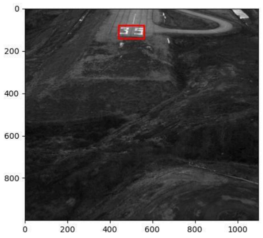
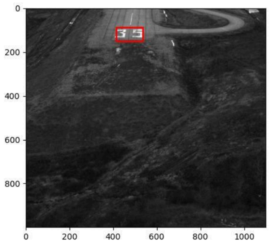
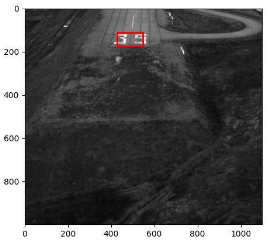
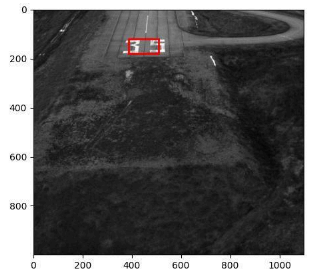
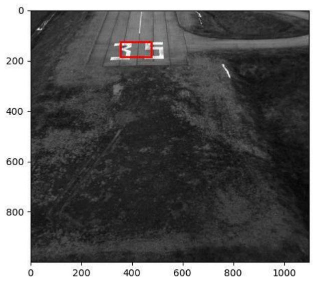
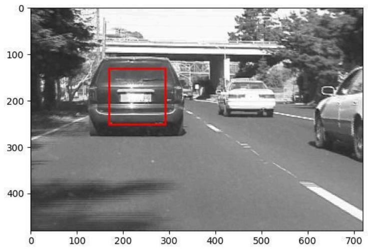
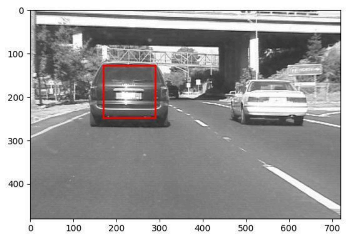
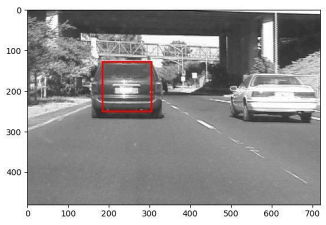
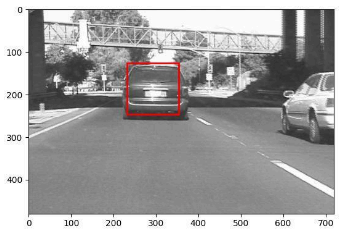
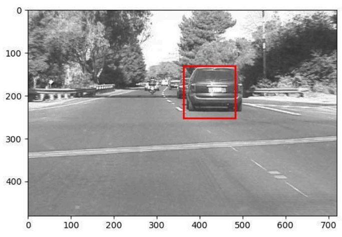
Computer Vision
Romil Nujella
B01099080

Q2.1) Lucas-Kanade Forward Additive Alignment with Translation

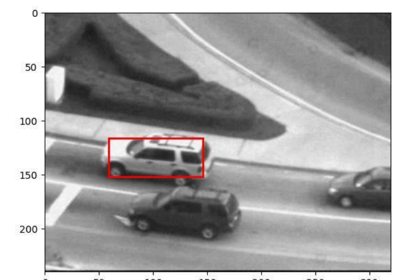
Landing Video: Frame 1, 15, 30, 40, 49

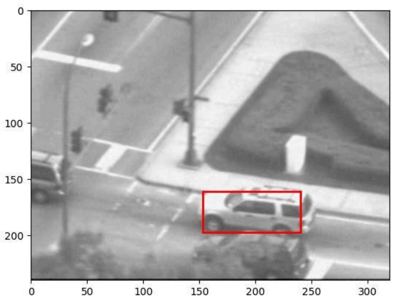






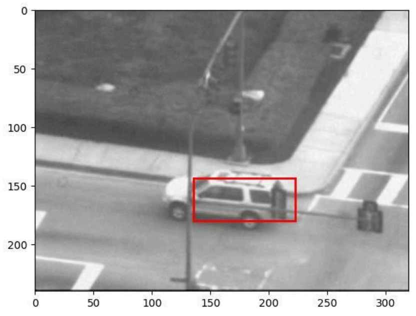
Car1 video: Frame 1, 75,100, 150, 250

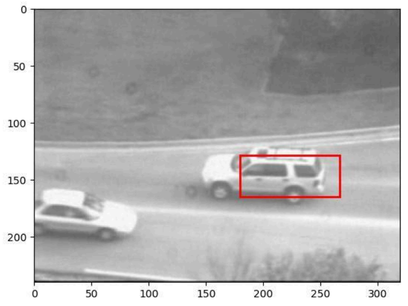






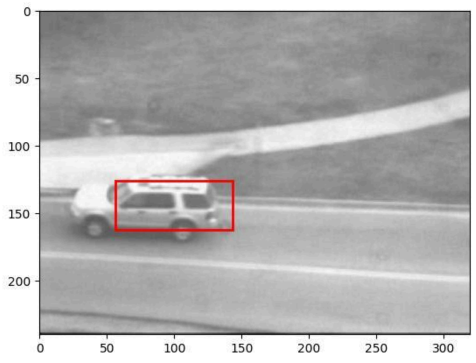
Car2 video: Frame 1, 50, 150, 300, 400





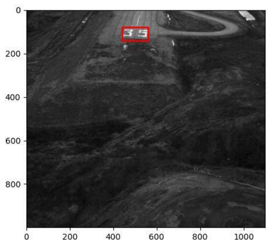


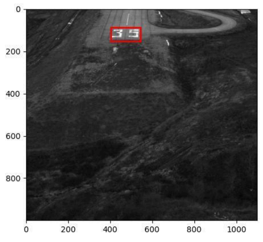


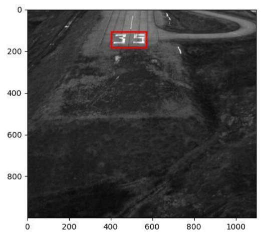


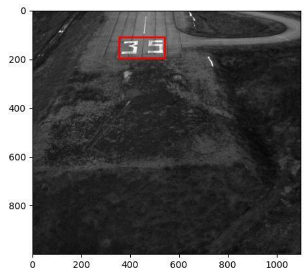
Q2.2) Lucas-Kanade Forward Additive Alignment with Affine Transformation

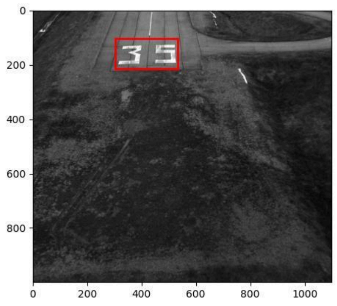
Landing Video: Frame 1, 15, 30, 40, 49



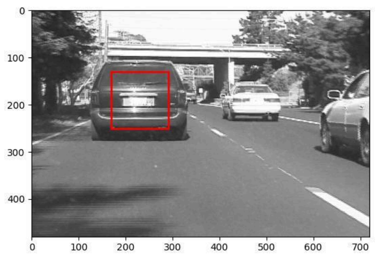


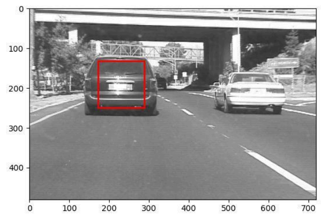


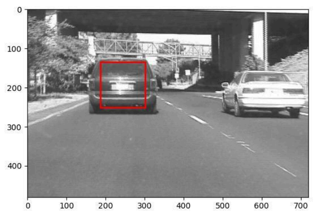


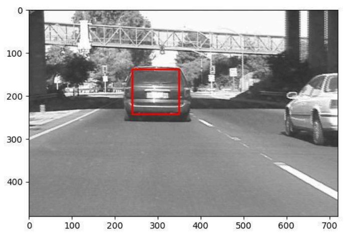


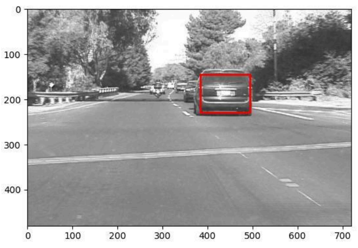
Car1 video: Frame 1, 75,100, 150, 250



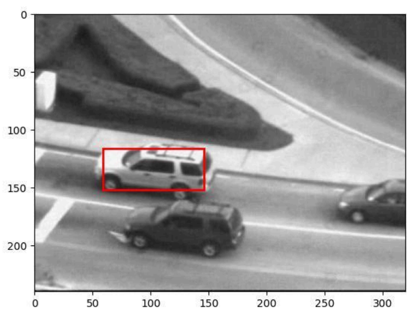


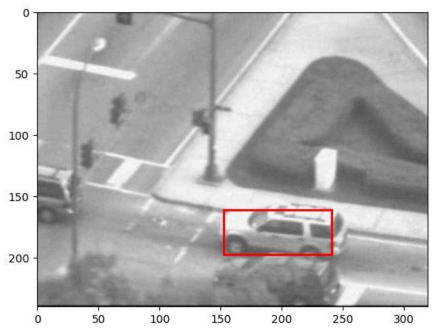


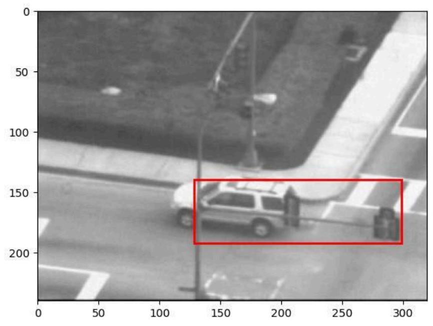


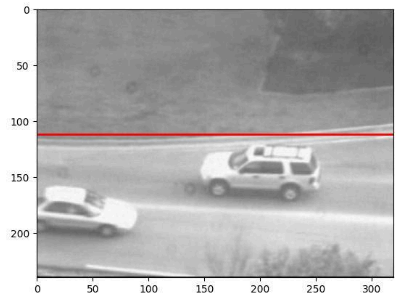


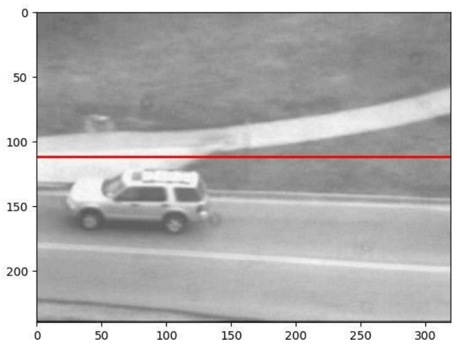
Car2 video: Frame 1, 50, 150, 300, 400



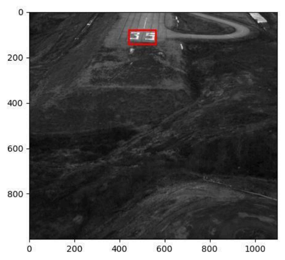
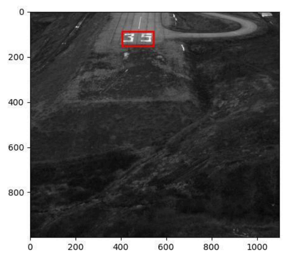
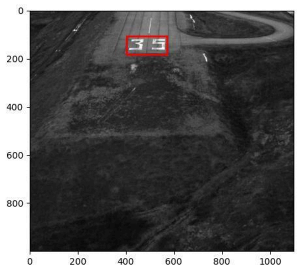
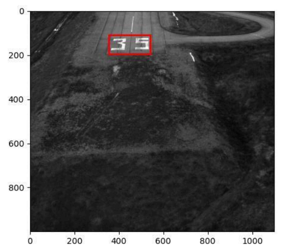
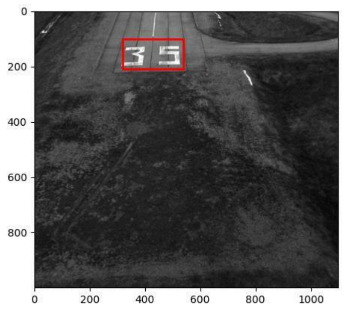




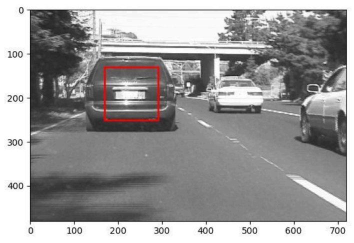
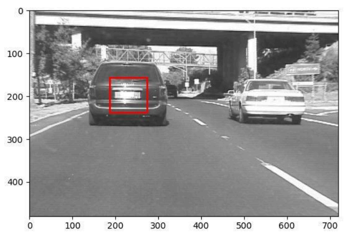
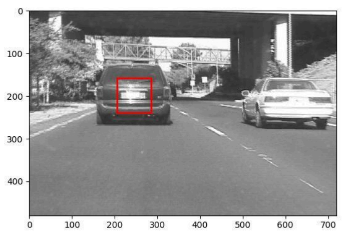
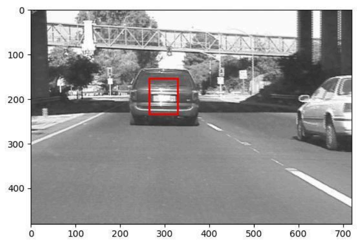
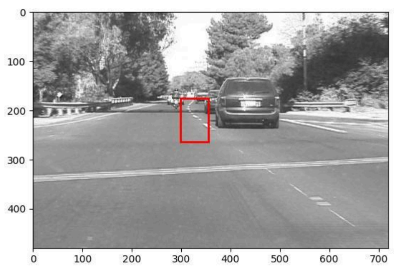




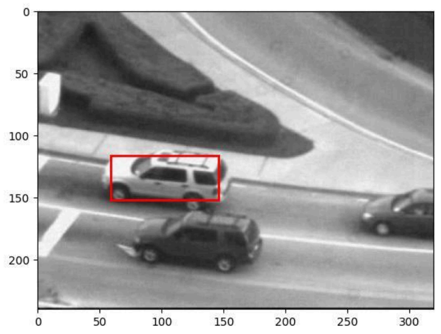
Q2.3) : Inverse Compositional Alignment with Affine Transformation
Landing Video: Frame 1, 15, 30, 40, 49

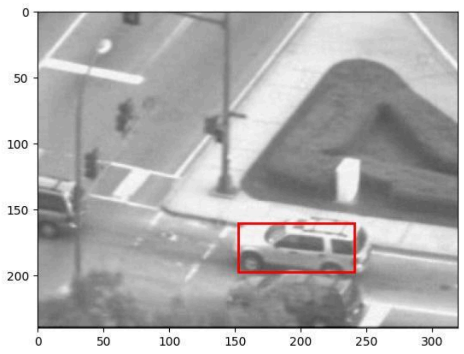






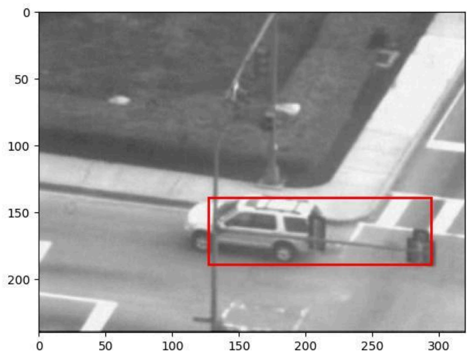
Car1 video: Frame 1, 75, 100, 150, 250

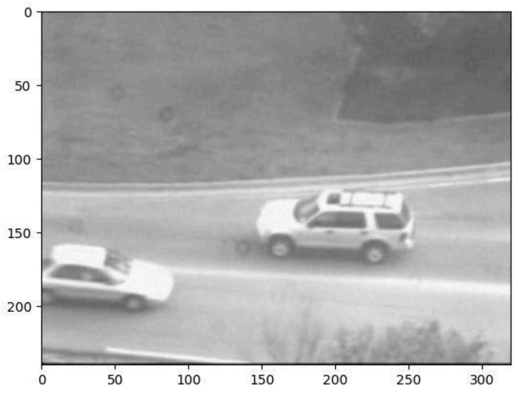






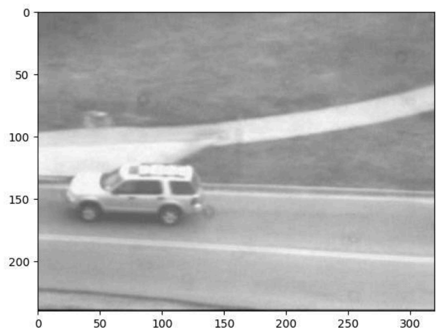
Car2 video: Frame 1, 50, 150, 300, 400











Q2.4)

Questions:

How do algorithms perform on each video?

- **Landing video**:
  The translation-only tracker (`test_lk.py`) starts reasonably but breaks down around frame 28 when the object scales significantly. The bounding box shrinks and no longer fully captures the number. In contrast, both affine trackers (`test_lk_affine.py` and `test_ic_affine.py`) perform consistently well, maintaining accurate alignment throughout the sequence.

- **Car1 video**:
  The translation-only tracker initially fails to enclose the entire car but starts performing better from frame 160 onward. The affine tracker struggles under low illumination (e.g., while passing under a bridge), causing deformation of the bounding box from frame 180–213. The inverse compositional tracker exhibits early instability, losing proper alignment from around frame 36 and drifting significantly by frame 173.

- **Car2 video**:
  The translation tracker is the most reliable here. It consistently tracks the car with reasonable accuracy, although the front of the car is not always enclosed. The affine tracker breaks down from frame 150 onward, resulting in oversized or disappearing bounding boxes. The inverse compositional method becomes unstable from frame 83, breaks fully by frame 289, and produces distorted shapes mid-sequence.

Which are the differences of the three algorithms in terms of performance and why do we have those differences?

- The **translation-only tracker** is more robust under gradual movement and lighting changes but cannot model scale, rotation, or shear.

- The **affine tracker** is capable of handling more complex transformations, which is helpful when objects rotate or scale. However, its increased flexibility makes it more sensitive to noise and changes in illumination.

- The **inverse compositional method** is more efficient computationally, since it precomputes gradients and the Hessian, but it is also more prone to numerical errors and drift when the residuals are large or the lighting varies significantly.

These differences stem from the complexity of the motion models used and how each algorithm deals with errors during tracking.

At what point does the algorithm break down and why does this happen?

- The **translation tracker** breaks down in sequences with large transformations (e.g., landing sequence after frame 28) because it assumes only translational motion.
- The **affine tracker** becomes unstable under poor lighting or complex deformation (e.g., car2 after frame 150) due to its sensitivity to noise and overfitting.
- The **inverse compositional tracker** tends to drift and break in the presence of strong residuals or occlusion (e.g., car1 after frame 167, car2 after frame 146) because its warp update assumes brightness constancy and small, smooth motion.

All algorithms fail when their underlying assumptions—such as small motion and consistent lighting—are violated, which is common in real-world sequences.

Q3.1) The Matthews-Baker method (also known as the inverse compositional Lucas-Kanade algorithm) separates tracking into two main phases: an initialization step where constant terms are precomputed, and runtime iterations where updates to the warp parameters are made. The method operates on a template image T with n pixels and an input image I with m pixels. The warp transformation W is defined by p parameters (for example, $p = 2$ for translation and $p = 6$ for affine warp).

Initialization Step – Precomputing Jacobian (J), Hessian (H), and H^{-1}

This step includes:

- Computing the image gradient of the template: $O(n)$
- Constructing the Jacobian matrix J of size $n \times p$: $O(n \cdot p)$
- Computing the Hessian matrix $H = J^tJ$: $O(n \cdot p^2)$
- Inverting the Hessian H^{-1} (a $p \times p$ matrix): $O(p^3)$

Total initialization cost : $O(n \cdot p + n \cdot p^2 + p^3)$

This cost is paid only once and does not depend on how many iterations are run.

Each iteration solves the update:

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

Breaking down the cost:

- Warping the image I at n pixel locations: $O(n)$
- Computing the error between warped image and template: $O(n)$
- Multiplying J^t ($p \times n$) by the error ($n \times 1$): $O(n \cdot p)$
- Multiplying H^{-1} ($p \times p$) by the result ($p \times 1$): $O(p^2)$

$O(n \cdot p + p^2)$

Comparison to Regular Lucas-Kanade (Forward Additive)

In the regular Lucas-Kanade method, all major components must be recomputed every iteration:

- Jacobian: $0(n \cdot p)$
- Hessian: $0(n \cdot p^2)$
- Inversion: $0(p^3)$

Total Per-Iteration Cost (Regular LK): $O(n \cdot p + n \cdot p^2 + p^3)$

Phase	Matthews-Baker (Inverse Comp.)	Regular Lucas-Kanade (Forward Additive)
Initialization	$O(n \cdot p + n \cdot p^2 + p^3)$	None
Each Iteration	$O(n \cdot p + p^2)$	$O(n \cdot p + n \cdot p^2 + p^3)$

Conclusion:

Matthews-Baker is computationally more efficient than the regular Lucas-Kanade method. By precomputing the Jacobian and Hessian, each iteration becomes faster, making it ideal for real-time applications or long video sequences. In contrast, the regular method incurs a heavier cost per frame due to recomputing gradients and matrix inversions.

Q3.2)

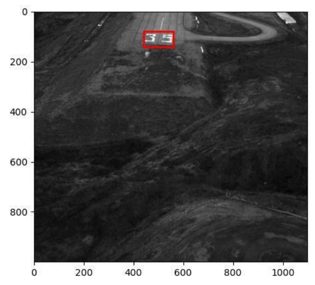
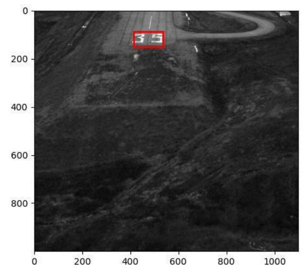
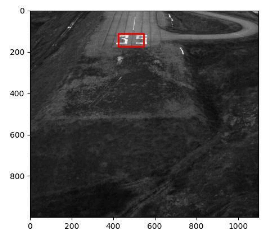
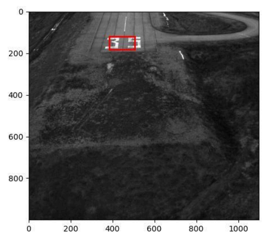
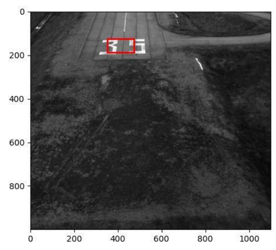
To address the issue of illumination sensitivity in the standard Lucas-Kanade tracker, I implemented a robust version using **M-estimators**. My `LucasKanadeRobust` function incorporates a custom `compute_weights` method that applies either **Huber** or **Tukey** weighting, based on residuals between the template and the current image. These weights help reduce the impact of pixels affected by lighting variation, such as shadows or glare.

The solution follows a **weighted least squares** approach. After computing residuals between warped image patches and the template, I calculate pixel-wise weights. The **Huber estimator** (with a default parameter a `= 1.339`) softens the influence of outliers, while the **Tukey estimator** (with a `= 4.685`) aggressively suppresses them. These weights are then used to re-weight the Jacobian and residuals before solving for the motion update.

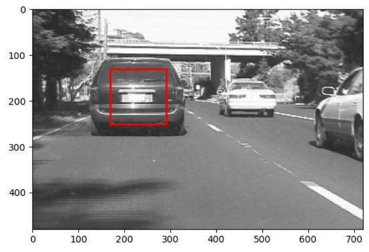
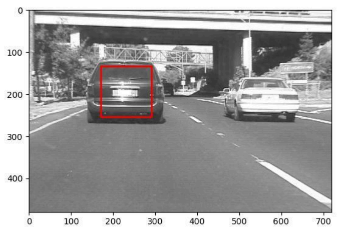
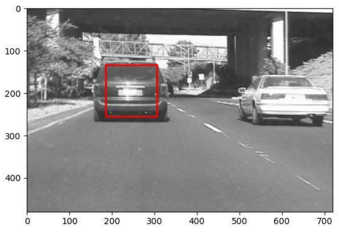
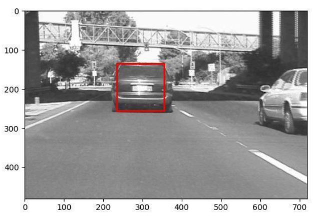
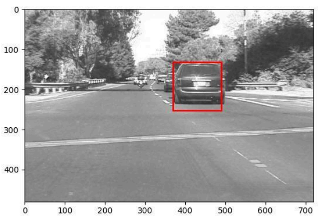
In addition to robustness, I incorporated a **pyramid-based tracking strategy** to handle large displacements. The tracker starts at a coarse resolution and refines the estimate through higher-resolution levels. At each level, the bounding box is appropriately scaled and optimized using the same robust weighted least squares process.

Testing on sequences with variable lighting confirmed that this method performs far more reliably than the original tracker. The robust version maintained lock on targets even when lighting changed significantly. In particular, **Tukey performed better for abrupt changes**, while **Huber handled smoother transitions more consistently**.

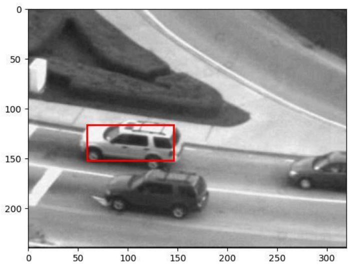
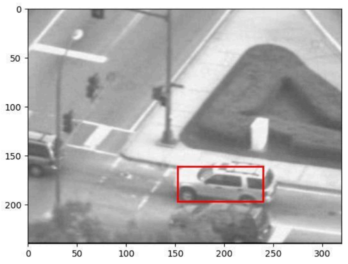
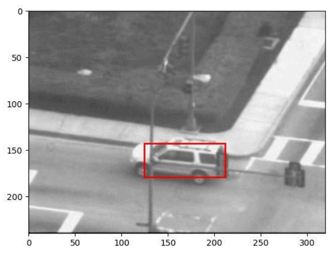
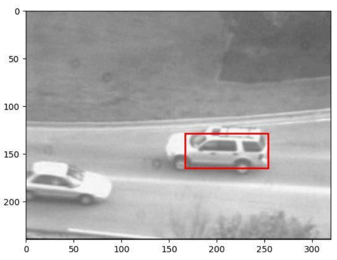
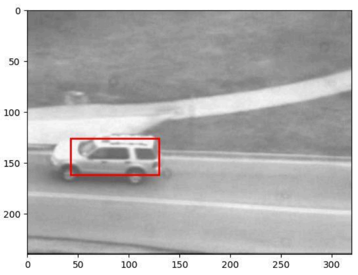
Landing Video: Frame 1, 15, 30, 40, 49

Car1 video: Frame 1, 75, 100, 150, 250

Car2 video: Frame 1, 50, 150, 300, 400

Q 3.3) To improve the tracker's performance on sequences with large object motion, I developed a pyramidal version of the Lucas-Kanade algorithm in the `LucasKanadePyramid` function. Rather than applying tracking directly on the original images, I construct **Gaussian pyramids** for both the template and current frame. Each pyramid level represents a progressively smaller and smoother version of the image, with the top being the most downsampled and the bottom the full-resolution image.

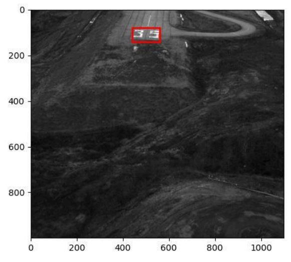
Tracking begins at the **coarsest level** of the pyramid, where the motion between frames appears minimal due to the reduced resolution. This makes it easier to estimate an initial rough alignment. The estimated motion vector from this level is then scaled and passed as the starting point to the next finer level.

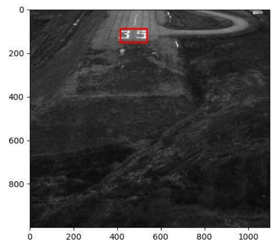
At each level, both the bounding box and motion vector are **rescaled** to fit the current image size. A set of Lucas-Kanade iterations are run to refine the alignment, and this process continues downward through the pyramid until the original image resolution is reached, producing a highly refined estimate of the target's movement.

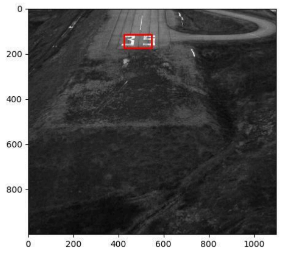
This hierarchical method helps address a major weakness of the original Lucas-Kanade tracker — its tendency to fail when the object shifts significantly between frames. By solving the problem incrementally across pyramid levels, the algorithm becomes more **robust to large displacements** and requires fewer iterations at high resolutions, improving both reliability and efficiency.

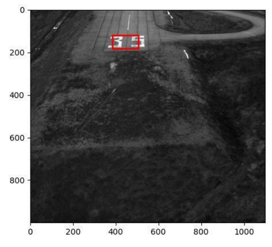
When tested on sequences involving **fast-moving or jumping objects**, my pyramid implementation consistently outperformed the standard tracker. It maintained stable and accurate tracking, even in situations where the original method would lose track of the target.

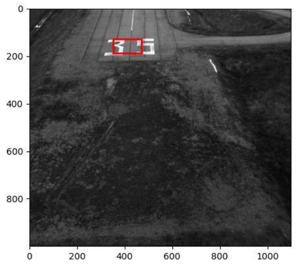
Landing Video: Frame 1, 15, 30, 40, 49



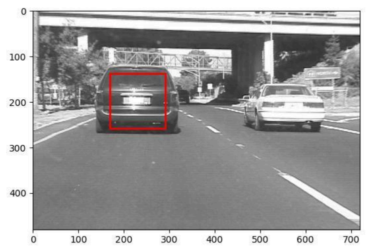


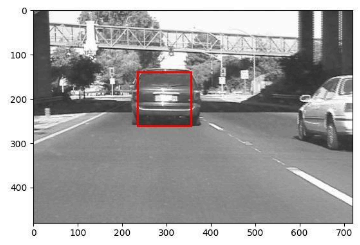
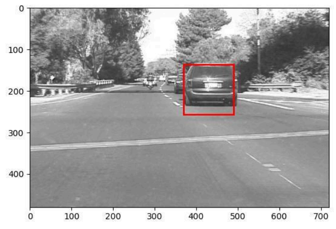






Car1 video: Frame 1, 75, 100, 150, 250



Car2 video: Frame 1, 50, 150, 300, 400