \;+\; \frac{\partial I_{t+1}(x')}{\partial x'^T}.\frac{\partial W(x;p)}{\partial p^T}.\Delta p$$

$$p^* \;=\; argmin_p \sum || \, I_{t+1}(x') \;+\; \frac{\partial I_{t+1}(x')}{\partial x'^T}.\frac{\partial W(x;p)}{\partial p^T}.\Delta p \;-\; I_t(x)||_2^2$$

$$\Delta p^* \;=\; argmin_{\Delta p} \sum || \, \frac{\partial I_{t+1}(x')}{\partial x'^T}.\frac{\partial W(x;p)}{\partial p^T}.\Delta p \;-\; (I_t(x) \;-\; I_{t+1}(x')) \, ||_2^2$$

$$\Longrightarrow \; p^* \;=\; p + \Delta p^*$$

$$Therefore, \; A \;=\; \frac{\partial I_{t+1}(x')}{\partial x'^T}.\frac{\partial W(x;p)}{\partial p^T} \;;\; b \;=\; I_t(x) \;-\; I_{t+1}(x')$$

$$A \;=\; \frac{\partial I_{t+1}(x')}{\partial x'^T} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \;=\; \frac{\partial I_{t+1}(x')}{\partial x'^T}$$

Therefore, for considering all points :

$$A = \begin{bmatrix} \frac{\partial I_{t+1}(x')}{\partial x'_1} & \frac{\partial I_{t+1}(x')}{\partial y'_1} \\ \frac{\partial I_{t+1}(x')}{\partial x'_2} & \frac{\partial I_{t+1}(x')}{\partial y'_2} \\ . & . \\ . & . \\ \frac{\partial I_{t+1}(x')}{\partial x'_N} & \frac{\partial I_{t+1}(x')}{\partial y'_N} \end{bmatrix} \text{ and } b = \begin{bmatrix} I_t(x_1) - I_{t+1}(x'_1) \\ I_t(x_2) - I_{t+1}(x'_2) \\ . \\ . \\ I_t(x_N) - I_{t+1}(x'_N) \end{bmatrix}$$

$$i.e., \ \Delta p^* = argmin_{\Delta p} \ || A\Delta p - b ||_2^2$$

$$E = || A\Delta p - b ||_2^2$$
$$\Delta p^* = argmin_{\Delta p} E$$

$$\implies \frac{\partial E}{\partial(\Delta p)} = A^T(A\Delta p - b) = 0$$

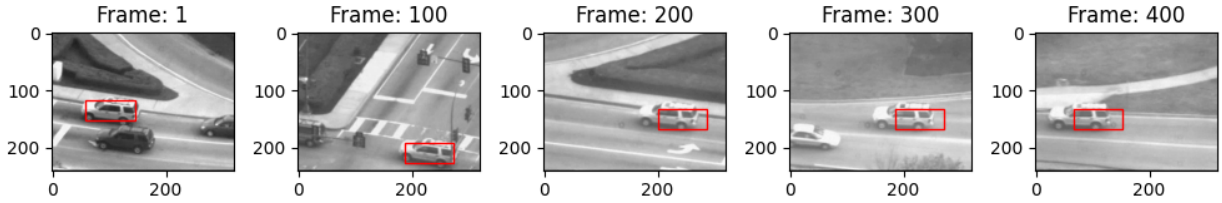
$$\Delta p^* = \left(A^T A\right)^{-1}\left(A^T b\right)$$

Therefore, $A^T A$ has to invertible / non − singular matrix

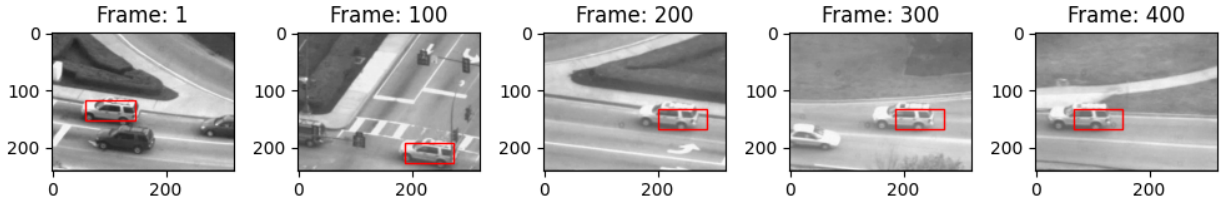
Q1.3)

testCarSequence.py:

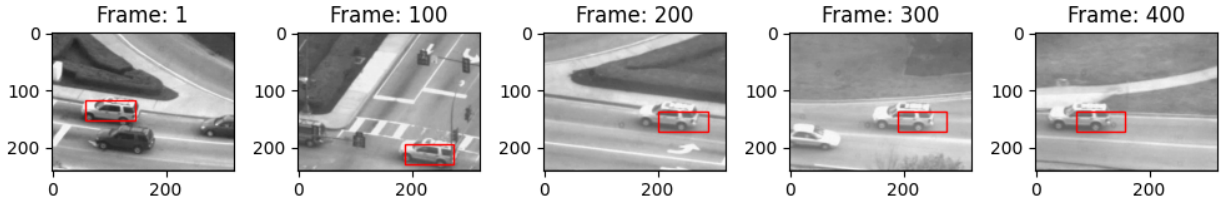
Threshold = 1e-4



Threshold = 1e-2

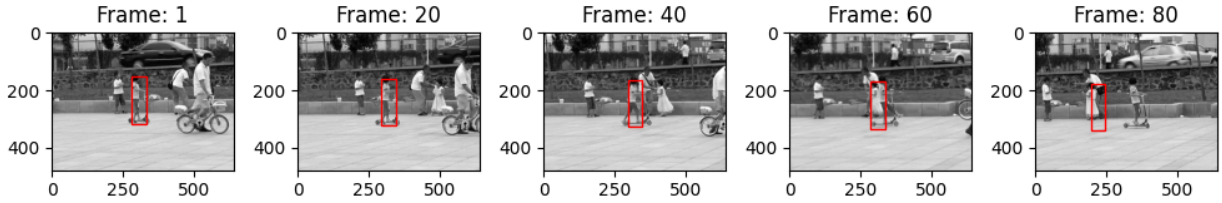


Threshold = 1e-1

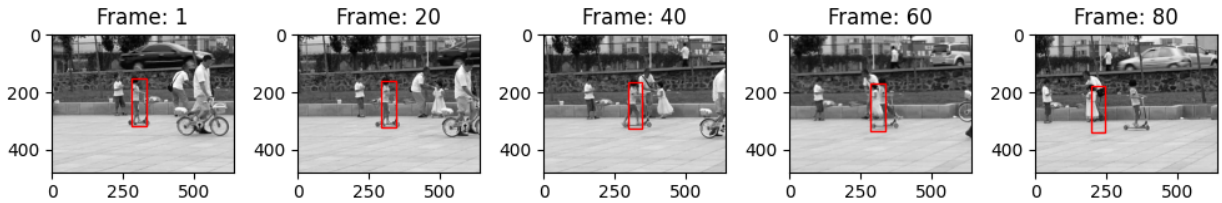


testGirlSequence.py:

Threshold = 1e-4



Threshold = 1e-1

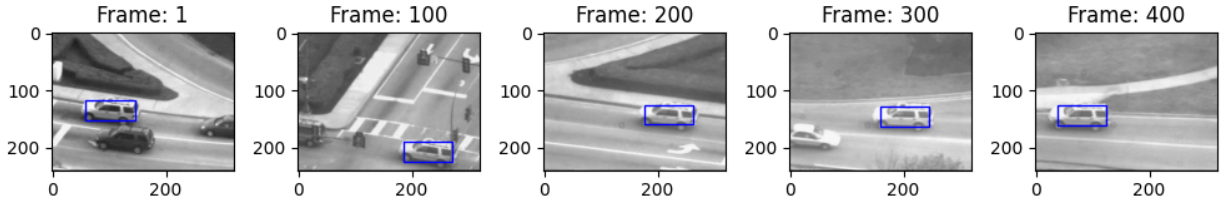


We can see that there is drift in the tracked image in the case of the car. However, in the girl's case, the tracking suddenly shifts from the skateboard boy to the girl after overlap. There isn't much difference in varying the threshold because the drift is due to occlusion.

Q1.4)

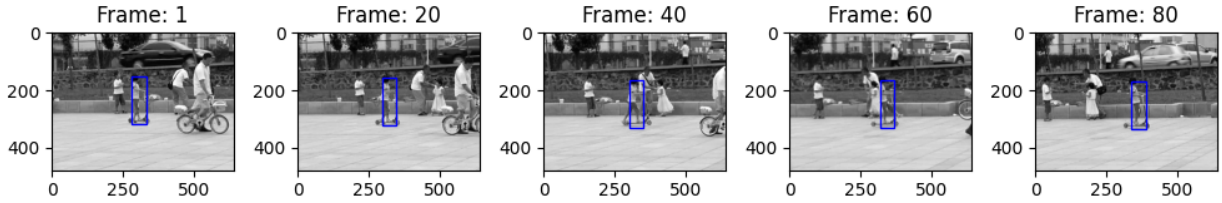
testCarSequenceWithTemplateCorrection.py:

Threshold: 1e-2 || Template_threshold = 3

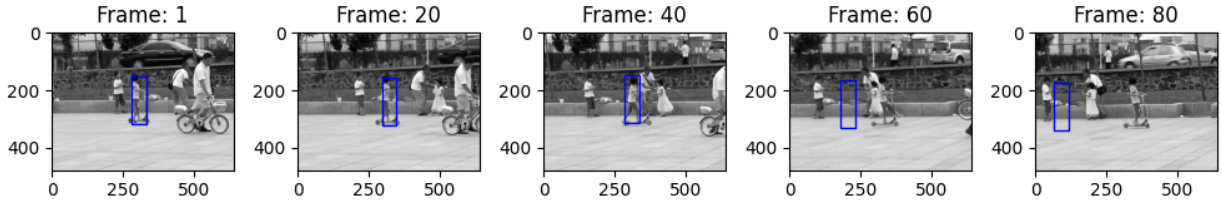


testGirlSequenceWithTemplateCorrection.py:

Threshold: 1e-2 || Template_threshold = 3



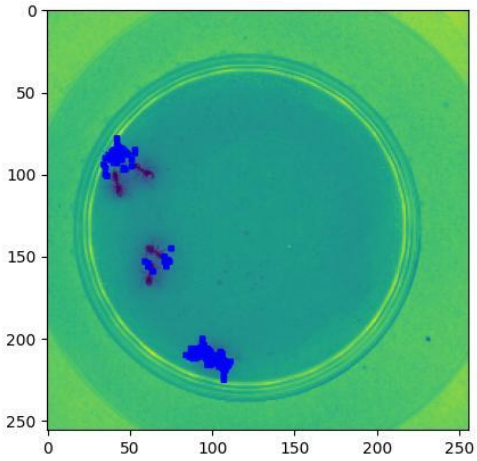
Threshold: 1e-2 || Template_threshold = 5



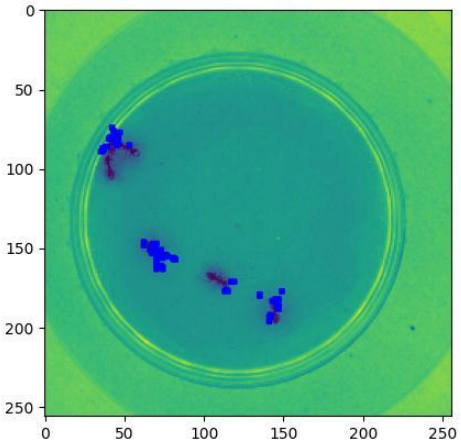
We can observe that there is no drifting now in both cases. However, for large values of tolerance (such as tolerance=5), the template gets drifted a lot. Small tolerance ensures that the template does not drift.

Q2.3)

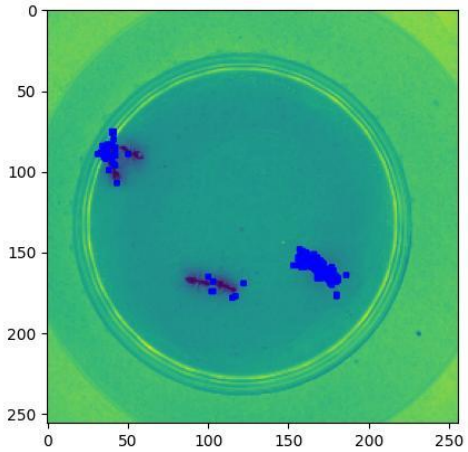
Ant Sequence:
Parameters Used:
Num_iters = 1000 || Threshold = 1e-1 || Tolerance = 0.1



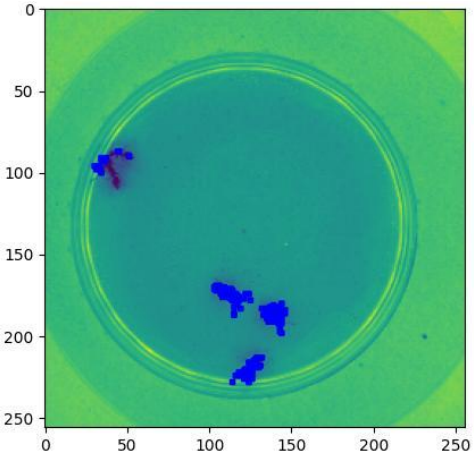
Frame 30



Frame 60



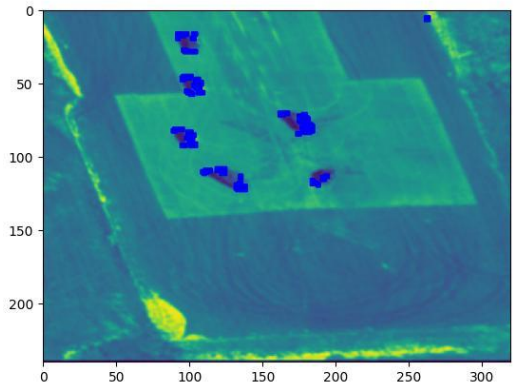
Frame 90



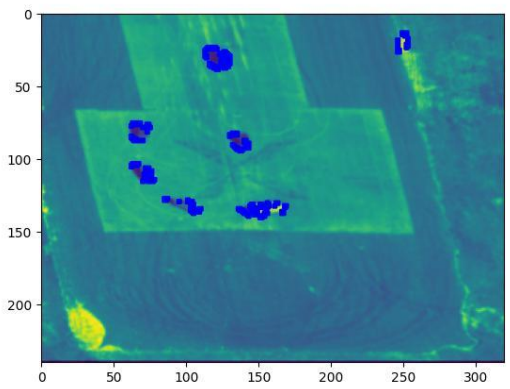
Frame 120

The output from the affine_transform does not account for the border pixels which are beyond the bounds. Therefore, the change captured is nullified using making it zero.

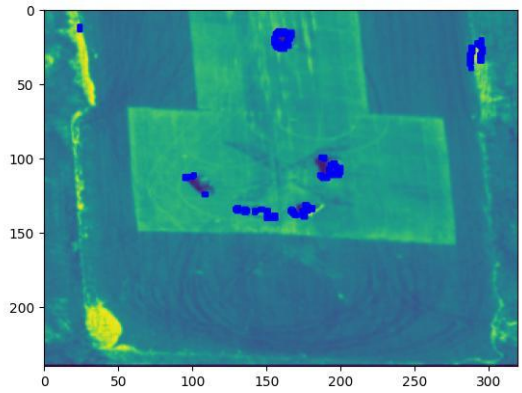
Aerial Sequence:
Parameters Used:
Num_iters = 1000 || Threshold = 1e-1 || Tolerance = 0.2



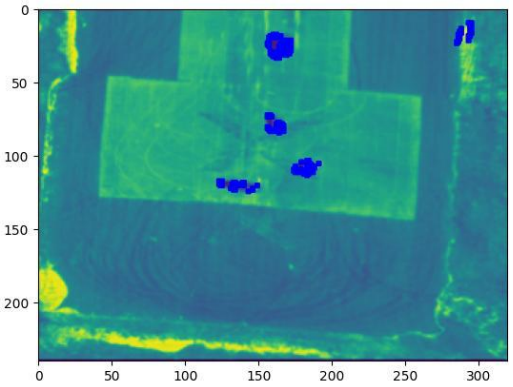
Frame 30



Frame 60



Frame 90



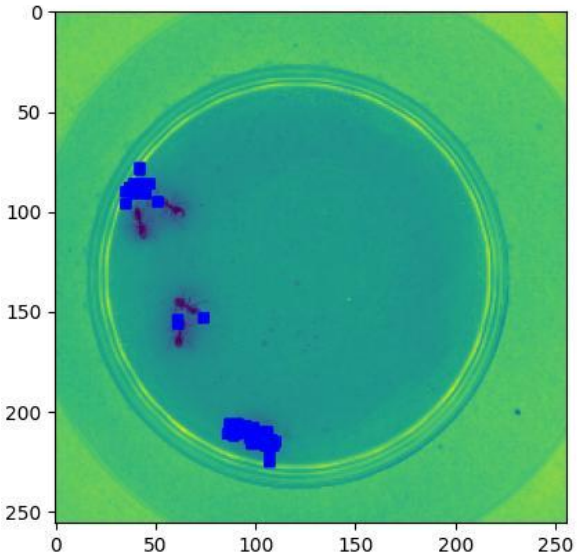
Frame 120

Q3.1)

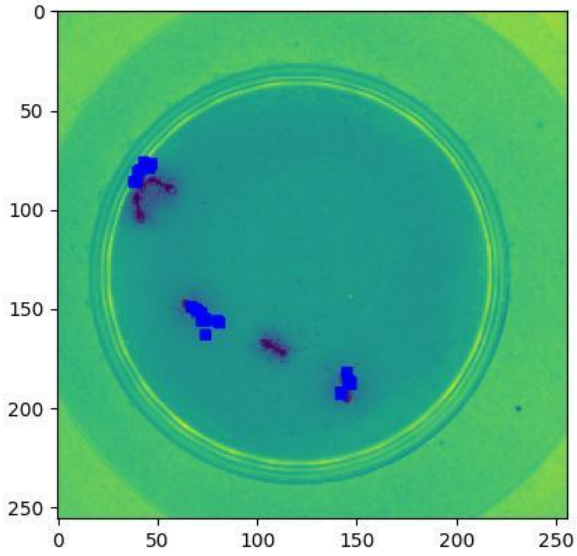
testAntSequence

Parameters Used:

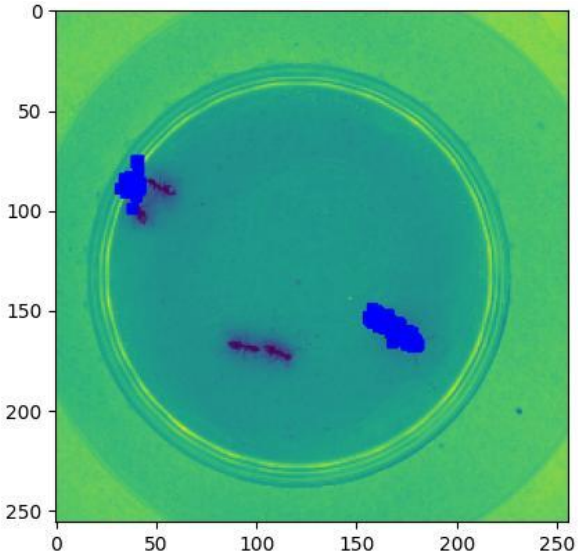
Num_iters = 1000 || Threshold = 1e-1 || Tolerance = 0.15



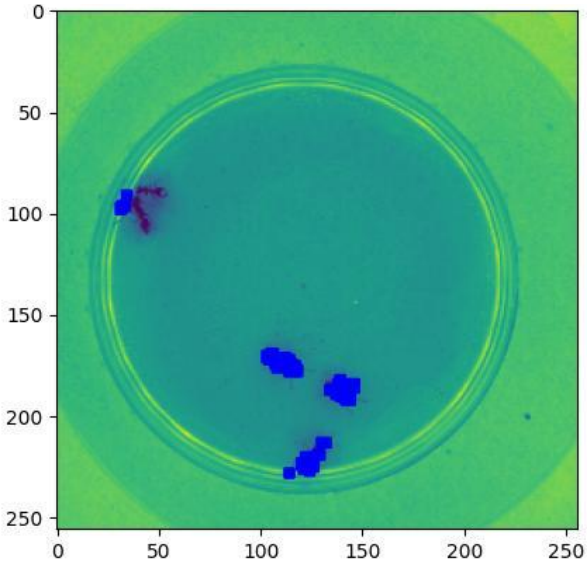
Frame 30



Frame 60

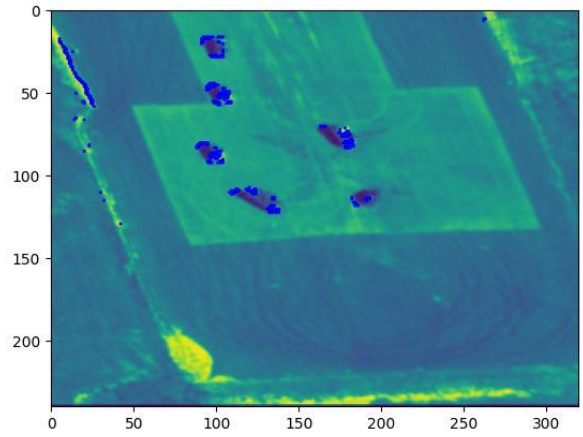


Frame 90

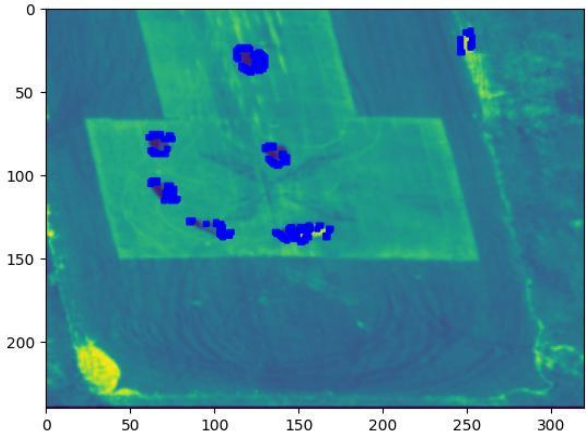


Frame 120

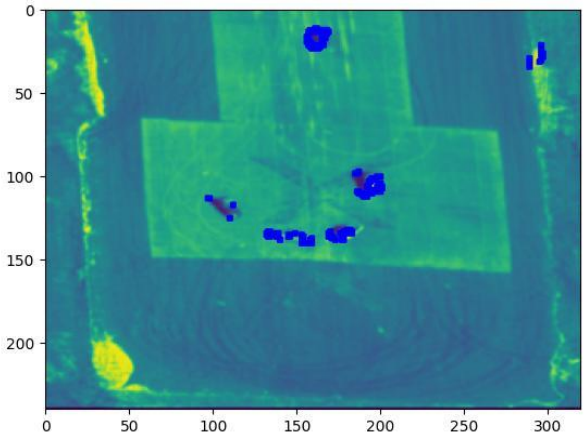
testAerialSequence
Parameters Used:
Num_iters = 1000 || Threshold = 1e-1 || Tolerance = 0.2



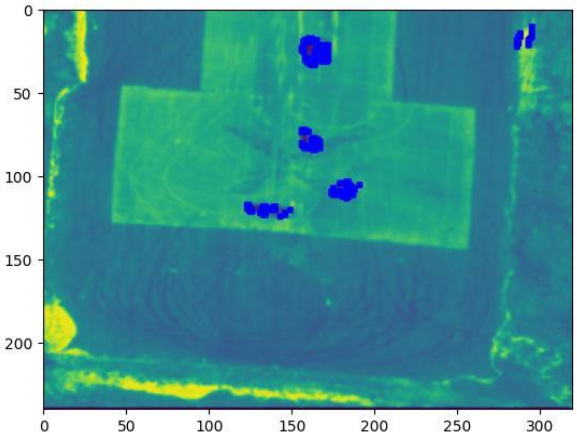
Frame 30



Frame 60



Frame 90



Frame 120

The time taken for tracking and saving all images in **testAerialSequence** = 11mins
Time taken for **testAerialSequence** in Q2.3 = 31mins

Therefore, there is a speed-up of approximately 3 times.

Inverse Compositional is more efficient and takes less time because the Hessian matrix is computed at the start and does not change for every iteration.

In the classical approach, the gradient changes for every iteration because the warp is changing. As a result, the Hessian needs to be computed at each iteration. Though the Jacobian does not change with iteration, since the gradient changes the term *A = gradient x Jacobian* changes. In the case of Inverse Compositional, this is computed once and is used for constructing Hessian and also the RHS.

The term A = $\left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]$ is computed just once and used for every iteration.

$$H = \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]$$

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